Université Paris Diderot
(Paris 7)

École Normale Supérieure
Équipe Crypto

Thèse de doctorat

Etudes d'hypothèses algorithmiques et attaques de primitives cryptographiques

Spécialité : Informatique

présentée et soutenue publiquement le 26 septembre 2011 par

Charles Bouillaguet

pour obtenir le grade de

Docteur de l'université Paris Diderot

devant le jury composé de

Responsable scientifique :	Pierre-Alain Fouque	(École Normale Supérieure, France)
Rapporteurs :	Joan Daemen	(ST Microelectronics, Belgique)
	Henri Gilbert	(ANSSI)
Examinateurs :	Hubert Comon-Lundh	(CNRS & École Normale Supérieure de Cachan, France)
	Arnaud Durand	(Université Paris 7, France)
	Ludovic Perret	(Université Paris 6, France)
	Vincent Rijmen	(Katholieke Universiteit Leuven, Belgique)
	Jacques Stern	(CNRS & École Normale Supérieure, France)

Remerciements

Je souhaite remercier tous ceux qui, par leur soutien, ont contribué à la réussite de cette thèse. Je remercie tout d'abord les membres du Jury d'avoir accepté d'être là aujourd'hui, en particulier ceux qui viennent de l'étranger. Je remercie chaleureusement Joan Daemen et Henri Gilbert, qui ont dû relire un manuscrit plutôt imposant en peu de temps, mais qui l'ont fait avec le sourire.

Cette thèse n'aurait pas été possible sans le soutien financier de la fondation EADS, envers qui je suis évidement très reconnaissant.

Matérialiste convaincu, je mesure le profit que j'ai pu tirer de l'environnement exceptionnel dont Pierre-Alain Fouque m'a fait bénéficier en me faisant entrer dans l'équipe de cryptographie du LIENS. Merci donc aux autres chercheurs de l'équipe qui ont toujours été à l'écoute, c'est-à-dire par ordre alphabétique Michel Abdalla (fournisseur officiel de café) Bruno Blanchet, Vadim Lyubashevsky, David Naccache et les Men-in-Black, Phong NGuyen, David Pointcheval (pour avoir en plus supporté les inconvénients d'être mon directeur de thèse officiel) et Damien Vergnaud (pour avoir prêté son appartement et son frigo). L'ambiance de cette équipe ne serait pas ce qu'elle est sans ses thésards et post-docs, et pour cela je remercie sans ordre particulier Léo Ducas pour nombre de séances de brainstorming souvent peu frucueuses, Qiang Tang, Georg Fuchsbauer et Steven Gay pour le ping-pong de l'après-midi en 2007, Malika Izabachène, Céline Chevalier et Yuan-Mi Chen pour être sympas, Aurore Guillevic (made in Bretagne) pour les mangas, Aurélie Bauer pour être la Gentille Organisatrice et pour avoir créé un séminaire qui a porté brièvement son nom en signe de reconnaissance éternelle, Olivier Blazy pour les produits dérivés Portal, Jérémie Jean parce qu'il a piraté "Questions pour un champion" ou autre un jeu du même style, Dario Fiore et Mario Streffler parce que ce sont de beaux gosses, David Cade parce que je n'avais jamais vu quelqu'un jouer à une sorte de Guitar-Hero en entrant en transe au cours de la partie, et enfin Miriam Paiola parce qu'elle est souriante et que comme elle est assise juste en face de moi c'est elle que je regarde tout le temps.

Je remercie plus particulièrement les thésards avec qui j'ai eu l'occasion de collaborer. Merci donc à Gaëtan Leurent, mon *sempai* qui plaçait *de facto* la barre assez haut, à Sébastien Zimmer pour ses petites siestes qui justifiaient les miennes et pour son aide tranquille mais efficace lors de mes débuts, à Elena Andreeva pour avoir initié une collaboration internationale fructueuse et amicale, à Mehdi Tibouchi qui a été mon oracle pour les problèmes de math qui me dépassaient, à Patrick Derbez pour sa créativité et sa souplesse d'esprit (aussi avec les horaires et la rigueur mathématique), et enfin à Joana Treger-Marim pour sa détermination à l'épreuve des balles, sa rigueur et sa jovialité.

Dans l'ensemble, je n'ai lu en détail que les thèses de Vivien Dubois, de Sébastien Kunz-Jacques, de Magali Turrel-Bardet et de Gilles Macario-Rat. Elles ont été pour moi des documents de travail utiles, sur le sérieux desquels je pouvais compter. J'espère juste que la mienne sera aussi bien.

Je remercie Bo-Yin Yang ainsi que Ludovic Perret et Jean-Charles Faugère, qui ont tous les trois instantanément accepté de participer et de s'investir dans les projets que je leur proposais. Antoine Joux, Vincent Rijmen et Adi Shamir et m'ont tous les trois fait le plaisir et l'honneur de travailler avec moi, et je les remercie de la facilité avec laquelle ils partagent leur expérience et en font profiter leur entourage. Au gré des rencontres, j'ai pu en bénéficier à plusieurs reprises.

Cette thèse n'aurait pas non plus été possible sans le soutien indéfectible de l'équipe administrative et technique de l'ENS. Je remercie chaleureusement Jacques Beigbeder, Nasser Bacha et Ludovic Ricardou pour le dévouement, le sérieux et l'efficacité dont ils ont toujours fait preuve face à mes demandes ou à mes appels au secours. Ils ne seront jamais assez remerciés. Merci aussi à Michelle Angely, Lise-Marie Bivard, Isabelle Delais, Joëlle Isnard et Valérie Mongiat, sans lesquelles je n'aurais jamais rien fait. Elles ne seront jamais assez remerciées non plus.

Je remercie aussi Orr Dunkelman, qui en plus de m'avoir fait bénéficier de son talent et de son sens de l'humour sur une base quotidienne lors de son séjour à l'ENS et même au-delà, est aussi un ami personnel. Finalement, je remercie Pierre-Alain Fouque, très certainement la personne qui a joué le rôle le plus important pour moi dans ces années de thèse. Il a supporté de m'avoir dans les jambes tous les jours pendant pratiquement trois ans avec un calme olympien. Il m'a guidé, m'a proposé des pistes utiles qui ont pu déboucher sur des résultats, s'est préoccupé de l'avancement et de l'aboutissement de mes travaux sans être ni laxiste ni autoritaire. Le spectre large de ses domaines de compétence m'a permit de travailler des sujets (multi)variés, et n'ayant même rien à voir entre eux. En gros, il est responsable des résultats que j'ai pu obtenir.

J'en vient à la partie la plus difficile, celle où il faut que je remercie mon amoureuse. Pour les polémiques de mauvaise foi sur "est-ce que la cryptologie est plus ou moins importante que l'influence du changement climatique sur le régime de la Loire moyenne", je ne sais pas si je dois te remercier. Néanmoins, tu as toujours supporté que je me passionne pour des trucs bizarres, et tu as toujours recollé les morceaux quand la déception succédait à l'enthousiasme prématuré. J'ai commencé cette thèse avant de te rencontrer, mais maintenant la thèse est finie et toi, tu es toujours là. Bref, la science c'est bien, mais toi c'est mieux.

Sommaire

Table des Matières

Chapitre 1

Introduction

Le premier chapitre de cette thèse reprend intégralement le texte de la brochure Sommes-nous prisonniers des codes secrets*, que j'ai écrite avec Pierre-Alain Fouque, et qui est parue aux Éditions du Pommier en avril 2011. Pour cette raison, il ne peut être distribué sur internet sans violer les droits de mon éditeur...*

Chapitre 2

Présentation de mes Travaux

All Computer Science is Algorithms

Donald Knuth

Les travaux que j'ai effectués durant ma thèse relèvent surtout du domaine de la cryptanalyse. Ils ne consistent donc pas tant en la construction de nouveaux mécanismes cryptographiques qu'en l'étude et la "démolition" de mécanismes existants.

J'ai commencé ma vie cryptographique à un moment où la concomitance de nouvelles attaques sur les fonctions de hachage usuelles décidait le NIST (le *National Institute for Standards and Technology*) à lancer une compétition afin de désigner une nouvelle fonction de hachage standardisée. Les nouvelles attaques et la compétition SHA-3 ont créé un regain d'intérêt pour la construction et l'étude de fonctions de hachage. Dans une première phase, j'ai donc travaillé sur quelques nouveaux modes opératoires qui venaient d'être conçus pour la compétition, avec la perspective d'éviter certaines nouvelles attaques. Je me suis intéressé en particulier aux attaques en seconde préimage, et j'en ai conçu une nouvelle en utilisant des techniques originales. J'ai pu l'appliquer à une grande variété de constructions, y compris à un nouveau mode opératoires conçu par Ron Rivest spécifiquement pour résister à ce genre d'attaques.

Au début de mes travaux, mon équipe d'accueil a créé l'évènement en présentant une attaque spectaculaire contre SFLASH, un schéma de signature "multivarié" qui était considéré comme sûr et était en passe d'être standardisé. J'ai par la suite été entraîné dans le courant de la cryptanalyse de schémas multivariés, parallèlement à mes travaux sur les mode opératoires de fonctions de hachage. C'est dans ce contexte que je me suis intéressé plus précisément à une des hypothèses algorithmique "standard" de la cryptographie multivariée, le problème de *l'Isomorphisme de Polynômes*. J'ai conçu plusieurs nouveaux algorithmes pour résoudre différentes variantes de ce problème, parfois très efficacement.

Enfin, durant ma thèse, un des principaux évènements cryptographiques a été la découverte de nouvelles attaques sur l'AES, notamment une attaque à clef liée plus rapide que la recherche exhaustive sur l'AES-256, ainsi qu'une attaque marginalement plus rapide que la recherche exhaustive sur l'AES-128. Je me suis donc intéressé à mon tour à l'AES, mais plutôt que d'attaquer le plus grand nombre de tours possible, je me suis concentré sur la recherche d'attaques très efficaces sur un petit nombre de tours. Ces attaques, bien qu'elles ne remettent pas en question la sécurité de l'AES, peuvent parfois s'appliquer à d'autres primitives cryptographiques qui utilisent des composants de l'AES. A cette fin, j'ai participé à la construction d'outils automatiques de recherche d'attaques, qui ont trouvé des attaques ayant échappées aux cryptanalystes.

Je me suis donc penché à la fois à la cryptanalyse symétrique (fonctions de hachage, AES) et asymétrique (schémas multivariés). J'ai logiquement été amené à manipuler des méthodes de cryptanalyse variées, à la fois statistiques (probabilités, paradoxe des anniversaires, comportement aléatoire) et algébriques (résolutions de systèmes d'équations polynomiaux, algèbre linéaire, calcul formel). La cryptanalyse symétrique est plus naturellement statistique, et la cryptanalyse asymétrique plus naturellement algébrique, car les schémas cryptographiques à clef secrète n'ont pas de structure algébrique, contrairement à la plupart des schémas à clef publique. Mes résultats les plus intéressants ont cependant été obtenus en utilisant des méthodes algébriques en cryptanalyse symétrique, et des méthodes statistiques en cryptanalyse multivariée.

Ma thèse s'articule en trois parties, en suivant la distinction thématique soulignée plus haut. Toutefois, il semblait plus logique de présenter mes travaux sur l'AES avant mes travaux sur la cryptanalyse multivariée, afin que mes travaux se présent selon la double progression :

$$\begin{aligned} \text{symétrique} &\rightarrow \text{asymétrique} \\ \text{statistique} &\rightarrow \text{algébrique.} \end{aligned}$$

1: **function** Merkle-Damgård (M)
2: **Étendre** puis **découper** le message M en r blocs $m_0, \ldots, m_r$ de m bits chacun.
3: **soit** h_{-1} l'état interne initial (traditionnellement nommé IV).
4: **pour chaque** bloc de message m_i, calculer $h_i = f(h_{i-1}, m_i)$.
5: **renvoyer** $H^f(M) = h_r$.
6: **end function**

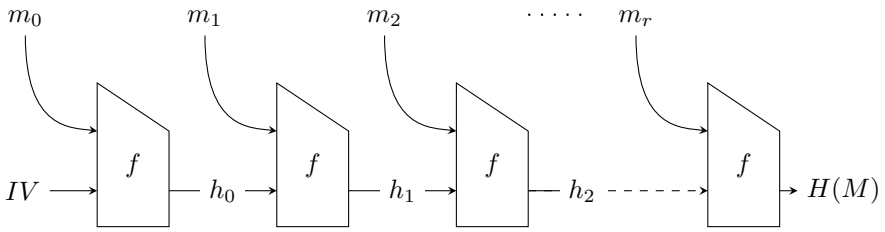


Figure 2.1 – Le mode opératoire de Merkle-Damgård. La fonction de compression f, compresse $n + m$ bits en n bits.

2.1 Etude de modes opératoires pour les fonctions de hachage

Depuis les années 1990 (et même avant, si on y regarde bien), la plupart des fonctions de hachage itèrent une fonction de compression via le mode opératoire de Merkle-Damgård (cf. Figure 2.1). Il est simple, efficace, et bénéficie d'une forme intéressante de sécurité prouvée. Cependant, en 2004, 2005 et 2006, trois nouvelles attaques, dues à Joux [Jou04], Kelsey et Schneier [KS05], ainsi que Kelsey et Kohno [KK06], sont apparues, visant spécialement ce mode opératoire. Ce sont donc des attaques qui s'appliquent automatiquement à (presque) toutes les fonctions de hachage existantes, et en tout cas à toutes celles qui reposent sur la construction de Merkle-Damgård.

L'attaque en seconde préimage de Kelsey et Schneier est probablement la plus "ennuyeuse" des nouvelles attaques génériques, car elle vise une des propriétés de base des fonctions de hachage cryptographiques, qui est par exemple nécessaire pour les schémas de signature numérique. De plus, elle remettait brutalement en question la croyance que le mode opératoire de Merkle-Damgård offrait un niveau de sécurité de l'ordre de 2^n évaluations de la fonction de compression face aux attaques en seconde préimage, puisqu'elle n'en requiert que $2^n/\ell$, où ℓ désigne la taille du message donc on cherche une seconde préimage.

2.1.1 Attaque contre le Dithered Hashing de Rivest [ABF+08]

Pour empêcher l'attaque de Kelsey et Schneier, Ron Rivest a proposé une modification simple du schéma de Merkle-Damgård, qui consiste à ajouter une entrée supplémentaire à la fonction de compression, qu'on appellera le *dithering*. Le dithering est une technique bien connue qui consiste à introduire volontairement une certaine quantité de "bruit" dans un signal afin d'éviter des effets de seuil ou la formation de motifs. En infographie, la technique est utilisée avec succès pour rendre des images avec peu de couleurs (cf. figure 2.2). L'idée de Rivest était d'introduire un "bruit", qui perturberait le processus de hachage. Si $\mathbf{z}[0], \mathbf{z}[1], \ldots$ est une "séquence de dithering" bien choisie, cela revient à définir :

$$h_i = f(h_{i-1}, m_i, \mathbf{z}[i])$$

image originale | noir et blanc par arrondi | noir et blanc par dithering

Figure 2.2 – Illustration du "dithering" en infographie

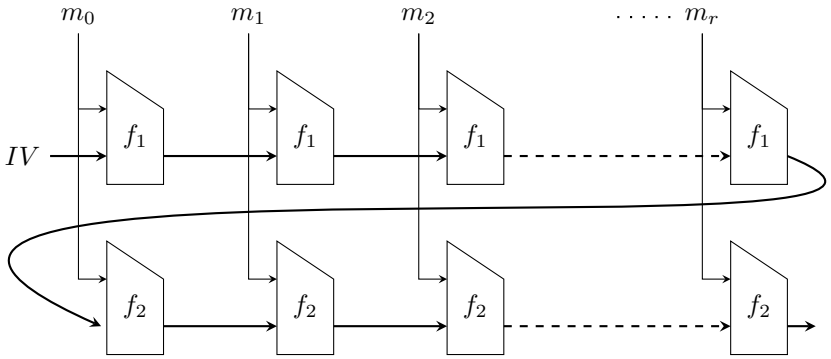


FIGURE 2.3 – Merkle-Damgård-Again.

L'attaque de Kelsey et Schneier, repose dans le fond sur le fait qu'on peut former un *message expansible* en trouvant quelques collisions, c'est-à-dire une famille de messages de toutes les tailles produisant le même état interne. La présence du dithering rend ceci impossible, car elle rend le haché d'un bloc de message dépendant non seulement de l'état interne précédent, mais aussi de sa position dans le message global. Même s'il était possible de produire un message de n'importe quelle taille produisant le même haché, la séquence de dithering serait décalée et viendrait tout perturber.

Pour attaquer cette construction, nous forgeons des seconde préimage M' qui ne diffèrent de M que sur un très petit nombre de blocs, typiquement $n/3$. Pour cela, nous utilisons la technique du "herding" de Kelsey et Kohno [KK06]. Nous repérons ensuite la séquence de taille $n/3$ qui revient le plus souvent dans $\mathbf{z}$, et nous appliquons notre attaque à toutes les positions correspondantes. (Mal)heureusement, la séquence $\mathbf{z}$ choisie par Rivest possède un certain nombre de facteurs fréquents, et notre attaque est donc à peine plus coûteuse que celle de Kelsey et Schneier.

2.1.2 Attaque contre Merkle-Damgård-Again [ABDK09]

Une manière simple mais efficace de rendre une fonction de hachage itérée plus résistante consiste à randomiser son état initial. Cela interdit déjà un certain nombre de calculs *offline*. Plus sophistiqué, il est possible de hacher le message une première fois, de se servir du haché pour redéfinir l'état initial, puis de re-hacher le message une seconde fois. Nous appelons le mode opératoire résultant *Merkle-Damgård-Again* (il est illustré figure 2.3), et on peut le définir par :

$$H'(M) = H(H(IV, M), M)$$

Cette construction (qui fait partie du folklore) n'est pas facile à attaquer, car modifier le message pour tenter de contrôler ce qui se passe dans l'une des deux itérations aboutit généralement à un résultat incontrôlable dans l'autre itération.

J'ai toutefois trouvé une attaque en seconde préimage contre cette construction folklorique, pour un coût à peine supérieur à celui de l'attaque de Kelsey et Schneier sur Merkle-Damgård. La difficulté consiste à attaquer la seconde itération tout en garantissant que la première produit toujours le même résultat. Cela est encore une fois possible à l'aide de techniques de "herding" qui permettent de contrôler la valeur de chainage dans la première itération. A partir de là, on construit des multicollisions à la Joux, qui donneront de la liberté dans la seconde itération tout en garantissant que le résultat de la première reste constant. Il est ensuite possible de rejouer l'attaque en seconde préimage utilisée contre le *dithered hashing* dans la seconde itération, "par dessus" les multicollisions.

2.1.3 Preuves de résistance aux attaques en seconde préimage

Face à l'apparition de l'attaque en seconde préimage de Kelsey et Schneier, ainsi que de celle que j'ai proposée, il est naturel de se demander s'il existe des modes opératoires qui résistent *prouvablement* à ces attaques. Il existait déjà des familles de fonctions de hachage qui résistent à des formes d'attaques en seconde préimage (les Universal One-Way Hash Functions). L'une des plus efficaces est due à Shoup [Sho00a], et a un niveau de sécurité prouvé de $2^n/\ell$ évaluations de la fonction de compression. Plus récemment, Andreeva d'un côté [AP08] et Yasuda de l'autre [Yas08] ont proposé de nouvelles variantes du mode de Merkle-Damgård bénéficiant de résistance prouvée aux attaques en seconde préimage. Mis à part la dernière, ces constructions sont assez peu pratiques, car il s'agit de *familles* de fonctions de hachage, et non d'une unique fonction de hachage.

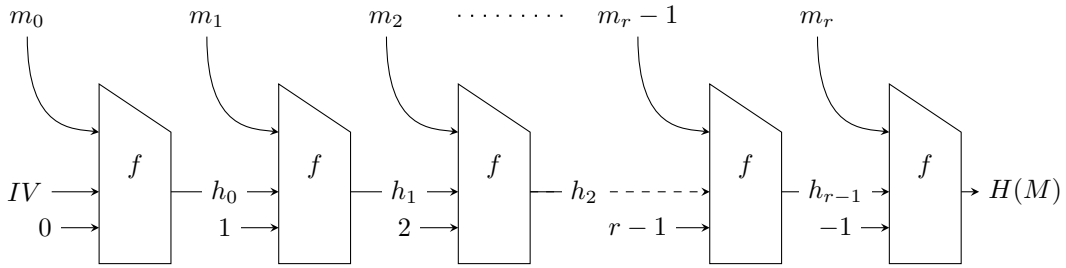


Figure 2.4 – Le mode opératoire Haifa.

2.1.3.1 Sécurité dans le modèle de l'oracle aléatoire [BFZ10]

D'un autre côté, Biham et Dunkelman ont proposé une autre variante simple de Merkle-Damgård, qu'ils ont nommée Haifa (et qui est illustré figure 2.4). Il s'agit (essentiellement) d'ajouter un compteur de tours à la fonction de compression :

$$h_i = f\left(h_{i-1}, m_i, i\right)$$

Biham et Dunkelman ont affirmé que calculer des secondes préimages sur Haifa était aussi dur que la recherche exhaustive, mais sans présenter de preuve. J'ai donc mis au point une preuve élémentaire de résistance aux secondes préimages pour Haifa : sous l'hypothèse que la fonction de compression est idéale, alors trouver une seconde-préimage sur son Haifa-itération nécessite environ 2^n évaluations de la fonction de compression. L'argument clef est que chaque fois que l'adversaire évalue la fonction de compression il doit fournir une valeur i du compteur. Dès lors, le résultat de la fonction de compression n'est exploitable que s'il s'agit précisément de h_i, et cela arrive avec probabilité 2^{-n} si la fonction de compression est aléatoire. La même technique de preuve permet au passage de prouver que l'attaque de Kelsey et Schneier est optimale si la fonction de compression est idéale.

L'hypothèse de travail (la fonction de compression est idéale) est très forte, mais il n'est pas évident de l'affaiblir. D'une part, si la fonction de compression est trop faible, le résultat peut ne plus être vrai. D'autre part, cela nécessiterait alors de changer de technique de preuve, et le résultat cesserait probablement d'être élémentaire.

2.1.3.2 Une perte de sécurité est inévitable dans le modèle standard

La quantité $2^n/\ell$ est déjà apparue plusieurs fois dans ce résumé : c'est le coût de l'attaque de Kelsey et Schneier, ainsi que de celle que j'ai découverte. C'est aussi le niveau de sécurité prouvée obtenus dans le modèle standard (*i.e.*, sans supposer que la fonction de compression est idéale) par les constructions de Shoup, Andreeva et Yasuda. En fait, il ne s'agit pas d'une coincidence, car ces trois constructions utilisent la même méthodologie pour démontrer leur résistance aux attaques en seconde préimage : il s'agit d'une preuve par réduction en boite noire à la sécurité de la fonction de compression. Plus précisément, il est démontré dans les trois cas qu'un adversaire en seconde préimage contre l'itération avec probabilité de succès ε peut être transformé en un adversaire en seconde préimage contre la fonction de compression avec probabilité de succès ε/ℓ.

Il est possible de démontrer que cette "perte de sécurité" d'un facteur ℓ est inévitable, et ce pour pratiquement n'importe quelle construction (dans certaines limites raisonnables). Pour cela, je montre qu'il est possible de construire un environnement autour de la réduction (c'est-à-dire un adversaire contre l'itération et une fonction de compression), dans lequel il est facile de montrer que la réduction ne peut réussir qu'avec probabilité $1/\ell$.

2.2 Recherche automatiques d'attaques sur des versions réduites de l'AES

La plupart du temps, lorsqu'ils sont confrontés à un système de chiffrement par bloc, les cryptanalystes cherchent à casser le plus grand nombre de tours possible, tout en utilisant un peu moins de temps que la recherche exhaustive et un peu moins de données que l'ensemble du *codebook*. Pour aller un peu plus loin, ils se placent parfois dans des modèles plus puissant : chiffrés choisis, clefs liées, sous-clefs liées, clef connue, etc. La signification pratique de ces modèles plus forts reste sujette à caution.

Dans ma thèse, j'ai choisi de m'intéresser à un aspect relativement inexploré de la sécurité des système de chiffrement par blocs : la sécurité d'un petit nombre de tours face à des adversaires disposant de peu de données. Les attaques avec peu de données peuvent parfois servir de "brique de base" dans des attaques plus

sophistiquées (on l'a vu récemment avec la cryptanalyse de GOST [Iso11], où une attaque sur les 32 tours est construite à partir d'une attaque sur 8 tours avec deux paires clair/chiffré connues).

La faible quantité de donnée qu'on se restreint à utiliser impose interdit l'usage des techniques de cryptanalyse "statistique" habituelles, telles que la cryptanalyse linéaire ou différentielle. Les attaques algébriques sembleraient à priori plus adaptées, mais elles n'ont jamais réussi à casser même un seul tour d'AES. J'ai donc cherché des attaques de type "guess-and-determine" dans un premier temps, puis des attaques de type "meet-in-the-middle", qui semblent a priori plus puissantes que les premières. Trouver ces attaques à la main est difficile, car il y a beaucoup de "possibilités" à explorer, et il est facile de se tromper à un moment donné. Il était donc naturel de concevoir des outils automatiques pour chercher ces attaques à ma place.

Ces outils travaillent à partir d'une description de l'AES sous forme d'un système d'équations linéaires avec une bijection inerte qui représente la boite-S. Ils exploitent de façon inattendue la structure algébrique de l'AES, notamment le fait que toutes les opérations admettent une description naturelle sur $\mathbb{F}_{2^8}$.

2.2.1 Recherche automatique d'attaques "Guess-and-Determine" [BDF11]

Les attaques les plus facile à chercher de manière automatique sont les attaques "guess-and-determine". Dans leurs versions les plus simples, ces attaques consistent à énumérer toutes les valeurs possibles d'une partie des secrets inconnus, puis d'utiliser la description du mécanisme cryptographique pour en déduire la clef secrète, et enfin vérifier si cette clef est bien la bonne. J'ai donc conçu un outil qui énumère tous les sous-ensembles de l'état interne, et détermine si la connaissance de chacun de ces sous-ensembles permet de déduire la clef secrète, et si oui de quelle manière. Pour cela, l'outil cherche si des combinaisons linéaires des équations permettent d'exprimer les parties inconnues de l'état interne en fonction de celles qu'on énumère. Ce processus est exponentiel, mais il suffit de l'exécuter une seule fois pour en déduire un programme qui effectue la récupération de clef.

Ce programme a trouvé des attaques sur 1, 2 et 3 tours complets de l'AES avec une seule paire clair/chiffré connue. Sur 1 tour, l'attaque trouvée, d'une complexité de 2^{40} opérations, est plus rapide que la meilleure connue (qui était en 2^{48}). L'attaque sur 3 tours, d'une complexité de 2^{120} opérations, était inconnue auparavant.

Cet outil ne parvient pas véritablement à utiliser la donnée de plusieurs paires clair/chiffré, car il n'est pas capable d'utiliser l'idée que si les différence sont connues à l'entrée et à la sortie d'une boite-S, alors les valeurs peuvent être déduites facilement.

2.2.2 Recherche automatique d'attaques "Meet-in-the-Middle" [BDF11]

En partant de ce que la conception de ce premier outil m'avait appris, j'ai pu participer à la construction d'un second outil plus puissant, qui cherche des attaques de type Meet-in-the-middle. Cet outil part de l'idée que s'il existe deux algorithmes permettant d'énumérer efficacement les valeurs que peuvent prendre deux ensembles A et B de variables apparaissant dans la primitive de chiffrement, alors il est peut-être possible d'énumérer $A \cup B$, en utilisant les équations qui lient ensemble les variables de A et B pour éliminer les combinaisons "impossibles". La procédure de recherche construit donc par "saturation" des algorithmes qui énumèrent ensembles de variables de plus en plus gros, jusqu'à ce que l'un d'entre eux contienne la clef secrète.

Un des avantages de ce nouvel outil par rapport au précédent est qu'il prend automatiquement en compte les propriétés "différentielles" de la boite-S. Cela lui permet d'exploiter efficacement la présence de plusieurs messages. Cet outil a amélioré (presque) toutes les attaques connues sur un petit nombre de tours d'AES, qu'elles aient précédemment été trouvée manuellement ou automatiquement.

2.2.3 Application

Résultats sur des versions réduites de l'AES [BDD+10]. Ce deuxième outil a trouvé beaucoup d'attaques sur l'AES lui-même. Il a amélioré l'attaque sur 1 tour avec un seul message (en 2^{32} opérations). Il a trouvé une attaque marginalement plus rapide que la recherche exhaustive sur 5 tours avec une seule paire clair/chiffré. Il a également trouvé une attaque en 2^{32} opérations sur 2 tours avec deux paires clair/chiffré connues, et en 2^8 opérations avec deux paires clair/chiffré choisies. Il a enfin trouvé une attaque contre 4 tours avec 4 paires clair/chiffré choisies en 2^{32} opérations. Dans un autre registre, il a amélioré l'attaque par faute de Piret et Quisquater [PQ03], en réduisant la complexité de 2^{32} opérations à 2^{24}.

Résultats sur d'autres primitives cryptographiques. L'outil a également donné des résultats sur un MAC basé sur l'AES, Pelican-MAC [DR05b] et sur un système de chiffrement par flot également basé sur l'AES, LEX [Bir05].

Sur Pelican-MAC, l'outil a aidé à la construction de la meilleure attaque connue à ce jour, en mettant au point une procédure qui révèle l'état interne de la fonction en temps 2^{32} une fois qu'une collision sur l'état interne est trouvée. Cela est obtenu en résolvant l'équation $AES_4(x) \oplus AES_4(x \oplus \Delta_i) = \Delta_o$ en x, où AES_4 dénote 4 tours complets d'AES sans addition de clef. L'attaque résultante a une complexité de 2^{64} requêtes au MAC (pour obtenir la collision).

Sur LEX, l'outil a également été utilisé pour construire une meilleure attaque. Il a d'abord aisément retrouvé tout seul la meilleure attaque connue, de complexité environ 2^{100} opérations, puis, avec l'intervention d'un utilisateur, il a produit une attaque de complexité 2^{80}, qui utilise 80 tera-octets de key-stream.

2.3 Étude de problèmes difficiles en cryptographie multivariée

La cryptographie multivariée est une appellation qui englobe l'ensemble des schémas (essentiellement à clef publique) donc la sécurité repose notamment sur la difficulté de la résolution de systèmes d'équations polynomiaux multivariés sur un corps fini. Quand ces polynômes sont de degré deux, le problème s'appelle MQ, pour Multivariate Quadratic.

2.3.1 Nouvel algorithme de résolution d'équations polynomiales sur $\mathbb{F}_2$ [BCC+10]

Durant mes travaux, j'ai été amené à m'interroger sur la difficulté *pratique* du problème MQ, afin de déterminer si des jeux de paramètres concrets pour certains schémas pouvaient être cassés avec les moyens matériels restreints à ma disposition. Sur $\mathbb{F}_2$, la méthode de résolution la plus efficace de MQ reste la recherche exhaustive, qui nécessite 2^n opérations s'il y a n variables. C'est en essayant de programmer une implémentation efficace que j'ai aperçu la possibilité d'améliorations algorithmiques. J'ai donc conçu un algorithme de recherche exhaustive extrêmement efficace, puisqu'il permet de tester entre 2 et 3 solutions *par cycle du processeur* sur chacun des coeurs d'une machine moderne. Ce nouvel algorithme a également été implémenté sur GPU, permettant d'obtenir des performances plus élevées pour un coût plus faible qu'avec les processeurs habituels. Il est possible de résoudre une instance de MQ avec $q = 2$ et $n = 64$ en un jour pour environ $7500 sur le *cloud*. La technique algorithmique s'adapte également à des systèmes de degré plus grand.

2.3.1.1 Attaque contre un schéma d'identification de Patarin [BFFP11]

En 1996, Patarin avait proposé de construire un schéma d'identification à clef publique basé sur la difficulté de *l'équivalence simultanée de paires de formes quadratiques* (QFSE pour faire court). Il s'agit, étant donné 4 formes quadratiques f_1, f_2, g_1 et g_2, de trouver une matrice inversible S telle que $g_1 = f_1 \circ S$ et $g_2 = f_2 \circ S$, lorsqu'elle existe. Aucun algorithme plus rapide que la recherche exhaustive n'était connu à ce jour pour résoudre ce problème malgré plusieurs articles lui étant consacré. Le schéma d'identification lui-même s'inspire très directement du schéma plus ancien de Goldreich, Micali et Wigderson reposant sur la difficulté de l'isomorphisme de graphe.

Durant ma thèse, j'ai conçu un algorithme de résolution du problème algorithmique sous-jacent, qui permet de casser en quelques secondes les paramètres proposés par Patarin, et dont le niveau de sécurité attendu était de l'ordre de 2^{256}. Cet algorithme fonctionne en faisant apparaître S comme la solution d'un problème d'équivalence de pinceaux matriciels construit à partir de la représentation des formes quadratiques par des matrices symétriques. Il apparaît alors que S vit dans un sous-espace vectoriel de dimension $\mathcal{O}(n)$ de l'ensemble des matrices, et qu'elle peut être entièrement déterminée par la résolution d'une (petite) instance de MQ.

2.3.2 Etude du problème de l'équivalence linéaire de polynômes

Une partie de la cryptographie multivariée repose dans le fond sur une procédure *d'obfuscation* assez simple. On choisit une fonction $f : (\mathbb{F}_q)^n \to (\mathbb{F}_q)^n$ qu'on sait facilement inverser, et qui admet une représentation par des polynômes quadratiques. Ensuite, on la "cache" en définissant $\mathbf{PK} = T \circ f \circ S$, où S et T sont des matrices inversibles secrètes. La clef publique $\mathbf{PK}$ (un vecteur de polynômes quadratiques multivariés) est alors une version "masquée" de f, qu'on espère impossible à inverser. Le problème de *l'équivalence linéaire de polynôme* (PLE pour faire court) consiste à retrouver S et T étant donné $\mathbf{PK}$ et f. La difficulté déduire la clef secrète de la clef publique dans un certain nombre de schémas multivariés repose donc sur le caractère difficile de ce problème. Ce problème se présente en deux variantes assez différentes, selon que f et $\mathbf{PK}$ sont formés de polynômes homogènes (cas dur) ou inhomogènes (cas facile).

2.3.2.1 Nouveaux algorithmes pour le problème IQMLE

Faugère et Perret avaient déjà observé en 2006 que le problème de *l'équivalence linéaire de polynômes quadratiques inhomogènes* (IQMLE) était (empiriquement) soluble en temps $\mathcal{O}\left(n^9\right)$. Néanmoins, leur assertion ne reposait pas sur une preuve mathématique rigoureuse, mais sur l'observation du comportement d'un algorithme de calcul de base de Gröbner.

J'ai mis au point deux algorithmes de résolution de IQMLE. Le premier, assez heuristique et inspiré de la méthode "To-n-Fro" de Patarin, Goubin et Courtois, fonctionne en temps $\mathcal{O}\left(n^3\right)$, et échoue parfois de manière surprenante. Le second fonctionne en temps $\mathcal{O}\left(n^6\right)$ en moyenne, et s'applique à la (large) fraction des instances qui satisfont une certaine condition. Ce deuxième algorithme fonctionne également en faisant apparaître S et T comme solutions d'un problème d'équivalence de pinceaux matriciels, ce qui permet là encore de démontrer qu'elles vivent dans un espace vectoriel de dimension $\mathcal{O}(n)$. Récupérer S et T revient alors à résoudre un système de $\approx n^3$ équations quadratiques en n variables, ce qui est possible en temps polynomial si les équations sont linéairement indépendantes.

Le caractère inhomogène de l'instance est crucial pour construire les pinceaux équivalents, et donc faire marcher l'ensemble de la technique.

2.3.2.2 Nouveaux algorithmes pour le problème QMLE

Je me suis enfin intéressé au cas difficile où les polynômes sont homogènes (le problème QMLE), et où les techniques précédentes ne s'appliquent pas. Dans cette configuration, la méthode "To-n-Fro" de Patarin, Goubin et Courtois a une complexité au moins aussi grande que q^{2n}. La complexité de l'algorithme de Faugère et Perret, qui repose sur un calcul de base de Gröbner, est globalement inconnue, mais pourrait être de l'ordre de $\mathcal{O}\left(2^{18n}\right)$ si un certain système d'équation quadratique a un comportement "régulier".

J'ai conçu deux algorithmes, l'un s'exécutant en $\mathcal{O}\left(q^{2/3n}\right)$ opérations, et l'autre, valable uniquement quand $q = 2$, s'exécutant en $2^{n/2}$ opérations. Les deux algorithmes fonctionnent en tentant de déterminer l'image de S en un point, ce qui permet de se ramener au cas inhomogène facile. Pour accomplir cet objectif plus vite que la recherche exhaustive, les deux algorithmes reposent sur le paradoxe des anniversaires et s'inspirent de la méthode de Weisfeiler-Lehman pour l'isomorphisme de graphe. Une pair de graphes est construite, et S est un isomorphisme transformant l'un en l'autre. Pour trouver l'image de S en un point, on échantillonne des noeuds dans le graphe et on explore la topologie de leur voisinage, qui est préservée par l'isomorphisme. Une collision entre la "forme" de deux voisinages indique alors la possibilité d'une relation de la forme $y = S \cdot x$. Le premier algorithme ne regarde que les voisins immédiats de chaque noeud, tandis que le second algorithme explore des voisinages plus gros.

2.3.3 Attaque d'une classe de clefs faibles dans HFE [BFJT09]

HFE est un schéma de chiffrement/signature proposé par Patarin en 1996. Il repose aussi sur l'idée de l'obfuscation d'une fonction aisément inversible, mais cette fois-ci la fonction obfusquée est gardée secrète. La sécurité de HFE ne repose donc pas sur la difficulté de problème d'équivalence linéaires de polynôme. Patarin avait proposé, pour diminuer la taille des clefs, l'utilisation d'une variante "sous-corps". Dans ce cas-précis, le Frobenius commute avec la fonction aisément inversible f, et il existe en fait une paire de matrices (U, V) telle que $\mathbf{PK} = U \circ \mathbf{PK} \circ V$. Récupérer U et V revient à résoudre une instance de QMLE, et ensuite, reconstruire la clef secrète à partir de U et V est faisable en pratique (et nous conjecturons que cela peut se faire en temps polynomial). Cette attaque permet, sur la variante sous-corps de HFE, de récupérer en pratique la clef secrète de HFE à partir de la clef publique pour les paramètres proposés par Patarin.

2.4 Mes publications

La liste de mes publications scientifiques figure dans les tables 2.1 et 2.2. Je suis également l'auteur de deux textes de vulgarisation. J'ai écrit une postface au roman *Le code de Cambridge* de Tony Gheeraert, paru en mars 2010 aux éditions Le Pommier, dans laquelle les idées et les techniques liées à la cryptographie apparaissant dans le roman sont détaillées. Je suis également l'auteur d'un petit livre de vulgarisation de la cryptographie intitulé *Sommes-nous prisonniers des codes secrets ?*, également paru aux éditions du Pommier en avril 2011, dans laquelle je discute les tenants et les aboutissants de la cryptographie dans la vie quotidienne de monsieur tout-le-monde.

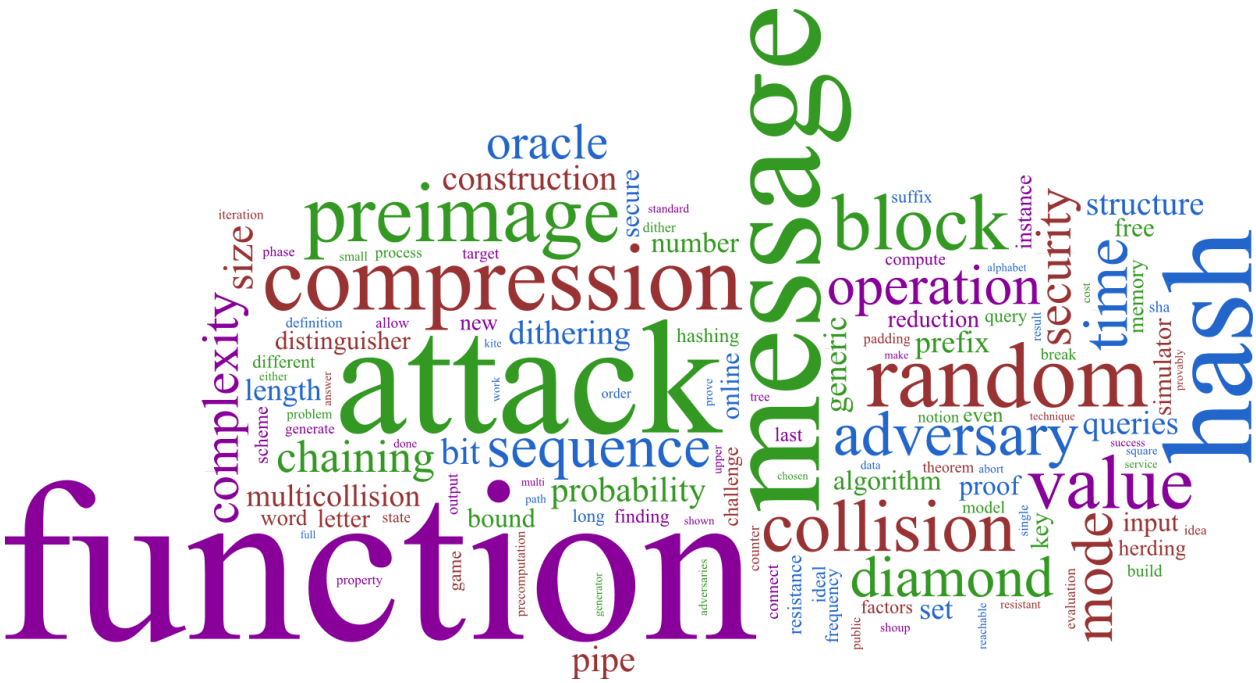
[ABF+08]	Second Preimage Attacks on Dithered Hash Functions Elena Andreeva, C. B., Pierre-Alain Fouque., Jonathan J. Hoch, John Kelsey, Adi Shamir et Sébastien Zimmer (Eurocrypt 2008)
[BF08]	Analysis of the Collision Resistance of RadioGatún Using Algebraic Techniques C.B. et Pierre-Alain Fouque (SAC 2008)
[ABDK09]	Herding, Second Preimage and Trojan Message Attacks beyond Merkle-Damgård Elena Andreeva, C. B., Orr Dunkelman et John Kelsey (SAC 2009)
[BDFL10]	Another Look at the Complementation Property C.B., Orr Dunkelman, Pierre-Alain Fouque et Gaëtan Leurent (FSE 2010)
[BCC+10]	Fast Exhaustive Search for Polynomial Systems in $\mathbb{F}_2$ C.B., Hsieh-Chung Chen, Chen-Mou Cheng, Tung Chou, Ruben Niederhagen, Adi Shamir et Bo-Yin Yang (CHES 2010)
[BFL10]	Security Analysis of SIMD C. B, Pierre-Alain Fouque et Gaëtan Leurent (SAC 2010)
[BDLF10]	Attacks on Hash Functions Based on Generalized Feistel : Application to Reduced-Round Lesamnta and SHAvite-3_{512} C. B., Orr Dunkelman, Gaëtan Leurent et Pierre-Alain Fouque (SAC 2010)
[BFFP11]	Practical Cryptanalysis of the Identification Scheme Based on the Isomorphism of Polynomial with One Secret Problem C. B., Jean-Charles Faugère, P.-A. Fouque et Ludovic Perret (PKC 2011)
[BDLF11]	New Insights on Impossible Differential Cryptanalysis C. B., Orr Dunkelman, Gaëtan Leurent et Pierre-Alain Fouque (SAC 2011)
[BDF11]	Automatic Search of Attacks on Round-reduced AES and Applications C. B., Patrick Derbez et Pierre-Alain Fouque (Crypto 2011)
[BFMR11]	Practical Key-recovery For All Possible Parameters of SFLASH C. B., Pierre-Alain Fouque et Gilles Macario-Rat (Asiacrypt 2011)

Table 2.1 – Mes publications scientifiques acceptées dans des conférences internationales avec comité de lecture.

[BFJT09]	A Family of Weak Keys in HFE (and the Corresponding Practical Key-Recovery) C. B., Pierre-Alain Fouque, Antoine Joux et Joana Treger (accepté au Journal of Mathematical Cryptology)
[BDD+10]	Low Data Complexity Attacks on AES C. B., Patrick Derbez, Orr Dunkelman, Nathan Keller et Pierre-Alain Fouque (soumis aux IEEE Transactions on Information Theory)
[ABD+10]	New Second Preimage Attacks on Hash Functions Elena Andreeva, C. B., Orr Dunkelman, P.-A. Fouque, Jonathan Hoch, John Kelsey, Adi Shamir et Sébastien Zimmer (soumis au Journal of Cryptology)
[BFZ10]	On the Resistance of Practical Hash Functions Constructions to Generic Second Preimage Attacks C. B., Pierre-Alain Fouque et Sébastien Zimmer (soumis aux Information Processing Letters)

Table 2.2 – Mes publications scientifiques soumises à journaux internationaux avec comité de lecture.

Modes of Operations of Hash Functions



Introduction

A *hash function* is "any well-defined procedure or mathematical function that converts a large, possibly variable-sized amount of data into a small datum, usually a single integer that may serve as an index to an array" [Wik11]. When used to accelerate data lookup in databases or associative arrays, the most interesting properties of hash functions are probably the speed at which they can be computed, and the uniformity of the output distribution on more-or-less arbitrary inputs.

Cryptographic Hash Function Security

To be used in a cryptographic environnment, a hash function must of course be efficiently computable and uniformly distributed, but this is hardly sufficient; a cryptographic hash function must also offer more features. An easily-invertible, very malleable [1] function is of little use in cryptography, but, on the other hand, "secret-protecting" functions can serve a handful of purposes. The usual intuition is that a cryptographic hash function is such that $H(M)$ reveals *nothing useful* about M. We briefly discuss several possible applications of a cryptographic hash functions H, highlighting the various features required therefrom.

Password Storage. User accounts are very often protected by passwords in multi-user computing environments or online services. Directly storing these passwords is a bad practice: as a result of a misconfiguration, for instance, users could possibly read the file containing all passwords. A mischievous user could try to boot his colleague's computer using an alternate media (such as bootable USB stick), then observe the content of the hard-drive, and steal his colleague's password. Lastly, online servers could be hacked, again resulting in password theft.

Hash functions can be used to securely store passwords in a variety of contexts. Instead of directly storing a password p, we store the hash of p. As long as H is a *one-way* hash function (also called *preimage-resistant* hash functions), then reading the password file neither reveals the actual passwords nor "equivalent" passwords that have the same hash. It is still possible to verify that users enter their passwords correctly, by evaluating H on a user's input and comparing the result with $H(p)$.

Faster and Smaller Digital Signatures. In most electronic signature applications, it is beneficial to sign the hash of a document rather than signing the document itself. Signature schemes tend to be slow (the RSA signature algorithm is typically cubic in the size of the modulus, which has to be longer than the message to sign), and it is desirable to keep the signatures as small as possible. Thus, instead of signing a document M, we instead sign $H(M)$. This practice can apparently be traced back down to [Wil80].

Doing so introduces the possibility that adversaries cheat the system without breaking the signature algorithm, but instead by attacking the hash function. If the hash function is not *collision-resistant*, *i.e.*, if it is possible to find two distinct messages M_1 and M_2 such that $H(M_1) = H(M_2)$, then a frauder could *repudiate* his own signatures: he could compute a colliding pair M_1, M_2 such that $H(M_1) = H(M_2)$, sign M_1 (*e.g.* his electronic tax forms), and then later on pretend that he signed M_2, since they have the same hash, and therefore the same signature.

If the hash function is not *second-preimage-resistant*, then a third party could forge signatures and try to *impersonate* a legitimate signer. Suppose Mélanie signs a message M, which is subsequently made public. The signature algorithm has been applied to $H(M)$, and if Victor (the Vilain) is able, given M, to find another message M' such that $H(M) = H(M')$, then he could exhibit a legitimate signature of M' by Mélanie, even though Mélanie herself never signed it.

Note that in this scenario two different flaws in the hash function lead to two different breakages of the signature scheme. If the hash function is not collision-resistant but only second-preimage resistant, then the signature scheme is secure against malicious outsiders, but not against malicious users.

1. The usual meaning of this notion is that an input can be chosen in order to enforce that the output has a given property.

Timestamping. Suppose we would like to set up a secure timestamping web service. The users would like to have a guarantee that they were in possession of a given file at a certain point in time. For instance, an inventor could want to prove that she owned a description of her invention before making it public. On the other hand, the users do not want us (the web service), or the general public to learn the content of their files. Cryptographic hash functions solve this problem nicely: users send us the hashes of their files, and we publish the hashes along with the times at which we received them. If the hash function is one-way, then no information about the users's files is leaked (the web service is "hiding").

In order to be reliable, for instance to stand up in a court of law, our timestamps should resist forgeries, in other words it must be *binding*: it must not be possible to send us a hash, and then, at a later point in time, to forge a file, with a partially chosen content, yielding this particular digest (the users really commit to their files). Here is a typical example in which such forgeries could be exploited: a con man could pretend to be a medium and to be able to predict the future. To prove his point, he would tell everyone that he had written down his prediction in a file and he would commit it to the timestamping web service. In fact, he would perform some kind of precomputation and commit a carefully chosen hash h_{fraud} to the web service. Then, after the actual event takes place, he would write down a description of what happened (let us call this "message" M_{event}) and he would try to find a suffix $M_{forgery}$ such that $H(M_{event}\|M_{forgery}) = h_{fraud}$. If he succeeded, the con man could pretend that his prediction has been fulfilled, and reveal $M_{event}\|M_{forgery}$ to prove his point. Of course his prediction file would contain a little bit of rubbish at the end, but who cares?

Observe that the relevant security notion of the hash function is a bit more sophisticated than what we have seen before. If H is collision-resistant, then of course the timestamping is secure, but this condition is not necessary. One-wayness is not sufficient, because the game played by the adversary is *a priori* easier than just inverting the hash function. The right notion is *Chosen Target Forced Prefix Collision* resistance [KK06].

How to Make it Simple

Hash functions are the swiss army knife of contemporary cryptography, and if we examined more and more common usages of hash functions (entropy extraction, key derivation, forward security, MACs, etc.) we would likely come up with a few more security notions that a hash function should satisfy. This is going to result in a (long) list of more-or-less natural security notions that a "good" hash function should satisfy to be used in all the known cryptographic contexts.

On the one hand, it is good to know precisely what cryptographers expect from hash functions, as this sets the objective hash function designers should try to achieve. On the other hand this leads to a horrible and unsatisfactory definitional mess. What if tomorrow the killer application of hash functions requires a completely new and unrelated security property? A possible way to simplify the theory of hash function would be to come back to the initial intuition and to observe that, in essence, a *public function without any exploitable structure* could be used every time a cryptographic hash function is required. In addition, this definition of cryptographic hash functions could very likely be considered as reasonable by most cryptographers. Unfortunately, this intuitive idea does not lend itself well to any useful kind of mathematical formulation, which prevents its effective use in security proofs. The problem originates from the fact that it is difficult, if not impossible, to formalize the notion of "not having any exploitable structure" or, equivalently, of "not having any bad property" in a satisfactory way. This would further require a precise definition of what a "bad" property is.

A satisfying way to create a function that does not have any "bad" property and that does not have any "exploitable structure" is to simply pick it at random. Indeed, one can easily check that a random function satisfies all the security notions outlined above. Furthermore, random functions are "usual" mathematical objects having well-understood properties.

The Random Oracle Model

The observation that random functions would make great hash functions has led to the developpement of the *Random Oracle methodology*, a computational model in which a "truly" random function (called the Random Oracle) from $\{0,1\}^*$ to $\{0,1\}^n$, for some n large enough (say $n = 256$), can be queried by everyone via some kind of public interface.

The Random Oracle Model (ROM) had been implicitly used in the 1980's, but was made explicit, formalized, and advocated by Bellare and Rogaway in [BR93]. It was quickly found to be a very powerful method to formally prove many protocols secure. For instance, standardized public-key schemes, such as RSA-PSS [BR96] and RSA-OAEP [BR94] are only proved secure in the ROM. In fact, there are so many papers proving a construction secure in the ROM that it is impossible to list them all: [Cor02a, Cor02b, KW03, BGH07, Ber08, GPV08], etc.

The Random Oracle Model is practical, but has been the object of much (and heated) debate. Assuming the existence of a public random function leads to a nice theory, but raises a burning problem in practice: how to implement the Random Oracle in real life? It is indeed not very realistic to assume the existence of a public random function from $\{0,1\}^*$ to $\{0,1\}^n$. For instance, the Random Oracle is an infinite source of statistically independent random bits (*i.e.*, it contains an infinite amount of statistical entropy). This means that a random function from $\{0,1\}^*$ to $\{0,1\}^n$ does not admit any finite description, while a hash function is just a fixed piece of code. This fact has been used by Canetti, Goldreich and Halevi to build a signature scheme that is provably secure in the Random Oracle Model, but that cannot be securely instantiated using any hash function [CGH04]. Yet, in "real life", the Random Oracle often turns out to be implemented with a cryptographic hash function.

Of course, there would be some other ways to address this issue. A world-wide organization could build a somewhat-secure hardware random number generator and use it to run a Random Oracle Web Service:

1: **function** RANDOM-ORACLE-WEB-SERVICE(*Query*)
2: **if** $Log[Query] = \perp$ **then** $Log[Query] \xleftarrow{\$} \{0,1\}^n$ (using physical device)
3: **return** $Log[Query]$
4: **end function**

Countless technical problems would occur though, a major one being that the amount of storage required to operate the random oracle web service (*i.e.*, the space required to store the query log) would be ever growing. A funny denial-of-service attack would then consist in repeatedly querying the random oracle on pseudo-random junk. In addition, this would force every cryptographic primitive to be connected to the internet to operate. Any similar way to implement the Random Oracle without using a fixed hash function that everyone can evaluate independently would very likely turn out to be as impractical as this Web Service.

Theory and Practice

So, when a cryptographic scheme proved secure in the Random Oracle Model is about to be implemented and deployed, then the public random function that does not actually exist has to be replaced by *something*, usually by a hash function. This creates a detrimental gap between theory and practice, because the Random Oracle is better than hash function can possibly be. Leurent and Nguyen for instance investigated the quality of instantiations of the Random Oracle in various situations and the consequences of hash function failure on ROM-secure schemes [LN09]; they found that some *de facto* classical Random Oracle instantiations were fairly weak (much weaker than the hard problem the public-key scheme where they are used relies on). In the worst case, the implementations of the Random Oracle by hash functions could lead to provably secure protocols becoming practically insecure when implemented. The example of Canetti *et al.* is fairly unnatural, but is sufficiently real to make cryptographers suspicious. In addition, a much less contrived example, due to Ristenpart, Shacham and Shrimpton recently attracted some attention [RSS11].

Consider a secure storage web service. Users upload their precious files, and the web service stores them reliably, typically using redundant storage. The users can be assured that their data will not be lost in the event their computer fails or is stolen. However, the web service must not fail either. To gain the trust of the users, the web service has set up an *audit* procedure that supposedly convince the users that their files are still properly stored. To verify whether the web service still has an unaltered copy of a given file, a user challenges the web service with an arbitrary string S, and the web service returns $H(F\|S)$, where F denotes the file's content. In the Random Oracle Model, the web service cannot cheat, and cannot possibly answer the challenge correctly without fully knowing F. However, as soon as H is instantiated by any secure but *iterated* hash function (this includes the MD and SHA family, as well as the future SHA-3), then the web service may cheat: it may store the internal state of the hash function after processing F, and use it later on to compute $H(F\|S)$, even if the actual content of F has been erased. This is a simple and natural example of a cryptographic construction that is secure in the ROM, but that blatantly fails when using any reasonable hash function.

The assimilation of hash functions with public random functions certainly simplifies protocol design, but the extension of this paradigm has consequences on hash function design. Hash function designers are told: "*create something that resembles a random function*". For all reasons advocated above, this objective is vague and it is not really a meaningful job assignment. In addition, this is very likely more difficult than designing (say) one-way functions or randomness extractors. Besides, the resulting multi-purpose hash functions are likely to be slower than their more specialized counterpart. An example: to extract a 128-bit session key from a Diffie-Hellman element $g^{xy} \bmod n$, a possible solution is to hash the bitstring representation of the group element. However, Chevalier, Fouque, Pointcheval and Zimmer have shown in [CFPZ09] that it is sufficient to *truncate* (instead of hashing) the bitstring representation of the group element to obtain sufficiently many random bits!

If, instead of relying on the existence of a public random function, protocol designers could instead rely

on the existence of a single function (or eventually of a family of functions) enjoying a well-defined security property such as collision-resistance, one-wayness, ... then their protocols would be secure in the *standard model*, and the problem would go away. There is a trend that tries to avoid the Random Oracle in security proofs [Sho00b, BB04, BF05, Can97, MRV99], but this is complicated, often results in less efficient schemes... and is sometimes provably impossible [Nie02, PV05, DOP05, KP09]. Let us nevertheless keep in mind that a possible way to avoid problems related to the Random Oracle is to try not to use it, but instead to use a function offering a more precise security property.

Let us look at the same problem from a different angle. The Random Oracle Model, by advocating "one hash function for everything" puts more stress on the single hash function that gets to be used for everything. This makes life simpler for protocol designers, as they do not have to determine precisely what security property is required from the hash function in their particular scheme. However, the problem is precisely that it is sometimes unclear what exact properties are required from the hash function. Sometimes, the security proof crucially rely on the fact that the Random Oracle can generate random bits, something that cannot be done by a hash function. A hash function supposed to simultaneously satisfy any possible security notion, including those that are impossible to satisfy (*e.g.* randomness) and those its designers did not think about, is more likely to have problems, than a merely collision-resistant function for instance. The situation may become more complicated once problems happen, because the Random Oracle-substitute provably does not satisfy some security notion, and the security of every scheme using it "as a random oracle" must be re-assessed with respect to the new vulnerability. For instance, all public-key schemes would not break as badly, and not in the same way, if collisions could be found for the standard hash function [LN09]. Ciphertext distinguishability would not be as bad as key-recovery, for instance.

On the Difficulty of Public Life

Like all other cryptographic constructions, the description of hash functions are public. Most other cryptographic constructions are expected to show some cryptographic strength against adversaries from which some information is kept secret (*e.g.*, secret key in a block cipher, PRNG seed, etc.). In strong contrast with nearly all the other cryptographic primitives, hash functions are expected to show cryptographic strength *without secret*. This is an endless source of difficulties, both in practice and in theory. In practice, because usually the existence of secret informations (the key of a MAC, the internal state of symmetric cipher, etc.) seriously complicates the attackers's life. On the contrary, no such obstacle lies in front of adversaries trying to find a collision on a hash function: they have full knowledge of *everything*, all input, output and internal state bits of the hash function. Adversaries against hash function are more powerful than most others, they have more control. This resulted in many standard hash functions getting broken, sometimes in practice (this is the case of MD4, MD5, SHA-0). Attacks have been found against MD4 [Dob98, WLF+05, Leu08a], MD5 [dBB93, Dob96, WY05, Kli06, SLdW07, Leu08b, SA09, SSA+09], SHA-0 [CJ98, WYY05b, BCJ+05, dCR08, JP07, MP08] and SHA-1 [WYY05a, dCR06, dCMR07], to mention only the widely used hash functions existing prior to the SHA-3 competition. Practical breaks of widely-deployed block ciphers are extremely rare by comparison.

Let us try to briefly explore the theoretical difficulties now. A well-established methodology to formally define the security of a cryptographic primitive is to define an *ideal* version thereof, and then to compare actual versions of the primitive with the ideal version. The maximum advantage of any distinguisher (with bounded ressources) between the ideal and the actual versions is then a relevant security indicator. This methodology have been used with great success for block ciphers, stream ciphers, pseudo-random functions, pseudo-random number generators, etc.

For hash functions, we have seen that the ideal primitive would be a public random function, and the actual primitives would be small and public pieces of code. This discrepancy makes it impossible to define the security of hash functions via a distinguishing game, where a distinguisher would have to tell apart a random function H^* and some actual hash function H. Because H^* can only be accessed via an external interface, then our distinguishing game would see a distinguisher connected to a black box containing either H^* or H, and trying to tell which one it is. However, because the description of H is public, the distinguisher can evaluate H by itself, and it is therefore able to predict the output of H on any input. The predictions will only be satisfied by the random function with a negligible chance, and the distinguisher will succeed with overwhelming probability after a single query.

So, while random functions are nice "ideal version" of hash functions, we do not know how to measure the security of an actual hash function, namely to measure how close is our hash function to a random function [2]. It is not clear at all that such a measurement makes sense, or even exist. To conclude this

2. We note that although it makes sense to play a distinguishing game between a block cipher where the key is randomly chosen and a random permutation, it does not make sense to play a distinguishing game between a particular block cipher and the ideal cipher...

(a) Michelangelo di Lodovico Buonarroti Simoni, The Delphic Sibyl (1510)

(b) John Collier, Priestess of Delphi (1891)

(c) J.W. Waterhouse, Consulting the Oracle, (1884)

introduction on the Random Oracle, we point out that several Oracle instantiations have been proposed in the past, see Figure 2.5a and 2.5b. Querying these oracles can be done as shown in Figure 2.5c.

Iterated Hashing

Compressing an arbitrary quantity of data into a small fingerprint can be achieved in a natural way by an *iterated* process, as first suggested in 1978 by Rabin [Rab78]: split the arbitrarily large input into a certain number of small chunks of a fixed size and process these chunks one at a time. These small chunks are usually called the *message blocks*, and are processed using an "inner" hash function that compresses a *finite* amount of data into a smaller amount.

Iterated hash functions maintain an *internal state*, which is initialized to a default value called the *Initialization Vector* (IV), and updated each time a message block is processed. The typical hash function

is therefore defined mathematically by a process of the form:

$$\begin{aligned} x_{-1} &= IV \\ x_i &= f(x_{i-1}, m_i), \end{aligned}$$

where the m_i's are the message blocks. After all the message blocks have been processed, a *finalization function* (possibly the identity) extracts the hash value from the internal state. The function f is the *compression function*. If the internal state is n-bit wide and the message blocks are m-bit wide, then f compresses $n+m$ bits into just n. Of course, if the length (in bits) of the "message" to be hashed is not a multiple of m, then a *padding scheme* will add bits to the last block so that it reaches a length of m bits.

This construction is practical: its time complexity is linear in the message size (which is optimal if the hash actually depends on every input bits), it requires a modest amount of memory, and it does not requires the whole message to be stored in a fast memory so that it can be applied "on-the-fly" to data streams. In addition, designing a compression function seems much simpler than designing a full-blown hash function from scratch. Most existing hash functions are built on the same approach: a compression function is used in some way to process the message, one chunk at a time. Thus, a hash function is formed of two essential components, the *compression function*, and the *mode of operation* that describes how the compression should be used. We usually denote the compression function by f, and the mode of operation by H. The full hash function obtained by combining the compression function with the mode of operation is denoted by H^f, or simply H when unambiguous.

Another trend in hash function design consist in iterating a *weak* function or permutation over a large internal state which is not completely under the control of the adversary. The security of the whole iteration then does *not* rely on any cryptographic strength of the underlying iterated construction. The *sponge* paradigm [BDPA08] is an implementation of this idea.

Modes of Operations

This way of building hash function raises interesting questions: what properties should the compression function satisfy? Can a secure hash function be obtained from a weak compression function? Conversely, is it possible to turn a good compression function into a good hash function? Are the security properties of H^f related to those of f? These question are central in hash function theory. This research area has been very active in the first decade of the twenty-first century, mainly because the usual, standard mode of operation (implemented in MD4, MD5, SHA-0, SHA-1, in all the SHA-2's, etc.) turned out to have unexpected and surprising problems. The preparation of the SHA-3 competition further stimulated research in modes of operation, because all the hash function designers of the world had to give some thoughts to this issue at some point. We even witnessed some candidates getting kicked out of the competition because of problems in their mode of operation.

Attacks against the modes of operation are special in many aspects. They apply to H^f for any choice of f, making them particularly irritating for any hash function designer investing a lot of time in the design of a secure compression function. They often require only simple tools (basic algorithms and discrete probabilities), but can become fairly intricate.

On the other hand, modes of operations admit a nice theory, and some form of "provable security". For instance, it *does* make sense to define the security of a mode of operation H by setting up a distinguishing game where a distinguisher would have to tell apart the Random Oracle and H^f, where f is a random compression function. A mode of operation not capable of resisting distinguishers up to a high number of queries should very likely be thrown away. As a matter of fact, the mode of operation of MD4, MD5, SHA-0, SHA-1 fails this test after two queries!

It thus appear that the random oracle model is a friendly framework to discuss the security of modes of operations. Assuming that the compression function is a public random function, it is sometimes possible to prove lower-bounds on the number of queries issued by the distinguisher to achieve a reasonable advantage. A proof that a mode of operation offer some security property "in the Random Oracle Model" (*i.e.*, assuming that the compression function is random) does not offer any guarantee that extends to the standard model, but it gives a good indication that the mode of operation is not inherently flawed: no attack can break it when the compression function is unbreakable.

Organization of the Part. Chapter 3 describes the mainstream mode of operation of hash function, the Merkle-Damgård construction, as well as the numerous generic attacks that have been discovered prior to this work. Variants of the Merkle-Damgård construction that have been designed with the explicit objective of preventing some of these attacks are also described, and known security results are recalled when they exist. Chapter 4 present a new generic second-preimage attack that applies to the Merkle-Damgård construction, as well as to some of its variants. It allows to break a proposal by Ron Rivest that explicitly aimed at preventing

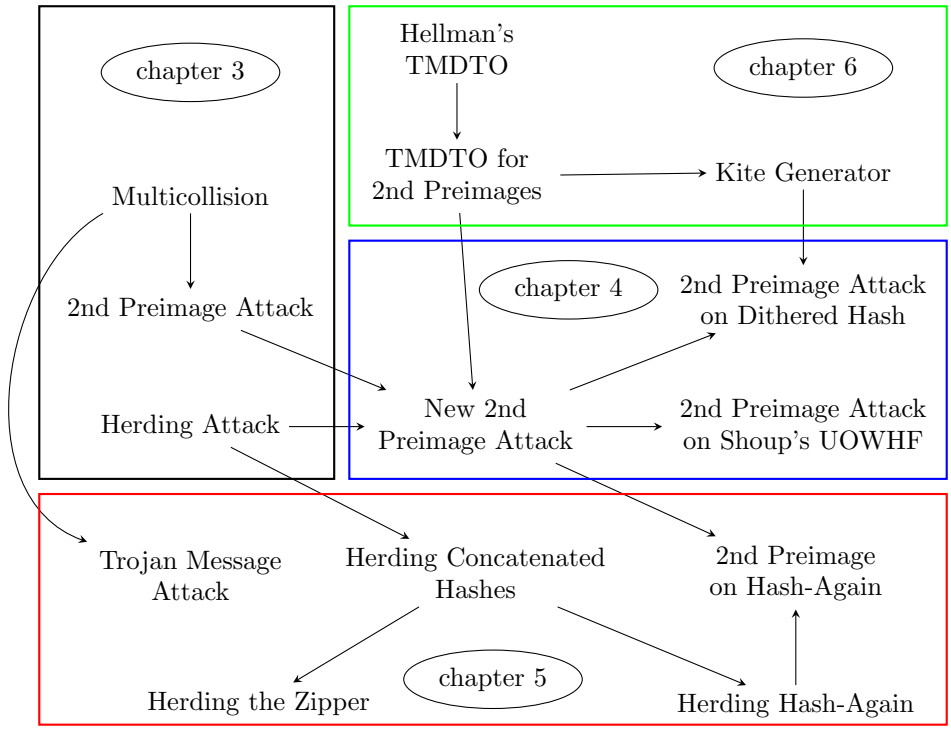


Figure 2.5: Overview of part I

generic second preimage attacks. Chapter 5 present new generic attacks on *concatenated hashes* and some non-streaming modes of operations. It culminates with a generic second preimage attack on the folklore mode of operation that consists in hashing two concatenated copies of the message, and a new generic attack on the Merkle-Damgård mode of operation, the "Trojan message attack". We have publicly demonstrated the latter with the MD5 hash function. This was made possible because collisions have become so easy to find. Chapter 6 discusses how Time-Memory-Data Trade-offs can be applied to generic second preimage attacks. It allows for instance to attack shorter messages than the normal versions of the corresponding attacks, at the expense of some large precomputation. In chapter 7 we present several *security proofs* for modes of operation. We prove for instance that there are easy ways to repair the Merkle-Damgård construction to make it provably secure against generic second preimage attacks. We also prove that it is possible to iterate "weak" compression functions in a provably secure way, for a certain notion of weakness that includes the existence of symmetries and differential paths.

Chapter 3

Modes of Operations and Generic Attacks

In this chapter we recall the computational models and the security notions used throughout this part. We also describe several modes of operations and discuss various attacks against them.

3.1 Generalities

A *hash function* $H : \{0,1\}^* \to \{0,1\}^n$ is a public function that should not have any exploitable structure. At the very least, a hash function must be:

- *Preimage-Resistant*: given $y \in \{0,1\}^n$, no adversary must be able to find $x \in \{0,1\}^*$ such that $H(x) = y$ faster than exhaustive search.
- *Second Preimage-Resistant*: given $x \in \{0,1\}^*$, no adversary must be able to find $x' \in \{0,1\}^*$ such that $x \neq x'$ and $H(x) = H(x')$ faster than exhaustive search.
- *Collision-resistant*: no adversary must be able to find $x, x' \in \{0,1\}^*$ such that $x \neq x'$ and $H(x) = H(x')$ faster than exhaustive search

To make these notions meaningful, we should first determine what the expected security level is. If H were a Random Oracle, then it would be easy to show that any adversary trying to find a (second-)preimage would success with probability $q/2^n$ after issuing q queries to the random oracle. Each query would result in a uniformly random answer in $\{0,1\}^n$, so that each query yields the (second-)preimage with probability 2^{-n}. The success probability scales linearly with the number of queries.

The situation is different for collision resistance because of the *birthday paradox*. Intuitively, after q queries to H, the adversary knows $q(q-1)/2$ input-output pairs, and each pair result in a collision with probability 2^{-n}. Thus, we expect to find a collision after only $2^{n/2}$ queries. This argument is informal and not rigorous, but it can be made formal. This often involves throwing balls at random into bins.

Theorem 3.1 ([Sho09])**.** *Suppose n balls are thrown independently and uniformly at random into m bins, and let $\mathcal{C}$ denote the event that at least one bin contains more than one ball. Then:*

$$1 - e^{-\frac{n(n-1)}{2m}} \leq \mathbb{P}[\mathcal{C}] \leq \frac{n(n-1)}{2m}$$

From this theorem, we deduce that if H is a random oracle, and if we keep hashing arbitrary but distinct inputs, we will get a collision with probability close to $1/2$ after $2^{n/2}$ queries. After $4 \cdot 2^{n/2}$ queries, the probability that a collision has been found is greater than 99.9%. It is often also interesting to estimate the probability that two random subsets of $\{0,1\}^n$ have a common element. A more colorful version of the previous theorem does exactly that.

Theorem 3.2 ([Vau05a])**.** *$\theta_1\sqrt{N}$ red balls and $\theta_2\sqrt{N}$ blue balls are thrown independently in a uniformly random way into N bins. Let $\mathcal{E}$ be the event that no bin simultaneously contain red and blue balls. Then*

$$\mathbb{P}[\mathcal{E}] \xrightarrow[N \to +\infty]{} e^{-\theta_1\theta_2}$$

In the sequel, we will often make use of these two theorems without mentioning them, and sometimes neglecting (small) constants factors, as is commonly done in the field.

The above security definitions are informal, and any attempt to make the definition of Collision-Resistance formal runs into difficulties. It is for instance useless to measure the smallest running time of any adversary returning a collision, because there always exist trivial adversaries that contain a precomputed hard-coded collision, print it and terminate in constant time. This difficulty can be removed by considering hash function *families*, or equivalently, *keyed* hash functions. The collision adversaries could then be challenged with the

key, and cannot contain precomputed collisions for every possible keys [1]. The drawback is that in practice keyed hash function are only seldom used.

We will now give formal definitions of these informal security notions for keyed hash functions. Let $\mathcal{M}$ denotes the space of all possible messages and $\mathcal{K}$ denotes the set of all possible keys. If A is an adversary and if $\mathbf{Adv}(A)$ is a measure of its *advantage*, then we write $\mathbf{Adv}(t)$ to mean the maximal value of $\mathbf{Adv}(A)$ over all possible adversaries A running in time t. We usually measure adversarial advantage by the success probability of the adversaries. We are now ready to speak of the difficulty with which an adversary finds collisions or (second-)preimages. The following definitions have been given by Rogaway and Shrimpton in [RS04], and are now standard. Let $\mathcal{K}$ denote the space of possible keys (typically $\{0,1\}^k$) and $\mathcal{M}$ denotes the space of possible messages (typically $\{0,1\}^m$). We write $x \xleftarrow{\$} X$ to denote the action of drawing x uniformly at random from the set X.

Definition 3.1. Let $H : \mathcal{K} \times \mathcal{M} \to \{0,1\}^n$ be a hash function family, let $\ell \in \mathbb{N}$ be such that $\{0,1\}^\ell \subseteq \mathcal{M}$, and let A be a (collision) adversary. The *Advantage of A against Coll* is

$$\mathbf{Adv}_H^{\mathrm{Coll}}(A) = \mathbb{P}\left[K \xleftarrow{\$} \mathcal{K}; (M, M') \xleftarrow{\$} A(K) : (M \neq M') \wedge (H_K(M) = H_K(M'))\right]$$

The *advantage of A against Sec/eSec/aSec* is

$$\begin{aligned}
\mathbf{Adv}_H^{\mathrm{Sec}[\ell]}(A) &= \mathbb{P}\left[K \xleftarrow{\$} \mathcal{K}; M \xleftarrow{\$} \{0,1\}^\ell; M' \xleftarrow{\$} A(K, M) : (M \neq M') \wedge (H_K(M) = H_K(M'))\right] \\
\mathbf{Adv}_H^{\mathrm{eSec}[\ell]}(A) &= \max_{M \in \{0,1\}^\ell} \mathbb{P}\left[K \xleftarrow{\$} \mathcal{K}; M' \xleftarrow{\$} A(K) : (M \neq M') \wedge (H_K(M) = H_K(M'))\right] \\
\mathbf{Adv}_H^{\mathrm{aSec}[\ell]}(A) &= \max_{k \in \mathcal{K}} \mathbb{P}\left[M \xleftarrow{\$} \{0,1\}^\ell; M' \xleftarrow{\$} A(M) : (M \neq M') \wedge (H_K(M) = H_K(M'))\right]
\end{aligned}$$

While there is a single kind of Collision adversaries, there are several notions of second preimage adversaries: Sec adversaries are imposed both the key and the challenge message, eSec adversaries may choose the challenge message and are imposed the key (the definition of the advantage assumes that they choose the easiest message for them), and finally aSec adversaries may choose the key and are imposed the challenge message. eSec stands for "everywhere second preimage resistance", and aSec stands for "always second preimage resistance". By definition, an eSec-secure hash function family is called a *Universal One-Way Hash Function family* (UOWHF), as defined by Naor and Yung [NY89].

The second preimage security notions depend on the length of the message because, as we shall see later, there are adversaries that perform better when ℓ is large. It is also possible to define a *keyless* notion of (second-)preimage resistance.

Definition 3.2. Let $H : \mathcal{M} \to \{0,1\}^n$ be a single hash function, let $\ell \in \mathbb{N}$ be such that $\{0,1\}^\ell \subseteq \mathcal{M}$, and let A be a (second-)preimage adversary. Then, the *advantage of A against Spr* is

$$\mathbf{Adv}_H^{\mathrm{Spr}[\ell]}(A) = \mathbb{P}\left[M \xleftarrow{\$} \{0,1\}^\ell; M' \xleftarrow{\$} A(M) : (M \neq M') \wedge (H(M) = H(M'))\right]$$

The *advantage of A against Pre* is

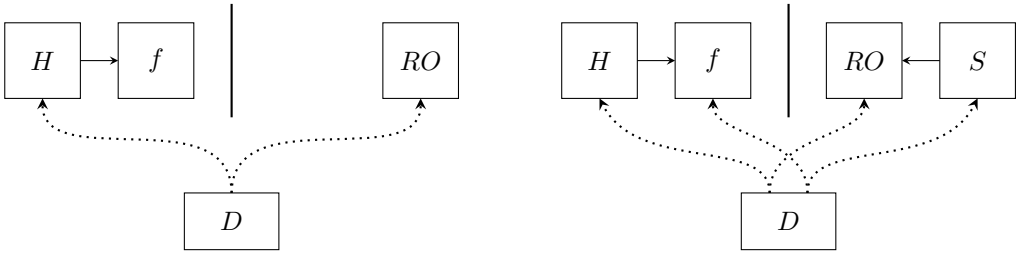
$$\mathbf{Adv}_H^{\mathrm{Pre}}(A) = \mathbb{P}\left[y \xleftarrow{\$} \{0,1\}^n; M \xleftarrow{\$} A(y) : H(M) = y\right]$$

An interesting attempt to formalize collision-resistance in the standard model that does not involve keyed hash functions has been given by Rogaway in [Rog06]. His idea is to consider a secure cryptographic protocol using a hash function H, and to build an explicitly *reduction*, *i.e.*, an efficient program that may interract with an adversary against the protocol, and that outputs a collision on H with essentially the same advantage as the attacker against the protocol. Thus, as long as mankind does not "know" any collision on H, then the protocol cannot be broken by hash functions collisions.

3.1.1 Indistinguishability and Indifferentiability

A reasonable way to reason about the security of modes of operations is to instantiate the inner primitive (typically the compression function) by an ideal version thereof (an ideal block cipher, a random function or permutation), and to see how the combination "mode of operation + ideal inner primitive" differs from the Random Oracle.

1. it is standard to assume that the time complexity of a program includes the time of reading a full description thereof, for instance to load it into memory prior to execution. Thus, an adversary with an exponentially large description has an exponential complexity.



(a) The Indistinguishability setting (b) The Indifferentiability setting

Figure 3.1: The distinguishing games involved in the study of modes of operations

Definition 3.3. A Turing Machine H (implementing a mode of operation) with oracle access to an ideal primitive f is said to be (t, q, ε)-*indistinguishable* from a Random Oracle if for any distinguisher D it holds that:

$$\left|\mathbb{P}\left[D^{H}=1\right]-\mathbb{P}\left[D^{RO}=1\right]\right|<\varepsilon$$

The distinguisher runs in time at most t and makes at most q queries.

Figure 3.1a illustrates this scenario. Informally, a mode of operation H is indistinguishable from a Random Oracle if no efficient algorithm D connected to either H or RO, is able to decide whether it is interacting with H or RO. It makes sense to require indistinguishability, since a hash function is not supposed to exhibit any "structure" that would make it different from a random function.

A good mode of operation should therefore not admit any distinguisher making only a polynomial number of queries, for instance. While this security notion is simple and natural, it assumes that the distinguisher D has black-box access to the hash function H, which implies in particular that D cannot access the compression function f nor query it on chosen inputs. This assumption does not make much sense, because in practice the description of the compression function is public. This flaw in the indistinguishablility framework has led to the definition of a more refined setting, the *indifferentiability framework*, by Maurer, Renner and Holenstein [MRH04].

Definition 3.4. A Turing Machine H (implementing a mode of operation) with oracle access to an ideal primitive f is said to be $(t_S, t_D, q, \varepsilon)$-*indifferentiable* from a Random Oracle if there exist a simulator S, such that for any distinguisher D it holds that:

$$\left|\mathbb{P}\left[D^{H,f}=1\right]-\mathbb{P}\left[D^{RO,S}=1\right]\right|<\varepsilon$$

The simulator has access to the Random Oracle and runs in time at most t_S. The distinguisher runs in time at most t and makes at most q queries.

Figure 3.1b illustrates this new framework. The idea is to let the distinguisher query not only the hash function H, but also the compression function f. However, we must then put *something* in place of f in the "Random Oracle" world, hence the need of a simulator that plays the role of f for the distinguisher. Of course, since the answers of f and H are correlated, then the answers of the simulator must be correlated to those of the Random Oracle, and it is necessary to let the simulator query the Random Oracle.

We note that if a hash function H is indifferentiable up to q queries, then no generic attacks can break H in less than q queries, otherwise the attack could be used as a distinguisher. Indifferentiability proofs are therefore proofs *that there are no generic attacks*. We conclude by observing that an indistinguishable mode of operation is by definition indifferentiable.

3.2 The Merkle-Damgård construction and its Security

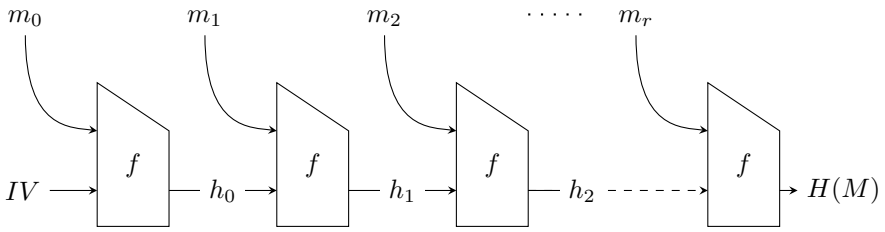
The Merkle-Damgård mode of operation used to be (and is still, at the time of this writing) the most mainstream and most widely implemented mode of operation of hash functions. The Merkle-Damgård mode of operation constructs a full hash function $H^f : \{0,1\}^* \to \{0,1\}^n$ by iterating a compression function $f : \{0,1\}^n \times \{0,1\}^m \to \{0,1\}^n$. The hash process is described and illustrated in Algorithm 3.1.

The common padding rule, referred to as the Merkle-Damgård *strengthening*, appends to the original message a single '1' bit followed by as many '0' bits as needed to complete an m-bit block after embedding the message length at the end.

It is worth observing that a hash function family can easily be derived from the Merkle-Damgård construction, by letting the IV be the key. In the sequel, unless mentioned otherwise, H denotes the Merkle-Damgård

Algorithm 3.1 The Merkle-Damgård mode of operation.

1: **function** Merkle-Damgård (M)
2: **Pad** and **split** the message M into r blocks $m_0, \ldots, m_r$ of m bits each.
3: **let** h_{-1} be the initialization value IV.
4: **for each** message block m_i compute $h_i = f(h_{i-1}, m_i)$.
5: **return** $H^f(M) = h_r$.
6: **end function**



mode of operation instantiated with any compression function, and Pad denote the padding scheme. When $M = m_0, \ldots, m_r$ is a message whose length is a multiple of the message block length, we denote by $f^*(h, M)$ the value $f(f(\ldots f(f(h, m_0), m_1) \ldots, m_{r-1}), m_r)$, *i.e.*, the result of setting the internal state to h and hashing M without padding. It follows that $H(M) = f^*(IV, \text{Pad}(M))$.

3.2.1 Effect of the "Strengthening"

Merkle [Mer89] and Damgård [Dam89] proved independently in 1989 that this innocent-looking padding rule has a very interesting consequence in terms of security, as it makes the scheme *collision resistance preserving*, in the sense that a collision on the hash function H^f implies a collision on the compression function f.

Theorem 3.3 ([Mer89, Dam89])**.** *The Merkle-Damgård mode of operation is Collision-Resistance Preserving, i.e., there is an algorithm that given two messages $M \neq M'$ such that $H(M) = H(M')$ finds two pairs $(h, m) \neq (h', m')$ such that $f(h, m) = f(h', m')$ and whose running time is linear in the size of M.*

Proof. Suppose we have two messages $M \neq M'$ such that $H^f(M) = H^f(M')$. We will show that it is easy to derive a collision on f from M and M'.

- Either $|M| \neq |M'|$. In this case, by definition of the padding scheme, the inputs of the last invocation of the compression are not the same when hashing M and M', and because M and M' collide, we have found a collision on f (on its last invocation).
- Or $|M| = |M'|$. Suppose that the compression function is invoked r times in both cases. In this case, there are again two possibilities. Either $(h_{r-1}, m_r) \neq (h'_{r-1}, m'_r)$, and we have a collision since $h_r = h'_r$, or $(h_{r-1}, m_r) = (h'_{r-1}, m'_r)$. The argument repeats. Either we find a collision along the way, or we reach the conclusion that $m_i = m'_i$, for all i, which is impossible.

□

Adding the message length in the last block, the so-called "strengthening", is the crucial ingredient in establishing the collision-resistance preservation of Merkle-Damgård. As an illustration of the well-known fact that the devil is in the details, we show that many other natural padding schemes would lead to a handful of security problems. Consider the padding scheme Pad_0 that simply pads the last message block with zeroes: it would fall to a very simple generic collision attack consisting in adding a single "0" bit at the end of the message.

This could be avoided with a more sophisticated padding scheme Pad_{10} that adds a single "1" bit and then pads the current block with zero bits. Sadly, this padding scheme would fall to a simple second-preimage attack called the *long message attack* [MvOV]. Given a (long) message $M = M_1, \ldots, M_k$, an adversary could hash random messages M' until the hash of M' is equal to one of the intermediate chaining values (say h_i) obtained in hashing M. She could replace the beginning of M, up to the i-th block, with M', and she would end up with as second preimage. The expected number of queries is $2^n/k$.

As a side effect, the strengthening defines a limit on the maximal length for admissible messages. In most deployed hash functions, this limit is 2^{64} bits, or equivalently 2^{55} 512-bit blocks. However, in SHA-384 and SHA-512, as well as in some SHA-3 candidates, this limit has been increased to 2^{128} bits, or equivalently 2^{118} 1024-bit blocks. In the sequel, we denote the maximal number of admissible blocks by 2^κ.

3.2.2 Blockcipher-based Compression Functions

With a reasonable mode of operation such as Merkle-Damgård, the problem of designing a (good) hash function is reduced to that of designing a (good) compression function. One of the most succesful approaches is to turn a *block cipher* into a compression function, with an appropriate mode of operation for compression functions. There are countless ways to perform this task, so we only mention the most popular, the *Davies-Meyer* constructions, that sets

$$f(h, m) = E_m(h) \oplus h$$

This construction is used in many standardized hash functions (MD4, MD5, all the SHA's, etc.). It is simple and efficient, since the complexity of evaluating the compression function is vastly the complexity of the block cipher itself. In addition, amongst all symmetric cryptographic primitives, block ciphers are probably the best understood, with the caveat that in the Davies-Meyer construction the adversary has full control over the key, which means to some extent that the block cipher E has to be secure against related-keys attacks. Winternitz first showed that the Davies-Meyer compression functions are optimally collision and preimage resistant when the block cipher is an *ideal cipher*, *i.e.*, a key-indexed collection of independent random permutations [Win84].

A noticeable property of the Davies-Meyer construction is related to *fixed points*. Given a message block m, it is straightforward to find a chaining value h^* such that $h^* = f(h^*, m)$: the value of h^* is by definition $E_m^{-1}(0)$. It follows that for any m, the fixed point h^* can be found for the cost of one compression function evaluation. This is mostly not a problem, because the attackers usually do not control the chaining value, but we will see in §3.3.3 that this can be leveraged into a second preimage attack.

Many other block-cipher based modes of operations have been described by Preneel, Govaerts and Vandewalle [PGV93], and have been analyzed in the *Ideal Cipher Model* by Black, Rogaway and Shrimpton [BRSS10]. Black, Cochran and Shrimpton proved that it is impossible to build a secure compression function with a single evaluation of a n-bit fixed permutation (*i.e.*, a block cipher under a fixed key) [BCS05] [2]. Shrimpton and Stam proved that it *is* possible with three calls to three different n-bit fixed permutations [SS08].

3.2.3 Collisions on the Compression function

Finding collisions against H can be acheived by finding collisions on the compression function, *i.e.*, by finding pairs $(h, m) \neq (h', m')$ such that $f(h, m) = f(h', m')$. This can always be done by brute force, with about $2^{n/2}$ evaluations of the compression function, even when the attacker only has black-box access to f. There are several kinds of collision-finding algorithms: the chaining values h and h' can be chosen by the attacker, or may be imposed upon her. In addition, it may be required that $h' = h$. This lead to four possible kinds of adversaries, shown in Algorithm 3.2. All these algorithms terminate after about $2^{n/2}$ iterations if f is a random function. In practice, when dealing with concrete hash functions, the complexity of finding the various kinds of collisions are not completely identical. For instance, on MD5, there exist a version of Collision that terminates in time equivalent to 2^{16} compression function evaluations, but the fastest equivalent of Dual-IV-Collision requires 2^{41} compression function evaluations [SSA+09]. To conclude on this subject, we note that the collision-finding techniques decsribed in Algorithm 3.2 have a high memory complexity, but it is possible to search for collisions *without any memory*, using cycle-finding techniques such as the algorithms of Floyd, Brent or Nivasch [Niv04].

3.3 Generic Attacks Against Merkle-Damgård

Because of its simplicity and of its nice security properties, the Merkle-Damgård mode of operation has been widely deployed. However, more than 15 years after its introduction,it started to exhibit weird properties, and to fall to nastier and nastier generic attacks.

3.3.1 The Length Extension Attack

The *length extension* attack is simple and dangerous, because it can be carried out without extensive computational resources. It allows to distinguish the Merkle-Damgård construction from a Random Oracle using only two queries.

Theorem 3.4 (length-extension attack)**.** *There is an adversary that, given only the size of a message M and its hash $H(M)$, but not M itself, is capable of computing $H(\text{Pad}(M)\|S)$ in time $\mathcal{O}(|S|)$, for any suffix S. If H were a random oracle, achieving the same thing would require about 2^n queries.*

2. More precisely, they proved that there exist *information-theoretic* adversaries issuing a small number of queries to the fixed permutation

Algorithm 3.2 Generic Brute-Force collision-finding algorithms

function Collision(IV)
  $T \leftarrow \emptyset$
  loop
    $m \xleftarrow{\$} \{0,1\}^m$
    $h \leftarrow f(IV, m)$
    if $T[h] = m'$ **then return** (m, m')
    $T[h] \leftarrow m$
  end loop
end function

function Pseudo-Collision( )
  $T \leftarrow \emptyset$
  loop
    $m \xleftarrow{\$} \{0,1\}^m$
    $x \xleftarrow{\$} \{0,1\}^n$
    $h \leftarrow f(x, m)$
    if $T[h] = (x', m')$ **then return** (x, m, x', m')
    $T[h] \leftarrow (x, m)$
  end loop
end function

function Dual-IV-Collision(IV_1, IV_2)
  $T_1 \leftarrow \emptyset$
  $T_2 \leftarrow \emptyset$
  loop
    $m \xleftarrow{\$} \{0,1\}^m$
    $h_1 \leftarrow f(IV_1, m)$
    $h_2 \leftarrow f(IV_2, m)$
    if $T_2[h_1] = m'$ **then return** (m, m')
    if $T_1[h_2] = m'$ **then return** (m, m')
    $T_1[h_1] \leftarrow m$
    $T_2[h_2] \leftarrow m$
  end loop
end function

function Dual-IV-Pseudo-Collision( )
  $T_1 \leftarrow \emptyset$
  $T_2 \leftarrow \emptyset$
  loop
    $m \xleftarrow{\$} \{0,1\}^m$
    $x_1 \xleftarrow{\$} \{0,1\}^n$
    $x_2 \xleftarrow{\$} \{0,1\}^n$
    $h_1 \leftarrow f(x_1, m)$
    $h_2 \leftarrow f(x_2, m)$
    if $T_2[h_1] = (x', m')$
    or $T_1[h_2] = (x', m')$
      then return (x, m, x', m')
    $T_1[h_1] \leftarrow (x_1, m)$
    $T_2[h_2] \leftarrow (x_1, m)$
  end loop
end function

Here is how the attack works. We first observe that $\text{Pad}(M)$ ends at a block boundary (by definition of the padding scheme). Let k therefore denote the length of $\text{Pad}(M)$ in blocks. We observe what happens during the process of hashing $M' = \text{Pad}(M)\|S$. First, the padding scheme is applied to M', resulting in a sequence of message blocks $m_0, \ldots, m_t$. Provided that S is more than m bits long, then generating m_t can be accomplished knowing only $|M|$ and S. Then by definition of the Merkle-Damgård construction, $H(M') = f^*(IV, m_0, \ldots, m_t)$. It then follows from the previous consideration that $m_1, \ldots, m_k = \text{Pad}(M)$, so that $H(M') = f(H(M), m_{k+1}, \ldots, m_t)$.

Prefix-MAC. One of the simplest possible ways to build a MAC from a hash function is probably the Prefix-MAC construction, defined by: $\text{prefix-MAC}(K, M) = H(K \,\|\, M)$. The length-extension attack makes it completely insecure when H is any Merkle-Damgård hash function. Indeed, after querying the MAC on M, an adversary would be able to forge the MAC on $\text{Pad}(M) \,\|\, S$, for any suffix S. This problem led to the definition of more complicated hash-based MAC constructions such as HMAC [KBC97]. Candidates for the SHA-3 competition were required to be immune to the length-extension attack.

Countermeasures. The attack can be avoided without modifying the Merkle-Damgård construction too much in several ways. A first possibility is to apply a *prefix-free* encoding to the messages before hashing them, *i.e.*, a function $pf(\cdot)$ such that $pf(M)$ cannot be a strict prefix of $pf(M')$, for any choice of $M \neq M'$ (a possibility is to include the message length at the begining, or to use a special "last block" bit in every message block). Another, but essentially equivalent option is to apply a one-way *finalization function* to the internal state before returning it, *i.e.*, to define $H(M) = F(h_r)$, where F should preferably be a permutation, otherwise there could be a loss of entropy.

When the length-extension attack is thwarted, for instance by the use of a prefix-free encoding, then the resulting Prefix-free-Merkle-Damgård mode of operation is indifferentiable from a random oracle up to $2^{n/2}$ queries, as shown by Coron, Doodis, Malinaud and Puniya [CDMP05].

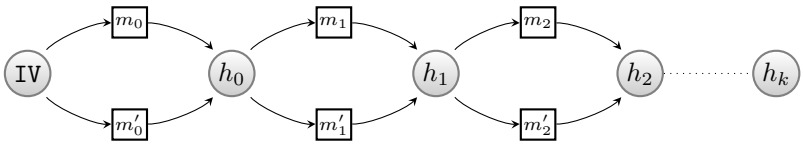


Figure 3.2: Joux's multicollision attack.

3.3.2 Joux's Multicollision Attack

> I wish I had the idea...
>
> An anonymous reviewer

The *multicollision* attack on Merkle-Damgård discovered by Joux in 2004 is fairly general and applies more generally to (most) iterated hash functions [Jou04]. The point is to find a *k-way multicollision*, *i.e.*, a set of k messages $M_1, \ldots, M_k$ all hashing to the same value, much faster than exhaustive search.

Theorem 3.5 ([Jou04])**.** *There exist an adversary that finds a k-way multicollision on H, for any k, with a time and memory complexity equivalent to that of running the generic* Collision *algorithm (or any ad hoc equivalent) $\lceil \log_2 k \rceil$ times. If H were a random oracle, this should require about $2^{n(k-1)/k}$ evaluations of H.*

For instance, finding a 4-way multicollision requires $2^{3n/4}$ queries on the random oracle, but can be achieved with only $2 \cdot 2^{n/2}$ evaluations of the compression function for any Merkle-Damgård hash function. The main idea of multicollision is shown in Figure 3.2. First compute two colliding message blocks:

$$(m_0, m'_0) \leftarrow \textsc{Collision}(IV)$$

and let $h_0 = f(IV, m_0)$ denote their common hash. Now, starting from h_0, we again compute two colliding message blocks:

$$(m_1, m'_1) \leftarrow \textsc{Collision}(h_0)$$

At this stage, we already have a 4-way collision: the four messages $(m_0, m_1), (m'_0, m_1), (m_0, m'_1)$ and (m'_0, m'_1) result in the same intermediate chaining value $h_1 = f(h_0, m_1)$. By finding a third pair (m_2, m'_2) of message blocks colliding from h_1, we could build an 8-way multicollision, by appending either m_2 or m'_2 at the end of our 4-way multicollision, and so on and so forth.

The multicollision attack is very difficult to avoid, because it is very general: it works for any padding scheme, and only exploit the fact that the hash function is iterated. For instance, it applies to the Prefix-free-Merkle-Damgård mode of operation, allowing it to distinguish it from a random oracle by constructing a 4-way multicollision, using about $4 \cdot 2^{n/2}$ queries. This shows for instance that Prefix-Free-Merkle-Damgård cannot be indifferentiable from a Random Oracle beyond $2^{n/2}$ queries, so that the proof of Coron *et al.* is tight [CDMP05].

3.3.3 Dean's Second Preimage Attack

As discussed in §3.2.1, the Merkle-Damgård strengthening prevents the long message second preimage attack: because the original message and the forgery do not have the same length, the padding scheme makes the last blocks different in $\textsc{Pad}(M)$ and $\textsc{Pad}(M')$, and even though the penultimate chaining values are the same, the last ones are extremely likely to be different because the last message blocks are different.

A possible way to mount a successful second preimage attack on Merkle-Damgård is to bypass the strengthening by forging preimages of the same size as the original message. In his PhD thesis [Dea99], Dean found a way to do this under the assumption that it is possible to find *fixed points* of the compression function.

Theorem 3.6 ([Dea99])**.** *If f is a Davies-Meyer compression function, then there is an adversary A that finds second preimages of ℓ-blocks messages for H^f with $2^n/\ell$ compression function evaluations, after a preprocessing phase of complexity $2^{n/2}$. If H were a Random Oracle, this would require 2^n compression function evaluations.*

This result is, to the best of our knowledge, only present in Dean's dissertation and has not been submitted to any refereed publication. It therefore received very little publicity and the cryptographic community was apparently unaware of it until several years later. This is a bit surprising, since this result is in fact quite strong: the most widely used construction of a hash function out of a block cipher has a non-trivial security flaw that nobody was aware of before.

Offline Phase. The adversary first tries to find two message blocks $m^{\nearrow}$ and m^* such that $f(IV, m^{\nearrow}) = h^*$ and $f(h^*, m^*) = h^*$. If the compression function were an arbitrary random function, then there would not be any way of finding these faster than exhaustive search. However, when f is a Davies-Meyer compression function, then $m^{\nearrow}$ and m^* can be found after $2^{n/2}$ compression function evaluation, even when the underlying block cipher is ideal:

1. Construct a table T_1 of $2^{n/2}$ entries (h_1, m_1) where $m_1 \xleftarrow{\$} \{0,1\}^m$ and $h_1 = f(IV, m_1)$.
2. Construct a table T_2 of $2^{n/2}$ entries (h_2, m_2) where $m_2 \xleftarrow{\$} \{0,1\}^m$ and $h_2 = E^{-1}_{m_2}(0)$.
3. Find $(m^{\nearrow}, m^*)$ such that there exist $h^* \in \{0,1\}^n$ with $(h^*, m^{\nearrow}) \in T_1$ and $(h^*, m^*) \in T_2$.

Theorem 3.2 tells us that the probability to find the actual collision in the two lists is $1 - 1/e$, and that this probability can be made arbitrarily large by slightly increasing the size of the two tables.

Online Phase. Presented with a (padded) challenge message $M = m_0, \ldots, m_\ell$, the adversary hashes random messages blocks $m^{\searrow}$ until $f(h^*, m^{\searrow}) = h_i$, where $h_i = f^*(IV, m_0, \ldots, m_i)$ is the i-th intermediate chaining value obtained while hashing M. We expect the adversary to find this "connecting" message block after $2^n/\ell$ evaluations of the compression function. The adversary can then forge a second preimage M' that has the same size as M by replacing the first i blocks of M. This yields:

$$M' = m^{\nearrow}, m^*, m^*, \ldots, m^*, m^{\searrow}, m_{i+1}, \ldots, m_\ell$$

It is then easy to check that $H(M') = H(M)$. They key idea of the attack is to replace a *prefix* of M by something of the same length that hashes to the same value.

3.3.4 Kelsey and Schneier's Second Preimage Attack

Dean's attack was in fact rediscovered independently by Kelsey and Schneier in 2005, and at this point reached the general crypto community. Building on Joux's multicollision attack, Kelsey and Schneier found a more general second preimage attack that applies to any compression function [KS05].

Theorem 3.7 ([KS05])**.** *There is an adversary A that finds second preimages of ℓ-blocks messages for H^f with $2^n/\ell$ compression function evaluations, after a preprocessing phase of complexity $\kappa \cdot 2^{n/2}$. If H were a Random Oracle, this would require 2^n compression function evaluations.*

Dean's trick to bypass the Merkle-Damgård strengthening was, after the right fixed point has been found, to be able to build a message of *any length* hasing to the internal chaining value h^*. Kelsey and Schneier observed that this is in fact a huge multicollision containing (at least) a message for any possible length in some interval. They called such a multicollision an *expandable message*. To build expandable messages, Kelsey and Schneier adapted Joux's multicollision technique: instead of finding collisions between single messages blocks, they find collisions between a single message block and a longer message.

Offline Phase. From a practical point of view, an expandable message is a datastructure $\mathcal{M}$ generated by a preprocessing step, associated with a hash value h^*. Then an instantiation procedure takes $\mathcal{M}$ and an integer ℓ, and forges an ℓ-block message M such that $f^*(IV, M) = h^*$. Pseudo-code for both steps are shown in Algorithm 3.3.

The complexity of Generate-Expandable-Message is equivalent to that of finding κ collisions generically. An algorithm of the type Dual-IV-Collision could be used to find $\mathcal{M}_i$ and $\mathcal{M}'_i$ at each step: choose arbitrarily the first 2^i blocks of $\mathcal{M}_i$, and simultaneously find $\mathcal{M}'_i$ and the last block of $\mathcal{M}_i$ by running Dual-IV-Collision$(h, f^*(h, \mathcal{M}_i))$. As a consequence, expandable messages could be built *in practice* for MD5.

Online Phase. A second preimage attack can be carried out without too much trouble once an expandable message has been constructed. The pseudo-code is shown in Algorithm 3.4. The same argument used to establish the correctness of Dean's attack also applies here: the attack uses the expandable message to replace a prefix of M by something of the same length and that hashes to the same thing. We expect $2^n/\ell$ iterations of the repeat...until loop.

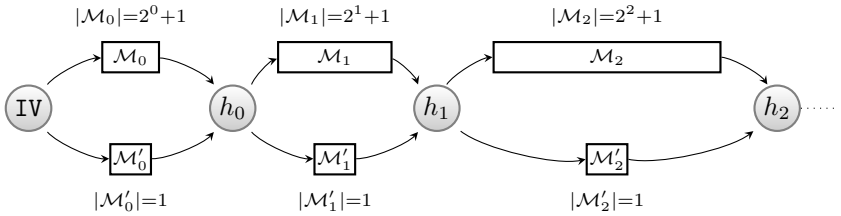
3.3.5 The Herding attack

The "herding attack" (also known as the "Nostradamus attack", or the "con man" attack, as mentionned in the introduction) was described by Kelsey and Kohno in 2006 [KK06]. It is a *chosen-target forced prefix preimage* attack: an adversary commits to a public digest value h. After the commitment phase, the

Algorithm 3.3 Expandable messages: a multicollision between messages of different lengths.

1: **function** GENERATE-EXPANDABLE-MESSAGE( )
2: $h \leftarrow IV$
3: **for** $i = 0$ to κ **do**
4: Find $(\mathcal{M}_i, \mathcal{M}'_i)$ such that $f^*(h, \mathcal{M}_i) = f(h, \mathcal{M}'_i)$, $|\mathcal{M}_i| = 2^i + 1$ and $|\mathcal{M}'_i| = 1$
5: **end for**
6: **return** $\mathcal{M} = (\mathcal{M}_0, \mathcal{M}'_0), \ldots, (\mathcal{M}_\kappa, \mathcal{M}'_\kappa)$
7: **end function**

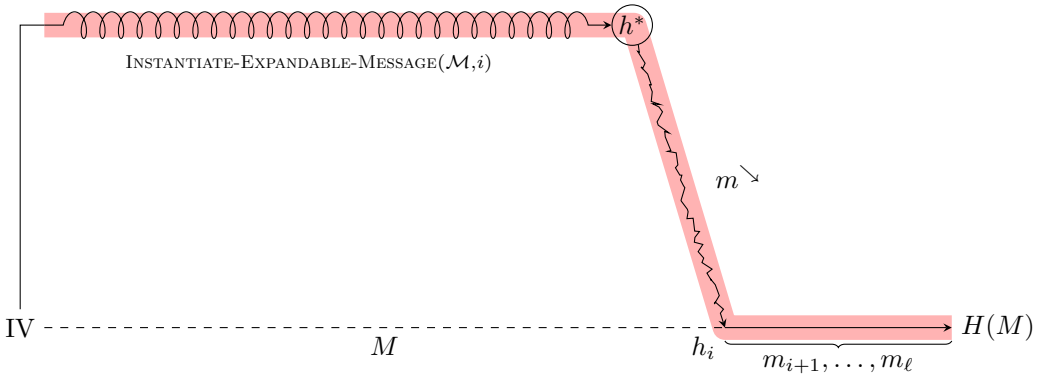
The notation $|m|$ denotes the size of the m in message blocks.

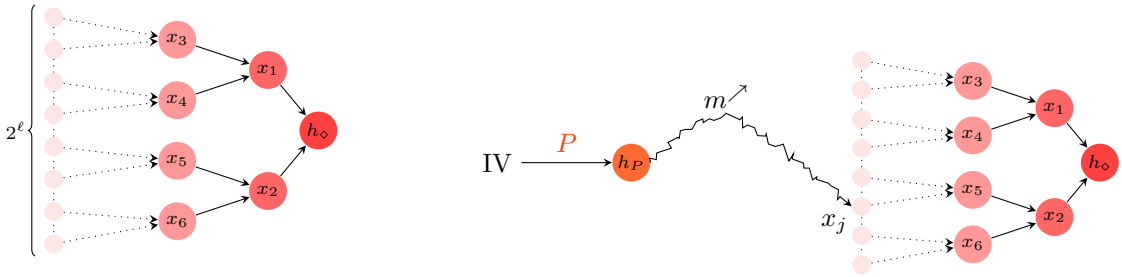


1: **function** INSTANTIATE-EXPANDABLE-MESSAGE($\mathcal{M}$, ℓ)
2: $b_\kappa, \ldots, b_0 \leftarrow$ binary writing of $(\ell - \kappa)$
3: $M \leftarrow \emptyset$
4: **for** $i = 0$ to κ **do**
5: **if** $b_i = 1$ **then** $M \leftarrow M \| \mathcal{M}_i$ **else** $M \leftarrow M \| \mathcal{M}'_i$
6: **end for**
7: **return** M
8: **end function**

Algorithm 3.4 The Kelsey-Schneier Second Preimage Attack.

function SECOND-PREIMAGE(M)
$m_0, \ldots, m_\ell \leftarrow$ PAD(M)
$h_{-1} \leftarrow IV$
for i $= 0$ **to** ℓ **do** $h_i \leftarrow f(h_{i-1}, m_i)$
$\mathcal{M} \leftarrow$ GENERATE-EXPANDABLE-MESSAGE()
$h^* \leftarrow f^*(IV, \mathcal{M}'_0, \ldots, \mathcal{M}'_\kappa)$
repeat
$m^{\searrow} \xleftarrow{\$} \{0,1\}^m$
until $f(h^*, m^{\searrow}) = h_i$ for some $i \geq \kappa$
return INSTANTIATE-EXPANDABLE-MESSAGE($\mathcal{M}, i$)$\|m_{i+1}, \ldots, m_\ell$
end function





(a) The diamond structure

(b) Online phase of the attack

Figure 3.3: The Herding Attack

adversary is challenged with a prefix P which she has no control over, and she is to produce a suffix S for which $h = H(P\|S)$. Of course, h is specifically chosen after a precomputation phase by the adversary. This attack is reminiscent of (and can certainly be seen as) an attack against the eSec security notion.

Theorem 3.8 ([KK06]). *There exist an adversary that chooses a hash value h after a precomputation phase of complexity $2^{n/2+\ell/2+2}$, and which is then capable of finding messages S such that $h = H(P\|S)$, for any prefix P, in $2^{n-\ell}$ compression function evaluations. If H were a Random Oracle, this would require 2^n queries.*

Offline Phase. The main idea behind this attack is to build a special data structure known as a 2^ℓ-*diamond structure*[3], and shown in Figure 3.3a. It contains 2^ℓ distinct chaining values $D = x_1, \ldots, x_{2^\ell}$ from which the adversary knows how to reach a common chaining value $h_\diamond$. To construct this data structure, which is in fact a tree, the adversary picks about $2^{n/2-\ell/2+1/2}$ single-block messages m_j, and evaluates $f(x_i, m_j)$ for all i and j. Due to the large number of values, it is expected that collisions occur, and it is expected that the adversary will find for each of the x_i's a corresponding message block $m_{\alpha(i)}$ such that the set $\{f(x_i, m_{\alpha(i)})\}$ contains only $2^{\ell-1}$ distinct values. The process is then repeated $\ell - 1$ more times until a final digest value $h_\diamond$ is found. The complexity of the first step should dominate the whole computation, so that the complexity of building the diamond structure is roughly $2^{n/2+\ell/2+2}$ compression functions evaluations.

We are aware that this construction algorithm is a bit vague, and that this complexity analysis is not rigorous. We essentially reproduced the description and analysis given in [KK06]. An improved and more rigorous analysis [BSU11] revealed that the announced complexity is underestimated by a factor $\sqrt{\ell}$. Because it is more convenient, and because the difference is not so large, we stick with the old complexity estimates.

Online Phase. In the online phase of the attack, the adversary is challenged with a prefix P, and she has to find a suffix S so that $H(P\|S) = h_\diamond$. To this end, the adversary hashes random message blocks $m^\nearrow$ until $f^*(P\|m^\nearrow) = x_i \in D$ for some i. Once P is "connected", via $m^\nearrow$, to the diamond structure, it is possible to follow the path connecting x_i to $h_\diamond$ (which is at the "root" of the diamond) and produce the required suffix S. Figure 3.3b illustrates the process.

The total time complexity of the attack is about $2^{n/2+\ell/2+2}$ offline compression function evaluations, and $2^{n-\ell}$ online compression function evaluations. Choosing $\ell = n/3$ minimizes the total cost of the attack, resulting in a total complexity of about $2^{2n/3}$ compression function evaluations.

3.4 Close Relatives of the Merkle-Damgård Construction

we have seen in §3.3 that in 2204, 2005 and 2006 many unexpected and disturbing generic attacks were discovered on the venerable Merkle-Damgård mode of operation. This stimulated the cryptographic community to research alternative new modes of operations, and led to a flourishing of new designs, some of which were explicitly aiming at preventing the generic attacks. In this section we survey the design that are sufficiently close to the original Merkle-Damgård.

3. We find this name somewhat unfortunate, but since it has been adopted we keep using it. We assume it has something to do with "Lucy in the Sky with Diamonds", as nothing else explains this name.

3.4.1 HAIFA : a HAsh Iterative FrAmework

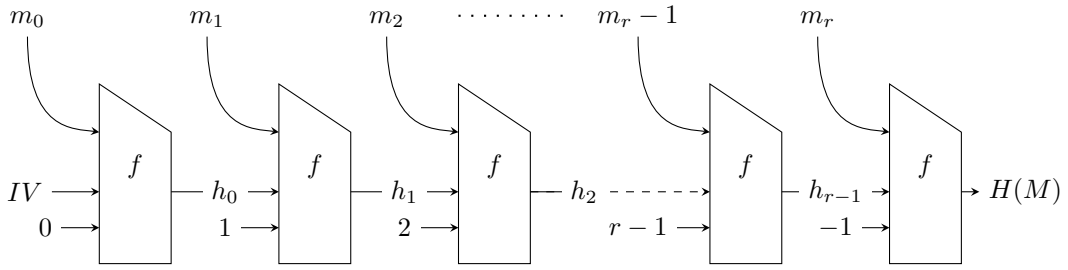
HAIFA is a collection of simple tweaks to Merkle-Damgård proposed by Biham and Dunkelman in 2006 [Eli06, BD07]. We will not describe the full HAIFA, because some details are irrelevant to us (for instance, HAIFA has a built-in salt mechanism that we deliberately ignore). A HAIFA hash function $H : \{0,1\}^* \to \{0,1\}^n$ is built by iterating a compression function

$$f : \{0,1\}^m \times \{0,1\}^n \times \{0,1\}^\kappa \to \{0,1\}^n .$$

The precise mode of operation is described in Algorithm 3.5 (and PAD denotes the usual Merkle-Damgård padding). The ability to "copy, cut, and paste" blocks of messages here and there is a fundamental ingredient in the generic second preimage attacks, and the inclusion of a "round counter" in the compression function seems very effective at preventing them, although Biham and Dunkelman did not give any security proof. HAIFA is immune to the length extension attack because of the "last-block" special padding, and is thus provably indifferentiable up to $2^{n/2}$ queries. However, the multicollision and the herding attack both apply, thus showing that this bound is tight. We will show in §7.1.2 that HAIFA is in fact *provably* resistant to *generic* second preimage attacks up to 2^n queries. HAIFA has had a great success, and it is implemented in five second round SHA-3 candidates: BLAKE, ECHO, Shabal, SHAvite-3 and Skein.

Algorithm 3.5 The HAIFA mode of operation.

1: **function** HAIFA(M)
2: **Pad** and **split** the message M into r blocks $m_0, \ldots, m_r$ of m bits each.
3: **let** h_{-1} be the initialization value IV.
4: **for each** message block m_i, $i < r$ compute $h_i = f(h_{i-1}, m_i, i)$.
5: compute $h_r = f(h_{r-1}, m_r, -1)$.
6: **return** $H^f(M) = h_r$.
7: **end function**



3.4.2 Rivest's Dithered Hashing

Even before HAIFA had been presented to the community, Rivest had come out with the idea of using a counter to make the hashing process more "irregular". However he was concerned about the loss of efficiency that this would incur. When a "normal" compression function is used in a HAIFA hash function, then κ bits of its input would be dedicated to the round counter (which is about 10% of the total input in the MD and SHA family), leading in a corresponding decrease in hashing speed. If a compression were to be designed from scratch to accommodate the counter, then there would still be more input bits to mix together.

To avoid this problem, Rivest suggested in 2005 to use *Dithered hashing* [Riv05]. Instead of dedicating κ bits of compression-function input to a counter, we would still perturbate the hash process by an additional input to the compression function, but we would try to make this additional input as small as possible. Rivest's idea was to use the consecutive elements of a fixed *dithering* sequence. If the dithering sequence is well-chosen, then this would still make the hash of a message block dependent on its position in the whole message.

Since the dithering sequence has to be at least as long as the maximal number of blocks in any message that can be processed by the hash function, it makes sense to consider infinite potential dithering sequences. Let $\mathcal{A}$ be a finite alphabet, and let the dithering sequence $\mathbf{z}$ be an eventually infinite word over $\mathcal{A}$. Let $\mathbf{z}[i]$ denote the i-th element of $\mathbf{z}$. The *dithered* Merkle-Damgård construction iterates a compression function $f : \{0,1\}^n \times \{0,1\}^* \times \mathcal{A} \to \{0,1\}^n$, by setting $h_i = f(h_{i-1}, m_i, \mathbf{z}[i])$ in the definition of the original Merkle-Damgård scheme.

Words and Sequences. Let ω be a word over a finite alphabet $\mathcal{A}$. We use the dot operator to denote concatenation. If ω can be written as $\omega = x.y.z$ (where x,y, or z can be empty), we say that x is a *prefix* of ω and that y is a *factor* of ω. A finite non-empty word ω is a *square* if it can be written as $\omega = x.x$, where x is not empty. A finite word ω is an *abelian square* if it can be written as $\omega = x.x'$ where x' is a permutation of x (*i.e.*, a reordering of the letters of x). A word is said to be *square-free* (respectively, *abelian square-free*) if none of its factors is a square (respectively, an abelian square). Note that abelian square-free words are also square-free.

Sequences Generated by Morphisms. We say that a function $\tau : \mathcal{A}^* \to \mathcal{A}^*$ is a *morphism* if for all words x and y, $\tau(x.y) = \tau(x).\tau(y)$. A morphism is then entirely determined by the images of the individuals letters. A morphism is said to be *r-uniform* (with $r \in \mathbb{N}$) if $|\tau(x)| = r \cdot |x|$ for any word x. If, for a given letter $\alpha \in \mathcal{A}$, we have $\tau(\alpha) = \alpha.x$ for some word x, then τ is *non-erasing* for α. Given a morphism τ and an initialization letter α, let $u_n = \tau^n(\alpha)$ denote the n-th iterate of τ over α. If τ is r-uniform (with $r \geq 2$) and non-erasing for α, then u_n is a strict prefix of u_{n+1}, for all $n \in \mathbb{N}$. Let $\tau^\infty(\alpha)$ denote the limit of this sequence: it is the only fixed point of τ that begins with the letter α. Such infinite sequences are called *uniform tag sequences* [Cob72] or *r-automatic* sequences [All94].

An Infinite Abelian Square-Free Sequence. Infinite square-free sequences have been known to exist since 1906, when Axel Thue exhibited the Thue-Morse word over a ternary alphabet (there are no square-free sequences of size greater than four on a binary alphabet).

The question of the existence of infinite *abelian* square-free sequences was raised by 1961 by Erdös, and was solved by Pleasants [Ple70] in 1970: he exhibited an infinite abelian square-free sequence over a five-letter alphabet. In 1992, Keränen [Ker92] exhibited an infinite abelian square-free sequence $\mathbf{k}$ over a four-letter alphabet (there are no infinite abelian square-free words over a ternary alphabet). In the sequel, we call this infinite abelian square-free word the *Keränen sequence*. Before describing it, let us consider the permutation σ over $\mathcal{A}$ defined by:

$$\sigma(a) = b, \quad \sigma(b) = c, \quad \sigma(c) = d, \quad \sigma(d) = a$$

Surprisingly enough, the Keränen sequence is defined as the fixed point of a 85-uniform morphism τ, given by:

$$\tau(a) = \omega_a, \quad \tau(b) = \sigma(\omega_a), \quad \tau(c) = \sigma^2(\omega_a), \quad \tau(d) = \sigma^3(\omega_a),$$

where ω_a is some magic string of size 85 (given in [Ker92, Riv05]).

Keränen-DMD. In [Riv05] Rivest suggests to directly use the Keränen sequence as a source of dithering inputs. The dithering inputs are taken from the alphabet $\mathcal{A} = \{a, b, c, d\}$, and can be encoded by two bits. The introduction of dithering thus only increases number of data bits in the input of the compression function by only two bits, which improves the hashing efficiency (compared to longer encodings of dither inputs). It is possible to generate the Keränen sequence online, one symbol at a time, in logarithmic space and constant amortized time.

Rivest's Concrete Proposal. To speed up the generation of the dithering sequence, Rivest proposed a slightly modified scheme, in which the dithering symbols are 16-bit wide. Rivest's concrete proposal, which we refer to as DMD-CP (Dithered Merkle-Damgård–Concrete Proposal) reduces the need to generate the next letter from the Keränen sequence. If the message M is r blocks long, then for $1 \leq i < r$ the i-th dithering symbol has the form:

$$\left(0, \mathbf{k}\left[\left\lfloor i/2^{13} \right\rfloor\right], i \bmod 2^{13}\right) \in \{0,1\} \times \mathcal{A} \times \{0,1\}^{13}$$

The idea is to increment the counter for each dithering symbol, and to shift to the next letter in the Keränen sequence, only when the counter overflows. This "diluted" dithering sequence can essentially be generated 2^{13} times faster than the Keränen sequence. The last dithering symbol has a different form (recall that m is the number of bits in a message block):

$$(1, |M| \bmod m) \in \{0,1\} \times \{0,1\}^{15}$$

The dithered Merkle-Damgård construction seems immune to the generic second preimage attacks of Dean (§3.3.3) and Kelsey and Schneier (§3.3.4), but the multicollision and herding attack do apply as-is. We also give a second-preimage attack in §4.2.

3.4.3 Shoup's Universal One-Way Hash Function.

As mentioned in §3.1, a *Universal One-Way Hash Function* (UOWHF for short) is a keyed function $H : \mathcal{K} \times \{0,1\}^* \to \{0,1\}^n$ required to satisfy the eSec security notion. The best possible UOWHFs will resist eSec adversaries up to 2^n queries.

Bellare and Rogaway studied the construction of variable input length UOWHF from *fixed input length* UOWHFs (*i.e.*, eSec-secure keyed compression functions) [BR97]. They also demonstrated that UOWHFs are sufficient for a number of signature applications. Shoup [Sho00a] improved on the former constructions by proposing a simpler scheme that also yields shorter keys (by a constant factor). It is a Merkle-Damgård-like mode of operation, but before every compression function evaluation in the iteration, the state is updated by XORing one out of a small set of possible masks into the chaining value. The number of masks is logarithmic in the length of the hashed message, and the order in which they are used is carefully chosen to maximize the security of the scheme. This is reminiscent of dithered hashing, except that here the dithering process does not decrease the bandwidth available to actual data (it just takes a few more operations).

Algorithm 3.6 Shoup's Universal One-Way Hash Function

1: **function** SHOUP(K, M)
2: **let** $(k, \mu_0, \ldots, \mu_\kappa) = K$ (the key of the iterated function)
3: **pad** and **split** the message M into r blocks $m_0, \ldots, m_r$ of m bits each.
4: **let** h_{-1} be the initialization value IV.
5: **for each** message block m_i compute $h_i = f_k\left(h_{i-1} \oplus \mu_{\nu_2(i)}, m_i\right)$.
6: **return** $H_K(M) = h_r$.
7: **end function**

Shoup's construction works just like Merkle-Damgård by iterating an eSec-secure compression function family $f_k : \{0,1\}^n \times \{0,1\}^m \to \{0,1\}^n$, as described in Algorithm 3.6, to obtain a variable input length UOWHF.

The scheme uses a set of masks $\mu_0, \ldots, \mu_{\kappa-1}$ (where $2^\kappa - 1$ is the length of the longest possible message), each one of which is a random n-bit string. The key of the whole iterated function consists of k and of these masks. The order in which the masks are applied is defined by a specified sequence over the alphabet $\mathcal{A} = \{0, \ldots, \kappa - 1\}$. The scheduling sequence is $\mathbf{z}[i] = \nu_2(i)$, for $1 \leq i \leq 2^\kappa$, where $\nu_2(i)$ denotes the largest integer ν such that 2^ν divides i. Similarly to the Merkle-Damgård construction, Shoup's UOWHF enjoys a nice security property.

Theorem 3.9 ([Sho00a]). *If an adversary is able to break the eSec[λ] notion of H^f with probability ε in time T, then one can construct an adversary that breaks the eSec notion of f in time $T + \mathcal{O}(\lambda)$, with probability ε/λ.*

In a nutshell, if no adversary can break the eSec property of f in time less than 2^n queries, then no adversary can break the eSec property of H^f in time less than $2^{n-\kappa}$ queries. In other words, Shoup's hash function family is reasonably close to the original Merkle-Damgård construction yet *provably* enjoys a flavor of second-preimage resistance in the standard model. We give an eSec-attack on the UOWHF in §4.3, showing in passing that the security bound is tight.

An intriguing connection between Shoup's and Rivest's hash functions shows up as soon as we notice that the scheduling sequence $\mathbf{z}$ chosen by Shoup is abelian square-free. In fact, one year after Shoup's construction was published, Mironov proved that an even stronger notion of repetition-freeness was necessary: $\mathbf{z}$ is, and has to be, *even-free*[Mir01]. A word is even-free if all of its non-empty factors contain at least one letter an odd number of times. It is easy to see that even-free words are abelian square-free.

3.4.4 The Wide-Pipe Construction

Faced with the ever-more worrying flow of new generic attacks, Lucks suggested in 2005 a completely different approach [Luc05]. His idea is quite simple: most generic attacks rely on the ability to find *internal state* collisions, and exploit these collisions in a clever way. If finding these collisions is impossible in the first place, then there would not be any generic attack. Making internal state collisions impossible to find in less than 2^n compression function evaluations is simply achieved by *doubling the size of the internal state*. To produce n-bit hash values, a compression function $f : \{0,1\}^{2n} \times \{0,1\}^m \to \{0,1\}^{2n}$ is iterated, resulting in a $2n$-bit final internal chaining value which is subsequently brought down to n bits by a finalization function g (which usually discards half of the bits). This is illustrated by Figure 3.4. When g is simply a truncation, then the resulting scheme is sometimes called "Chop-Merkle-Damgård". From a historical perspective, CELLHASH and Subterranean, two earlier designs of Daemen are "wide-pipe" and thus predate by about 15 years the more systematic proposal of Lucks.

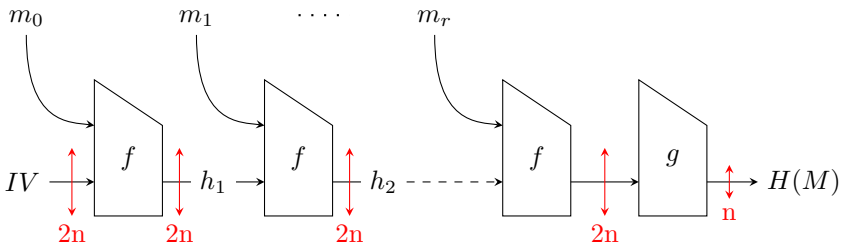


Figure 3.4: The "double-pipe" Merkle-Damgård hash function.

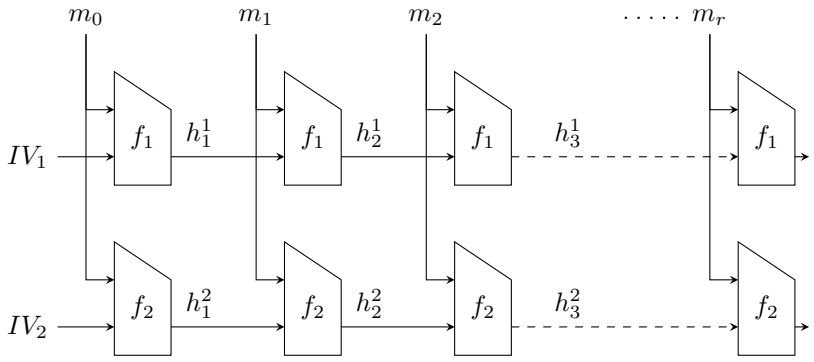


Figure 3.5: Concatenated hashing.

Intuitively, all the attacks of §3.3 cannot be replayed, because finding collisions on the compression function takes longer than inverting the full hash function. More formally, it is shown in [Luc05] that any k-way multicollision (resp. second preimage) on H results in either a collision on f or a k-way multicollision on $g \circ f$ (resp. a collision on f or a second preimage on $g \circ f$), which shows that as long as f and $g \circ f$ are secure, then H is free of multicollision and second-preimage adversaries.

It is possible to make an even stronger point in the indifferentiability framework. Coron *et al.* have shown that the double-pipe is indifferentiable up to $2^{n/2}$ queries, which does not discard multicollision and second-preimage attacks, but Chang and Nandi proved that the mode of operation is in fact indifferentiable up to $2^n/n$ queries, thus showing that no generic attack applies below this bound [CN08]. This bound was subsequently improved to $2^n/2$ queries —which is asymptotically tight— by Bertoni, Daemen, Peeters and Van Assche in [BDPA09], and by Daemen, Dusenge and Van Assche in [DDA11]. These authors describe several sufficient conditions that a mode of operation must meet to be covered by their result, which is therefore very general.

The wide-pipe mode of operation is therefore *provably secure against all generic attacks*. It also enjoys some other theoretical advantages. For instance, consider the Merkle-Damgård iteration of a compression function f. If f falls to a pseudo-preimage attack of time complexity T (given y, an attacker can find h and m such that $f(h, m) = y$), then the full H is susceptible of a Meet-in-the-Middle preimage attack of complexity $\sqrt{4T} \cdot 2^{n/2}$. This is for instance how a break of the MD4 compression function led to a break of the full hash function [Leu08a]. Such a lifting is impossible on double-pipe hash functions.

3.4.5 Concatenated Hashing

A simple way to build a hash function would simply be... to concatenate the output of two existing hash functions. The implicit hope is that even if one of the hash function breaks, the other one would resist (or, at the very least, not break in the same way). This idea was apparently first suggest by Preneel in his PhD thesis [Pre93], and variants thereof made their ways into several standards. For instance, in TLS a pseudo-random function family is created by considering the XOR of SHA-1 and MD5 [DA99]. We consider the concatenation $\mathcal{CH}$ of k Merkle-Damgård hash function (resulting in a bigger Merkle-Damgård hash):

$$\mathcal{CH}(M) = H^{f_1}(IV_1, M) \,\|\, H^{f_2}(IV_2, M) \,\|\, \ldots \,\|\, H^{f_k}(IV_k, M).$$

Fig 3.5 illustrates the construction with $k = 2$. Unfortunately, concatenating two iterated n-bit hash functions does not result in anything near a secure $2n$-bits hash function.

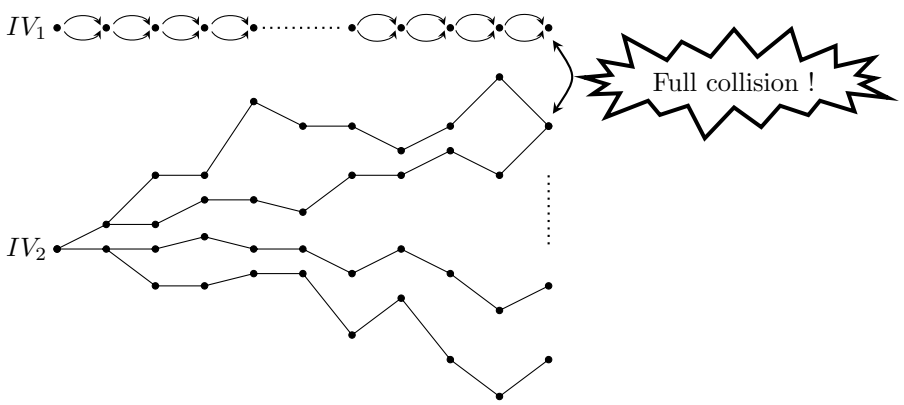


Figure 3.6: Joux's attack against concatenated hashes

3.4.5.1 Joux's Collision Attack

We describe the collision attack of [Jou04] against the concatenated hash $\mathcal{CH}$ with two pipes. Starting from two fixed chaining values IV_1 and IV_2 in the two pipes, the adversary first finds a $2^{n/2}$-way multicollision (using the attack of §3.3.2) for the first hash function H^{f_1}. The adversary then evaluates H^{f_2} on the $2^{n/2}$ messages of the multicollision, all yielding the same chaining value for H^{f_1}, while yielding a set of $2^{n/2}$ chaining values for H^{f_2}, as shown in Figure 3.6. The adversary then looks for the expected collision in this set. To construct a 2^{ℓ}-*multicollision* on the two pipes, just replay Joux's attack using the two-pipe collision finding algorithm described above ℓ times.

Joux also shows that this idea can be extended to find (multi)collisions in the concatenation of an arbitrary number k of hash functions. To build a collision on k parallel pipes, the adversary proceeds inductively: first construct a $2^{n/2}$-way multicollision on the first $k-1$ pipes and hash the $2^{n/2}$ messages in the last pipe. Then, by the birthday bound, a collision is expected amongst the set of $2^{n/2}$ values generated in the last pipe. This collision is present in all the previous $k-1$ pipes, and hence results in a full collision on all the k pipes.

The cost of building a collision on k pipes is the cost of building the multicollision, plus the cost of hashing the $2^{n/2}$ messages of length $(n/2)^{k-1}$. Solving the recurrence yields a time complexity of $k \cdot (n/2)^{k-1} \cdot 2^{n/2}$ compression function calls. More generally, the complexity of building a 2^{ℓ}-way multicollision on k pipes is exactly ℓ times the preceding expression, or $\ell \cdot k \cdot (n/2)^{k-1} \cdot 2^{n/2}$.

Preimages on the concatenation can also be found for roughly 2^n compression function evaluations using similar techniques [Jou04]. We give a herding attack in §5.1.

3.4.5.2 Some Form of Provable Security

The intuition that the concatenation $\mathcal{CH} = H^{f_1} \| H^{f_2}$ might be secure even when f_1 and f_2 are very badly broken compression functions contains some truth. Hoch and Shamir have studied the security of this concatenation under the assumption that f_1 and f_2 are *weak compression functions* [HS08]. In this setting, f_1 and f_2 are public random functions, but the adversary has access to *inversion oracles* that finds preimages in unit time for either f_1 or f_2. More specifically the adversary has access to the following oracles:

- $(h, ?, y) \to m$ such that $f_1(h, m) = y$
- $(h, ?, y) \to m$ such that $f_2(h, m) = y$
- $(?, m, y) \to h$ such that $f_1(h, m) = y$
- $(?, m, y) \to h$ such that $f_1(h, m) = y$

The oracles return $\perp$ if no solution exist, otherwise they choose uniformly at random a possible answer. Observe that H^{f_1} and H^{f_2} are both very badly broken when taken individually. Using the inversions oracle, an attacker can forge preimages for the iterated hash functions in unit time. Their concatenation is nevertheless much more secure.

Theorem 3.10 ([HS08])**.** *Let $\odot$ denote any group operation (including concatenation on the free group), and let f_1, f_2 be two weak compression functions. Then the concatenated hash function defined by $H(M) = H^{f_1}(M) \odot H^{f_2}(M)$ is indifferentiable from a random oracle with $q \ll 2^{n/2}$ queries, even in the presence of the inversion oracles.*

The attack of Joux spun a wider interest in *hash function combiners*, *i.e.*, simple ways of turning several hash functions into a single and potentially strong one in a black-box way. Boneh and Boyen first considered the case of $H_1 \| H_2$, and showed in 2006 that any secure construction that evaluates H_1 and H_2 once cannot

output fewer bits than simply concatenating their outputs [BB06]. They left as an open problem to study the combination of ℓ hash functions amongst which k are collision resistant. The concatenation of the output of $\ell - k + 1$ of them is collision resistant by definition, and Pietrzak showed that no secure combiner can output less bits [Pie07]. Lastly, Lehmann and Fischlin observed that restricting the input domain of $H_1 \| H_2$ to messages of at most $n/4$ blocks thwarts Joux's attack [FL07]. They showed that in this case, any adversary allowed to find a polynomial number of collisions on the compression functions cannot find a collision on the concatenation. The limitation on the input domain can be mitigated by using hash trees (cf §3.5.1).

3.5 Other Hash Function Modes of Operation

We move on to the "Oddities Museum" section of our hash function modes of operation zoo. We describe some hash function modes of operation that are not very similar to Merkle-Damgård.

3.5.1 Tree Hashes

Tree hashes were first suggested in [Mer89]. Let $f : \{0,1\}^n \times \{0,1\}^n \to \{0,1\}^n$ be a compression function used in the tree hash H^f. An ℓ-bit message M is initially padded with a single "1" bit and as many "0" bits as needed to obtain $\mathrm{pad}_{\mathrm{TH}}(M) = m_1, m_2, \ldots, m_L$, where each m_i is n-bit long, $L = 2^\kappa$ for $\kappa = \lceil \log_2(\ell + 1)/n \rceil$. Consider the resulting message blocks as the leaves of a full binary tree of depth κ. Then, the compression function is applied to any two leaves with a common ancestor, and its output is assigned to the common ancestor. This procedure is followed in an iterative manner. A final compression function is applied to the output of the root and an extra final strengthening block, normally containing the length of the input message M. The resulting output is the final tree hash. Formally, the tree hash function $H^f(M)$ is defined by Algorithm 3.7. The main advantage of tree hashes is that the hash function can be evaluated in parallel. Tree hashes are inherently victims of a long-message second preimage attack, unless specific countermeasures prevent it.

Algorithm 3.7 Tree Hashing

function Tree-Hash(M)
  $m_1, m_2, \ldots, m_L \leftarrow \mathrm{pad}_{\mathrm{TH}}(M)$
  for $j = 1$ to $2^{\kappa-1}$ **do** compute $h_{1,j} = f(m_{2j-1}, m_{2j})$
  for $i = 2$ to κ **do**
    for $j = 1$ to $2^{\kappa-i}$ **do** compute $h_{i,j} = f(h_{i-1,2j-1}, h_{i-1,2j})$
  end for
  return $H^f(M) = f(h_{\kappa,1}, |M|)$.
end function

3.5.2 Merkle-Damgård-Again

> Hash twice, double the fun!
>
> Hash function folklore

The "Hash-Twice" constructions, aka Merkle-Damgård-Again, is a folklore *non-streamable* mode of operation, meaning that the whole message must be present in memory during the whole hash process. The idea is quite simple: hash a message using the Merkle-Damgård iteration of some compression function, then define the IV to be the resulting hash value, and hash the message again! Formally, we define

$$H = f^*\big(f^*(IV, M), M\big),$$

where M is appropriately padded. Figure 3.7 illustrates the construction. This construction does not enjoy any security result that we are aware of, but it seems remarkably resistant: none of the generic attacks described in §3.3 apply as-is. However, a generalized version of Joux's multicollision does apply [HS06, NS07] with essentially the same complexity. In addition, we describe a herding attack in §5.2 and a second preimage attack in §5.3, with essentially the same complexity as their Merkle-Damgård counterparts.

3.5.3 The Zipper hash

The "Zipper" mode of operation has been proposed by Liskov in 2006, and is also a non-streamable mode of operation quite similar to Merkle-Damgård-Again. Let f_1, f_2 be two compression functions, and let M

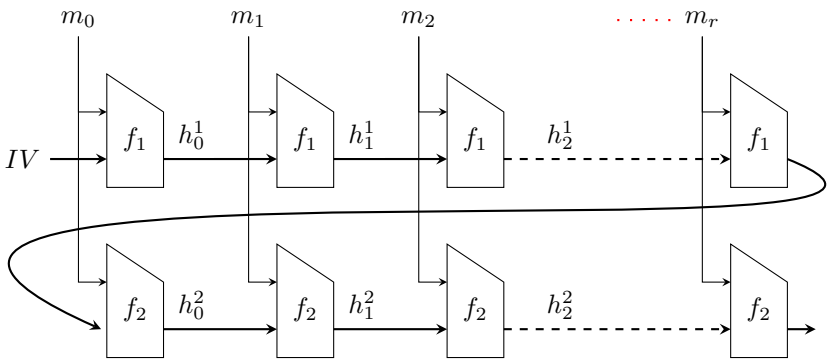


Figure 3.7: Merkle-Damgård-Again.

an appropriately padded message. We denote by $\widetilde{M}$ the message M in which the blocks are written in the opposite order. The zipper hash $\mathcal{ZH}$ is defined as:

$$\mathcal{ZH}(M) = f_2^*\left(f_1^*(IV, M), \widetilde{M}\right).$$

Liskov proved that this apparently bad construction enjoys a very interesting security feature, similar to that of the concatenated hashing (cf. §3.4.5), namely that the construction is secure even if the two compression functions are very weak.

Theorem 3.11 ([Lis06])**.** *Let f_1, f_2 be two weak compression functions. Then the Zipper hash function $\mathcal{ZH}$ is indifferentiable from a random oracle with $q \ll 2^{n/2}$ queries, even in the presence of the inversion oracles.*

This bound cannot be improved because the generalized variant of Joux's multicollision attack does apply [HS08, NS07]. We also give a herding attack in §5.2.1.

Chapter 4

New Generic Second Preimage Attacks

In this chapter we present a new generic second preimage attack against the Merkle-Damgård mode of operation and some derivatives. This new attack in particular breaks Rivest's Dithered hashing, which was specifically designed to avoid generic second preimage attacks. This work led to a joint publication at EUROCRYPT 2008, along with Elena Andreeva, Orr Dunkelman, Johnathan Hoch, John Kelsey, Adi Shamir and Sebastien Zimmer [ABF+08]. Andreeva, Dunkelman and Kelsey had in fact independently discovered the attack at the exact same time we did.

This chapter describes a new generic second preimage attack against Merkle-Damgård. It relies heavily on the diamond structure introduced by Kelsey and Kohno [KK06] and described in §3.3.5. On the plain Merkle-Damgård construction, the new attack does not really improve on the previously-known generic second preimage attacks of Dean [Dea99] and Kelsey-Schneier [KS05]. The new attack is however much more flexible, and can be applied to a broader spectrum of hash functions. We first describe the new attack in its simplest form against the plain Merkle-Damgård mode of operation in §4.1. We then show that it breaks Rivest's dithered hashing in §4.2, and that it matches the provable security bound on Shoup's UOWHF in §4.3. A refined, more sophisticated version of the attack breaks the folklore Merkle-Damgård-again mode of operation, but as this require some more machinery, we delay its exposition to chapter 5.

4.1 A New Generic Second Preimage Attack Against Merkle-Damgård

Our new technique to find second preimages on Merkle-Damgård hash functions relies on the "diamond structure" of the herding attack that was described in §3.3.5.

The idea of the attack can be summarized as follows. Assume we have performed the offline phase of the herding attack, and have computed a diamond structure with 2^ℓ "leaves" (or external nodes) and such that the chaining value at the root is $h_\diamond$. This can be done with about $2^{n/2+\ell/2+2}$ compression function evaluations. Then, given a target message M of length 2^κ blocks, connecting $h_\diamond$ to one of the 2^κ chaining values encountered during the computation of $H(M)$ takes only $2^{n-\kappa}$ compression function calls. Then, connecting an arbitrary prefix P of the right size to the diamond structure takes time $2^{n-\ell}$, and leads to a successful second preimage forgery, since all the steps have complexity less than 2^n. The attack is illustrated and more precisely described in Algorithm 4.1.

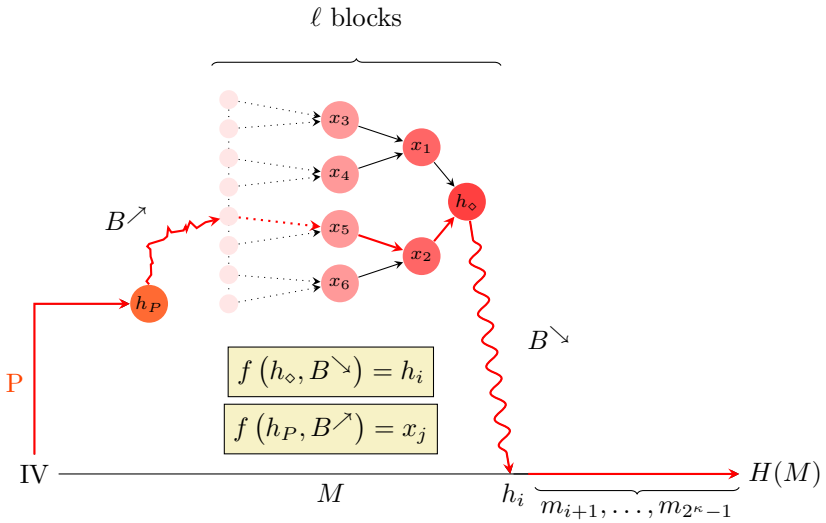
Complexity. The first step allows for precomputation and its time and space complexity is about $2^{(n+\ell)/2+2}$ (see §3.3.5). The second step of the attack is carried out online with $2^{n-\kappa}$ work, and the third step takes $2^{n-\ell}$ work. The total time complexity of the attack is then $2^{(n+\ell)/2+2}$ precomputation and $2^{n-\kappa}+2^{n-\ell}$ online computations and their sum is minimal when $\ell=(n-4)/3$ for a total of about $5\cdot 2^{2n/3}+2^{n-\kappa}$ computations.

Comparison with Previously Existing Generic Attacks. The attacks of [Dea99, KS05] are slightly more efficient than ours. We present the respective offline and online complexities for the old and new variants of the attack in Table 4.1 and we compare the attacks for MD5 ($n=128,\kappa=55$), SHA-1 ($n=160,\kappa=55$), SHA-256 ($n=256,\kappa=118$), and SHA-512 ($n=512,\kappa=118$) in Table 4.2. Still, our technique gives the adversary more control over the second preimage. She may choose a large chunk of the forgery, typically one half. For example, she could choose to reuse most of the target message, leading to a second preimage that differs from the original by only $\ell+2$ blocks.

The main difference between the older techniques and ours is that the previous attacks build on the use of expandable messages. We note that our attack just offers a short patch. At the same time, our attack can also be viewed as a new, more flexible technique to build expandable messages, by choosing a prefix of the appropriate length and connecting it to the collision tree. This can be done in time $2^{(n+\ell)/2+2}+2^{n-\ell}$. Although it is more expensive, this new technique can be adapted to work even when an additional dithering input is given, as we shall now demonstrate.

Algorithm 4.1 Summary of our new attack on the plain Merkle-Damgård construction.

1. Construct a collision tree of depth ℓ with a final chaining value at the root $h_\diamond$.
2. Connect $h_\diamond$ to some intermediate chaining value in the target message M. This is done by generating random message blocks B, until $f(h_\diamond, B) = h_i$ for some i, such that $\ell + 1 \leq i < |M|$. Let $B^{\searrow}$ be the message block satisfying this condition.
3. Generate an arbitrary prefix P of size $i - \ell - 2$ blocks, and assume that it hashes to h_P, then "connect" it to the diamond. This can be done by generating random message blocks B, until $f(h_P, B) = x_j$, where x_j is a chaining value labeling a leaf of the diamond. Let $B^{\nearrow}$ denote this block, and let T be the chain of ℓ blocks traversing the diamond from x_j to $h_\diamond$.
4. Form a message $M' = P||B^{\nearrow}||T||B^{\searrow}||m_{i+1}, \ldots, m_{2^\kappa - 1}$.



Attack	Complexity			Avg. Patch Size	Message Length
	Offline	Online	Memory		
Dean*	$2^{n/2+1}$	$2^{n-\kappa}$	2	$2^{\kappa-1}$	2^κ
Kelsey-Schneier	$\kappa \cdot 2^{n/2+1} + 2^\kappa$	$2^{n-\kappa}$	$2 \cdot \kappa$	$2^{\kappa-1}$	2^κ
New	$2^{(n+\ell)/2+2}$	$2^{n-\ell} + 2^{n-\kappa}$	$2^{\ell+1}$	$\ell + 2$	2^κ
Using TMDTO (§. 6.2)	$2^{(n+\ell)/2+2} + 2^{n-\lambda}$	$2^{n-\ell} + 2^{2\lambda}$	$2^{\ell+1} + 2^{n-2\lambda}$	$\ell + 2$	2^λ

* — This attack assumes the existence of easily found fixed points in the compression function

Table 4.1: Comparison of Long Message Second Preimage Attacks

4.2 Application to Dithered Hashing

In this section we describe a generic second preimage attack against Rivest's Dithered Hashing. This mode of operation was explicitly designed to avoid the existing second preimage attack (this was the motivation for choosing an abelian square-free dithering sequence). Before describing the attack itself, we investigate several properties of the dithering sequence.

4.2.1 More Sequence Background

The number of distinct factors of a given size of an infinite word gives an intuitive notion of its *complexity*: a sequence is more complex (or richer) if it possesses a large number of different factors. We denote by $Fact_{\mathbf{z}}(\ell)$ the number of factors of size ℓ of the sequence $\mathbf{z}$. Because they have a very strong structure, r-uniform sequences have special properties, especially with regard to their complexity.

Theorem 4.1 (Cobham, 1972, [Cob72]). *Let $\mathbf{z}$ be an infinite sequence generated by an r-uniform morphism, and assume that the alphabet size $|\mathcal{A}|$ is finite. Then $\mathbf{z}$ has* linear *complexity bounded by:*

$$Fact_{\mathbf{z}}(\ell) \leq r \cdot |\mathcal{A}|^2 \cdot \ell.$$

Function (n, κ)		MD5 (128,55)	SHA-1 (160,55)	SHA-256 (256,118)	SHA-512 (512,118)
Dean	Offline:	2^{65}	2^{81}	2^{129}	2^{257}
	Online:	2^{73}	2^{105}	2^{138}	2^{394}
	Memory:	2	2	2	2
	Patch:	2^{54}	2^{54}	2^{117}	2^{117}
Kelsey-Schneier	Offline:	2^{71}	2^{87}	2^{136}	2^{264}
	Online:	2^{73}	2^{105}	2^{138}	2^{394}
	Memory:	110	110	234	234
	Patch:	2^{54}	2^{54}	2^{117}	2^{117}
New	Offline:	$2^{93.5}$	$2^{109.5}$	2^{189}	2^{317}
	Online:	2^{74}	2^{106}	2^{139}	2^{395}
	Memory:	2^{56}	2^{56}	2^{119}	2^{119}
	Patch:	57	57	120	120
First connection with TMDTO	Offline:	$2^{98.3}$	$2^{122.3}$	$2^{194.3}$	2^{394}
	Online:	2^{65}	2^{81}	2^{129}	2^{257}
	Memory:	$2^{65.6}$	$2^{81.6}$	$2^{129.6}$	$2^{257.6}$
	Patch:	66	82	130	258
	Length:	2^{32}	2^{40}	2^{64}	2^{118}

The values were optimized for minimal online complexity.
TMDTO values were optimized for equal online and memory complexities.
Memory and patches are measured in blocks.
Length is given for cases where the message length may be shorter than 2^{κ}.

Table 4.2: Comparison of the long-message second preimage attacks on real hash functions

A polynomial algorithm which computes the exact set of factors of a given length ℓ can be deduced from the proof of this theorem. It is worth mentioning that similar results exist in the case of sequences generated by non-uniform morphisms [ELR75, Pan84], although the upper bound can be quadratic in ℓ. The bound given by this theorem, although attained by certain sequences, is relatively rough. For example, since the Keraïen sequence is 85-uniform, the theorem gives $Fact_{\mathbf{k}}(\ell) \leq 1360 \cdot \ell$. For $\ell = 50$, this gives $Fact_{\mathbf{k}}(50) \leq 68000$, while the factor-counting algorithm reveals that $Fact_{\mathbf{k}}(50) = 732$. Hence, for small values of ℓ, the following more *ad hoc* upper bound may be tighter:

Lemma 4.2. *Let* $\mathbf{z}$ *be an infinite sequence over the alphabet* $\mathcal{A}$ *generated by an* r*-uniform morphism* τ*. For all* ℓ*,* $1 \leq \ell \leq r$*, we have :*

$$Fact_{\mathbf{z}}(\ell) \leq \ell \cdot \Big(Fact_{\mathbf{z}}(2) - |\mathcal{A}|\Big) + \Big[(r+1) \cdot |\mathcal{A}| - Fact_{\mathbf{z}}(2)\Big].$$

Proof. If $\ell \leq r$, then any factor of $\mathbf{z}$ of size ℓ falls in one of these two classes:
- Either it is a factor of $\tau(\alpha)$ for some letter $\alpha \in \mathcal{A}$. There are no more than $|A| \cdot (r - \ell + 1)$ such factors.
- Or it is a factor of $\tau(\alpha).\tau(\beta)$, for two letters $\alpha, \beta \in \mathcal{A}$ (and is not a factor of either $\tau(\alpha)$ or $\tau(\beta)$). For any given pair (α, β), there can only be $\ell - 1$ such factors. Moreover, $\alpha.\beta$ must be a factor of size 2 of $\mathbf{z}$.

So $Fact_{\mathbf{z}}(\ell) \leq |A| \cdot (r - \ell + 1) + Fact_{\mathbf{z}}(2) \cdot (\ell - 1)$. □

For the particular case of the Keränen sequence $\mathbf{k}$, we have $r = 85$, $\big|\mathcal{A}\big| = 4$ and $Fact_{\mathbf{k}}(2) = 12$ (all non-repeating pairs of letters). This yields $Fact_{\mathbf{k}}(\ell) \leq 8 \cdot \ell + 332$ when $\ell \leq 85$, which is tight, as for $\ell = 50$ it gives: $Fact_{\mathbf{k}}(50) \leq 732$.

4.2.1.1 Factor Frequency

Formally, let us denote by $N_\omega(x)$ the number of occurrences of ω in x (which is expected to be a finite word), and by $\mathbf{z}[1..i]$ the prefix of $\mathbf{z}$ of size i. The *frequency* of a given word ω in the sequence $\mathbf{z}$ is the limit of $N_\omega(\mathbf{z}[1..i])/i$ when i goes to $+\infty$.

We will see later on that we will have to choose a given factor of the Keränen sequence in the course of our attack against Dithered Hashing, and that the complexity of the attack will depends on the frequency this factor. If the frequency of the various factors is non uniform, then the attack should exploit this bias (just like any cryptographic attack).

We denote by $2^{-H_\infty(\mathbf{z},\ell)}$ the frequency of the most frequent factor of length ℓ in the sequence $\mathbf{z}$. It follows immediately that $H_\infty(\mathbf{z},\ell) \leq \log_2 Fact_{\mathbf{z}}(\ell)$. Hence, when the computation of $H_\infty(\mathbf{z},\ell)$ is infeasible, $\log_2 Fact_{\mathbf{z}}(\ell)$ can be used as an upper-bound. It is however possible to determine precisely the frequency of certain words in sequences generated by uniform morphisms. For instance, it is easy to compute the frequency of individual letters in the sequence generated by a morphism τ: if x is some finite word and $\alpha \in \mathcal{A}$, then by definition we find:

$$N_\alpha(\tau(x)) = \sum_{\beta \in \mathcal{A}} N_\alpha(\tau(\beta)) \cdot N_\beta(x) \tag{4.1}$$

In this formula, $N_\alpha(\tau(\beta))$ is easy to determine from the description of the morphism τ. We define:

$$\begin{aligned} \mathcal{A} &= \{\alpha_1, \ldots, \alpha_k\} \\ U_s &= \left(\frac{N_{\alpha_j}(\tau^s(a))}{\ell^s}\right)_{1 \leq j \leq |\mathcal{A}|} \\ M &= \left(\frac{N_{\alpha_i}(\tau(\alpha_j))}{\ell}\right)_{1 \leq i,j \leq |\mathcal{A}|} \end{aligned}$$

It follows from equation (4.1) that:

$$U_{s+1} = M \cdot U_s.$$

The frequency of individual letters is given by the vector $U_\infty = \lim_{s\to\infty} U_s$. Fortunately, this vector lies in the kernel of $M - I_k$ (and is such that its component sum up to one). For instance, for the Keränen sequence, and because of the very symmetric nature of τ, we find that M_1 is a circulant matrix:

$$85 \cdot M = \begin{pmatrix} 19 & 18 & 27 & 21 \\ 21 & 19 & 18 & 27 \\ 27 & 21 & 19 & 18 \\ 18 & 27 & 21 & 19 \end{pmatrix}$$

We quickly obtain: $U_\infty = \frac{1}{4}(1,1,1,1)$, meaning that no letter occurs more frequently than the other. The frequencies of digrams (*i.e.*, two-letters words) are slightly more complicated to compute, as the digram formed from the last letter of $\tau(\alpha)$ and the first letter of $\tau(\beta)$ is automatically a factor of $\tau(\alpha\beta)$ but is not necessarily a factor of either $\tau(\alpha)$ or $\tau(\beta)$ individually. We therefore need a new version of equation (4.1) that takes this fact into account. Let us define $\Omega_2 = \{\omega_1, \ldots, \omega_r\}$, the set of factors of size two of $\mathbf{z}$. If ω is such a factor, we obtain:

$$N_\omega(\tau(x)) = \sum_{\gamma \in \mathcal{A}} N_\omega(\tau(\gamma)) \cdot N_\gamma(x) + \sum_{\omega_j \in \Omega_2} \left[N_\omega(\tau(\omega_j)) - N_\omega(\tau(\omega_j[1])) - N_\omega(\tau(\omega_j[2]))\right] \cdot N_{\omega_j}(x) \tag{4.2}$$

Again, in order to obtain a system of linear relations, we define:

$$\begin{aligned} V_s &= \left(\frac{N_{\omega_i}(\tau^s(a))}{\ell^s}\right)_{1 \leq i \leq |\Omega_2|} \\ M_1 &= \left(\frac{N_{\omega_i}(\tau(\alpha_j))}{\ell}\right)_{1 \leq i \leq |\Omega_2|, 1 \leq j \leq |\mathcal{A}|} \\ M_2 &= \left(\frac{N_{\omega_i}(\tau(\omega_j)) - N_{\omega_i}(\tau(\omega_j[1])) - N_{\omega_i}(\tau(\omega_j[2]))}{\ell}\right)_{1 \leq i,j \leq |\Omega_2|} \end{aligned}$$

and equation (4.2) implies:

$$V_{s+1} = M_1 \cdot U_s + M_2 \cdot V_s$$

Again, we are interested in the limit V_∞ of V_s when s goes to infinity, and this vector is a solution of the equation: $V_\infty = M_2 \cdot V_\infty + M_1 \cdot U_\infty$. For the Keränen sequence $\mathbf{k}$, where

$$\Omega_2 = \{ab, ac, ad, ba, bc, bd, ca, cb, cd, da, db, dc\},$$

we observe that:

$$85 \cdot M_1 = \begin{pmatrix} 6 & 3 & 9 & 9 \\ 8 & 5 & 8 & 5 \\ 4 & 10 & 10 & 7 \\ 7 & 4 & 10 & 10 \\ 9 & 6 & 3 & 9 \\ 5 & 8 & 5 & 8 \\ 8 & 5 & 8 & 5 \\ 10 & 7 & 4 & 10 \\ 9 & 9 & 6 & 3 \\ 3 & 9 & 9 & 6 \\ 5 & 8 & 5 & 8 \\ 10 & 10 & 7 & 4 \end{pmatrix}$$

Because the magic string that defines the Keränen sequence begins and ends with an "a", then the digram formed by the last letter of $\tau(\alpha)$ and the first letter of $\tau(\beta)$ is more precisely $\alpha.\beta$. Thus, M_2 is in fact $1/85$ times the identity matrix. We can then compute V_∞, and we find:

Factor	ab	ac	ad	ba	bc	bd	ca	cb	cd	da	db	dc
Frequency	$\frac{9}{112}$	$\frac{13}{168}$	$\frac{31}{336}$	$\frac{31}{336}$	$\frac{9}{112}$	$\frac{13}{168}$	$\frac{13}{168}$	$\frac{31}{336}$	$\frac{9}{112}$	$\frac{9}{112}$	$\frac{13}{168}$	$\frac{31}{336}$

Here, a discrepancy is visible, with "ba" being nearly 15% more frequent than "ab". Computing the frequency of factors of size less than ℓ is not harder, and the reasoning for factors of size two can be used as-is. In fact, equation (4.2) holds even if ω is a factor of $\mathbf{z}$ of size less than ℓ. Let us define:

$$\begin{aligned} S &= \left(\frac{N_\omega\left(\tau\left(\alpha_j\right)\right)}{\ell} \right)_{1 \le j \le |\mathcal{A}|}, \\ T &= \left(\frac{N_\omega\left(\tau\left(\omega_j\right)\right) - N_\omega\left(\tau\left(\omega_j[1]\right)\right) - N_\omega\left(\tau\left(\omega_j[2]\right)\right)}{\ell} \right)_{1 \le j \le |\Omega_2|}. \end{aligned}$$

Equation (4.2) then brings:

$$\frac{N_\omega\left(\tau^{s+1}\left(a\right)\right)}{\ell^{s+1}} = S \cdot U_s + T \cdot V_s$$

And the frequency of ω in $\mathbf{z}$ is then $S \cdot U_\infty + T \cdot V_\infty$. The frequency of any word could be computed using this process recursively, but we will conclude here, as we have set up the machinery we need later on.

4.2.2 Description of the Attack

We now try to adapt the attack of §4.1 to the Dithered Hashing, assuming that there is an arbitrary dithering sequence $\mathbf{z}$. The first problem arises in the offline phase, because in order to construct the diamond structure we must evaluate the compression function, and to evaluate the compression we must choose dithering letters. A simple solution is to use the same dithering symbol for all the edges at the same depth in the tree, as shown in Figure 4.1. A word of ℓ letters is then required to build a diamond structure with 2^ℓ external nodes. We also need an additional letter to connect the "root" of the diamond structure to the message M. To summarize, in order to build a diamond structure of depth ℓ, we have to fix a word ω of size $\ell + 1$, use $\omega[i]$ as the dithering symbol at depth i, and use the last letter of ω to realize the connection to the given message. Let us denote this customized diamond structure by D_ω.

The dithering sequence makes the hash of a block dependent on its position in the whole message. Therefore, the diamond structure D_ω can be connected to its target only at certain positions, namely, at the positions where ω and $\mathbf{z}$ match. The set of positions in the message where this is possible is then given by:

$$\mathit{Range}(\mathbf{z}, \omega) = \left\{ i \in \mathbb{N} \,\middle|\, \left(\ell + 1 \le i\right) \wedge \left(\mathbf{z}[i-\ell] \ldots \mathbf{z}[i] = \omega\right) \right\}.$$

The attack is precisely described in Algorithm 4.2. The adversary tries random message blocks $m^{\searrow}$, computing $f(h_\diamond, m^{\searrow}, \omega[\ell])$, until some h_i is encountered. If $i \in \mathit{Range}(\mathbf{z}, \omega)$, then the second preimage attack may carry on. Otherwise, another block $m^{\searrow}$ needs to be found. Therefore, the goal of the adversary is to build the diamond structure with a word ω which maximizes the cardinality of $\mathit{Range}(\mathbf{z}, \omega)$.

To attain this objective, ω should be the *most frequent* factor of $\mathbf{z}$. Its frequency, also known as the min-entropy of $\mathbf{z}$, is therefore very important in computing the complexity of our attack. In the worst case, all factors of size $\ell + 1$ appear in $\mathbf{z}$ with the same frequency, and the probability that a randomly chosen factor of size $\ell + 1$ in $\mathbf{z}$ is the word ω is $1/\mathit{Fact}_{\mathbf{z}}(\ell + 1)$. However, when it is possible to actually compute

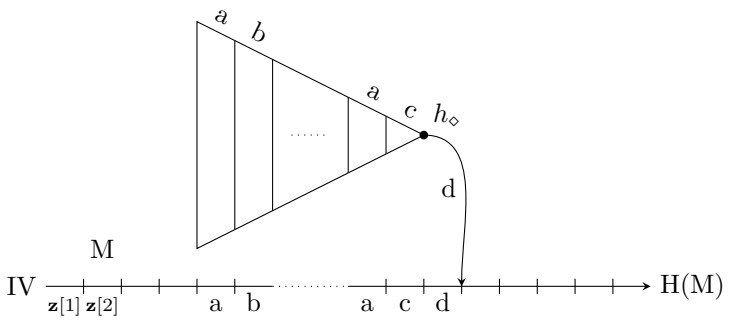


Figure 4.1: A diamond built on top of a factor of the dithering sequence, connected to the message.

Algorithm 4.2 Second preimage attack on Dithered Hashing.

1. Choose the most frequent factor ω of $\mathbf{z}$, of length $\ell + 1$.
2. Build a collision tree of depth ℓ using the first ℓ symbols of ω as the dithering symbols in all the leaf-to-root paths. Let $h_\diamond$ be the target value of the tree.
3. Connect $h_\diamond$ to some intermediate chaining value in the target message M (say h_i), by using $\omega[\ell]$ as the dithering letter. Repeat until $i \in Range(\mathbf{z}, \omega)$, where

$$Range(\mathbf{z}, \omega) = \Big\{ i \in \mathbb{N} \;\Big|\; \big(\ell + 1 \leq i\big) \wedge \big(\mathbf{z}[i - \ell] \ldots \mathbf{z}[i] = \omega\big) \Big\}.$$

   Let $m^{\searrow}$ be a message block satisfying this condition, i.e., $h_i = f(h_\diamond, m^{\searrow}, \omega[\ell])$.
4. Generate an arbitrary prefix P of size $i - \ell - 2$ blocks, and assume that it hashes to h_P, then "connect" it to the diamond. This can be done by generating random message blocks B, until $f\left(h_P, B, z[i - \ell - 1]\right) = x_j$, where x_j is a chaining value labeling a leaf of the diamond. Let $m^{\nearrow}$ denote such a block, and let T be the chain of ℓ blocks traversing the diamond from x_j to $h_\diamond$.
5. Form a message $M' = P \parallel m^{\nearrow} \parallel T \parallel m^{\searrow} \parallel m_{i+1}, \ldots, m_{2^\kappa - 1}$.

the min-entropy of $\mathbf{z}$, better results can be achieved. In any case, the cost of finding the second preimage for a given sequence $\mathbf{z}$ is

$$2^{n/2+\ell/2+2} + 2^{H_\infty(\mathbf{z},\ell+1)} \cdot 2^{n-\kappa} + 2^{n-\ell},$$

where $H_\infty(\mathbf{z}, \ell + 1)$ is the min-entropy of $\ell + 1$ words in the sequence $\mathbf{z}$. In most "good sequences" (where the probability of all factors is roughly the same) or when the computation of the exact $H_\infty(\mathbf{z}, \ell + 1)$ is infeasible, we can give an upper bound for the attack:

$$2^{n/2+\ell/2+2} + Fact_{\mathbf{z}}(\ell + 1) \cdot 2^{n-\kappa} + 2^{n-\ell}.$$

4.2.3 The Multi-Diamond Attack

So far, we only used a single diamond D_ω, built using a single factor ω of the dithering sequence. As mentioned earlier, this diamond can only be used at specific locations, corresponding to the set of locations of $\mathbf{z}$ where ω appears. We note that while the locations to connect into the message are determined by the dithering sequence, the complexity of connecting to the diamond structure depends (mostly) on the parameter ℓ, which can be chosen by the adversary. Hence, to make the attack faster, we could try to enlarge the range of our herding device at the expense of a more costly precomputation and more memory.

Let $\omega_1.\beta.\alpha$ and $\omega_2.\gamma.\alpha$ be two factors of size $\ell + 2$ of the dithering sequence both ending by a given letter α, with $\alpha, \beta, \gamma \in \mathcal{A}$ (so that ω_1 and ω_2 are made of ℓ letters). We can build two independent diamonds D_{ω_1} and D_{ω_2} using ω_1 and ω_2, respectively, to feed the dithering symbols. Assume that the root of D_{ω_1} (respectively, D_{ω_2}) is labelled by $h^1_\diamond$ (respectively, $h^2_\diamond$). Now, we could find a colliding pair (m_1, m_2) such that $f(h^1_\diamond, m_1, \beta) = f(h^2_\diamond, m_2, \gamma)$. Let us denote by $h_{\diamond\diamond}$ the resulting chaining value. Figure 4.2 illustrates this. This last node can be connected to the message using α as the dither symbol. We have "herded" together two diamonds with two different dithering words, and the resulting "multi-diamond" is more useful than any of the two diamonds separately. This claim is justified by the fact that the range of the new multi-diamond is the union of the two ranges of the two separate diamonds.

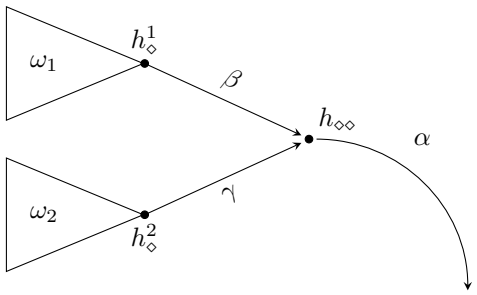


Figure 4.2: A "Multi-diamond" with 2 words.

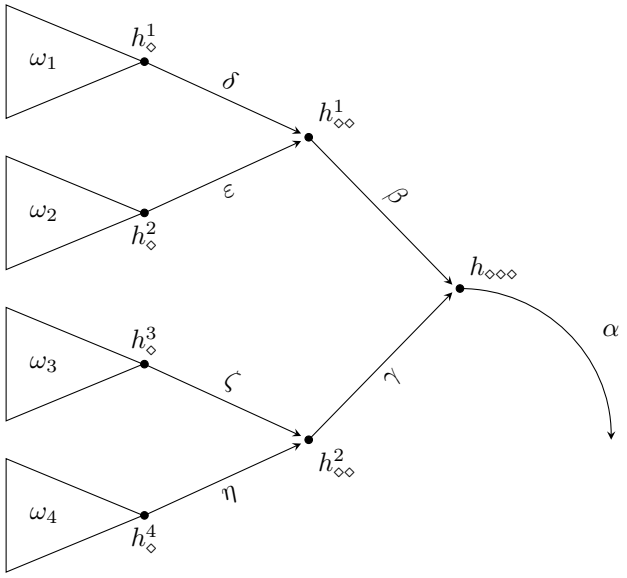


Figure 4.3: A "Multi-diamond" with 4 words.

This technique can be used twice, as exemplified by Figure 4.3 to provide an even bigger range, as long as there is a set of four factors of $\mathbf{z}$ of size $\ell + 3$ that can be written:

$$\{\omega_1.\delta.\beta.\alpha, \quad \omega_2.\varepsilon.\beta.\alpha, \quad \omega_3.\zeta.\gamma.\alpha, \quad \omega_4.\eta.\gamma.\alpha\},$$

where $|\omega_i| = \ell$ and $\alpha, \beta, \gamma, \delta, \varepsilon, \zeta, \eta \in \mathcal{A}$. A total number of 3 colliding pairs are needed to assemble the 4 diamonds together into this new multi-diamond.

Let us generalize this idea. We say that a set of 2^k words is *suffix-friendly* if all the words end by the same letter, and if after chopping the last letter of each word, the set can be partitioned into two suffix-friendly sets of size 2^{k-1} each. A single word is always suffix-friendly, and thus the definition is well-founded. Of course, a set of 2^k words can be suffix-friendly only if all the words are of length greater than k. If the set of factors of size $\ell + k + 1$ of $\mathbf{z}$ contains a suffix-friendly subset of 2^k words, then the technique described here can be recursively applied k times.

An apparently tricky problem is to determine the biggest k such that a given set of words, say Ω, contains a suffix-friendly subset of size 2^k. This is fortunately doable in time polynomial in the sizes of Ω and $\mathcal{A}$.

Also, given a word ω, we define the *restriction* of a multi-diamond herding tree to ω by removing nodes from the original until all the paths between the leaves and the root are labelled by ω. For instance, restricting the multi-diamond of Figure 4.3 to $\omega_1.\delta.\beta.\alpha$ means keeping only the first sub-diamond and the path $h^1_\diamond \to h^1_{\diamond\diamond} \to h_{\diamond\diamond\diamond}$.

Now, assume that the set of factors of size $\ell+k+1$ of $\mathbf{z}$ contains a suffix-friendly subset of size 2^k denoted by $\Omega = \{\omega_1, \ldots, \omega_{2^k}\}$. The multi-diamond formed by herding together the 2^k diamonds corresponding to the ω_i's can be used in place of any of them, as mentioned above. Therefore, its "frequency" is the sum of the frequency of the ω_i. However, once connected to the message, only its restriction to the $\ell + k + 1$ letter of $\mathbf{z}$ before the connection can be used. This restriction is a diamond with 2^ℓ leaves (followed by a "useless" path of k nodes).

The cost of building a 2^k-multi-diamond is 2^k the time of building a diamond of size ℓ plus the cost of finding $2^k - 1$ collisions. The cost of connecting the prefix to the multi-diamond is still $2^{n-\ell}$ (this step is the

same as in our original attack). Lastly, the cost of connecting the multi-diamond to the message depends on the frequency of the factors chosen to build it, which ought to be optimized according to the actual dithering sequence.

4.2.4 Cryptanalysis of Keränen-DMD

The cost of the single-diamond attack against Keränen-DMD depends on the properties of the sequence $\mathbf{k}$ that have been outlined in §4.2.1. Let us emphasize again that since it has a very regular structure, $\mathbf{k}$ has an unusually low complexity, and despite being strongly repetition-free, the sequence offers an extremely weak security level against our attack. Following the ideas of §4.2.1, the min-entropy of $\mathbf{k}$ for words of size $\ell \leq 85$ can be computed precisely: for $29 \leq \ell \leq 85$, the frequency of the most frequent factor of size $\ell+1$ is $1/(4 \cdot 85) = 2^{-8.4}$ (if all the factors of length, say, 50 were equally frequent, this would have been $1/732 = 2^{-9.5}$). We thus have $H_\infty(\mathbf{z}, \ell+1) = 8.4$, and the cost of our attack on Keränen-DMD, assuming that $29 \leq \ell \leq 85$, is:

$$2^{n/2+\ell/2+2} + 2^{n-\kappa+8.4} + 2^{n-\ell}.$$

If n is smaller than $3\kappa - 8.4$, the optimal value of ℓ is reached by fixing $\ell = (n-4)/3$. For n in the same order as 3κ, all the terms are about the same (for $n > 3\kappa$, the first term can be ignored). Hence, to obtain the best overall complexity (or to optimize the online complexity) we need to fix ℓ such that $2^{n-\kappa+8.4} = 2^{n-\ell}$, *i.e.*, $\ell = \kappa - 8.4$. For example, for $\kappa = 55$ the optimal value of ℓ is 46.6. The online running time (which is the majority of the cost for $n > 3\kappa$) is in this case $2^{n-46.6}$ which is much smaller than 2^n in spite of the use of dithering. For larger values of ℓ, *i.e.*, $85 \leq \ell < 128$, we empirically measured the min-entropy to be $H_\infty(\mathbf{k}, \ell+1) = 9.8$ so that $\ell = \kappa - 9.8$ can be used when $n \approx 3\kappa$.

We also successfully applied the multi-diamond attack to Keränen-DMD. We determined the smallest ℓ such that the set of factors of size ℓ of the Keränen sequence $\mathbf{k}$ contains a 2^k suffix-friendly set, for various values of k:

k	$\min \ell$	$Fact_{\mathbf{k}}(\ell)$
4	4	88
5	6	188
6	27	540
7	109	1572
8	194	4256

From this table we conclude that our choice of k we will most likely be 6, since choosing 7 would require us to choose $\ell \geq 109$, which is going to unbalance the cost of the two online connections steps. Amongst all the possible suffix-friendly sets of size 2^6 found in the factors of size about 50 of $\mathbf{k}$, we chose one having a high frequency using a greedy algorithm making use of the ideas exposed in §4.2.1. We note that checking whether this yields *optimal* multi-diamonds is out of the scope of this work. In any case, we found that the frequency of our multi-diamond, shown in Figure 4.4, is $2^{-3.97}$.

If n is sufficiently large (for instance, $n = 256$), the offline part of the attack is still of negligible cost. Then, the minimal online complexity is obtained when $2^{n-\kappa+3.97} = 2^{n-\ell}$, *i.e.*, $\ell = \kappa - 3.97$. The complexity of the attack is then roughly $2 \cdot 2^{n-\kappa+4}$ for sufficiently large values of n. This represents a speed-up of about 21 compared to the single-diamond attack.

4.2.5 Cryptanalysis of DMD-CP

We now apply our attack to Rivest's concrete proposal. We first need to evaluate the complexity of its dithering sequence. Recall from §3.4.2 that it is based on the Keränen sequence, but that we move on to the next symbol of the sequence only when a 13-bit counter overflows (we say that it results in the *dilution* of $\mathbf{k}$ with a 13-bit counter). The original motivation was to reduce the cost of the dithering, but it has the unintentional effect of increasing the resulting sequence complexity. It is possible to study this dilution operation generically, and to see to which extent it makes our attack more difficult.

Lemma 4.3. *Let $\mathbf{z}$ be an arbitrary sequence over $\mathcal{A}$, and let $\mathbf{d}$ denote the sequence obtained by diluting $\mathbf{z}$ with a counter over i bits. Then for every ℓ not equal to 1 modulo 2^i, we have:*

$$\begin{aligned} Fact_{\mathbf{d}}(\ell) = \big(2^i - (\ell \mod 2^i) + 1\big) \cdot Fact_z\big(\lceil \ell \cdot 2^{-i} \rceil\big) \\ + \big((\ell \mod 2^i) - 1\big) \cdot Fact_{\mathbf{z}}\big(\lceil (\ell-1) \cdot 2^{-i} \rceil + 1\big) \end{aligned}$$

Proof. The counter over i bits splits the diluted sequence $\mathbf{c}$ into chunks of size 2^i (a new chunk begins when the counter reaches 0). In a chunk, the letter from $\mathbf{z}$ does not change, and only the counter varies. To obtain the number of factors of size ℓ, let us slide a window of size ℓ over $\mathbf{d}$. This window overlaps at

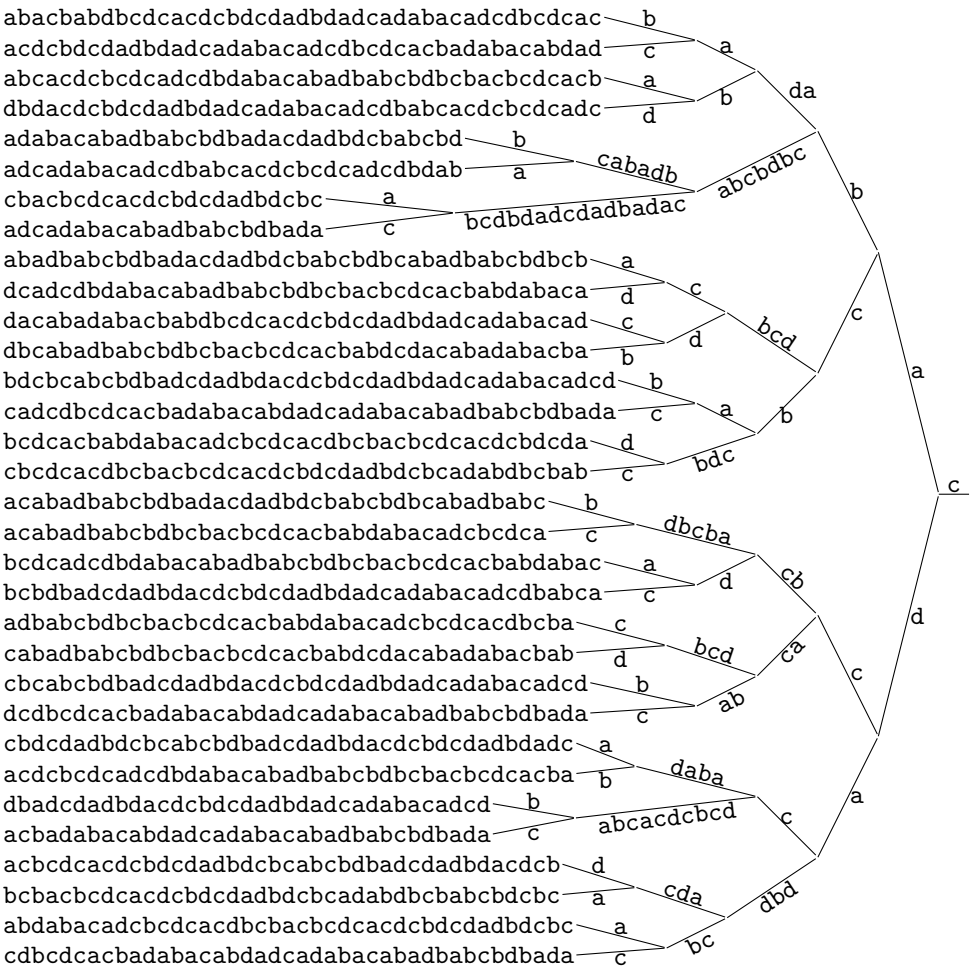


Figure 4.4: A suffix-friendly set of 32 factors of size 50 from the Keränen sequence.

least $\lceil \ell \cdot 2^{-i} \rceil$ chunks (when the beginning of the window is aligned at the beginning of a chunk), and at most $\lceil (l-1) \cdot 2^{-i} \rceil + 1$ chunks (when the window begins just before a chunk boundary). These two numbers are equal if and only if $\ell \equiv 1 \mod 2^i$. When this case is avoided, then these two numbers are consecutive integers.

This means that by sliding this window of size ℓ over $\mathbf{d}$ we observe only factors of $\mathbf{z}$ of size $\lceil \ell \cdot 2^{-i} \rceil$ and $\lceil \ell \cdot 2^{-i} \rceil + 1$. Given a factor of size $\lceil \ell \cdot 2^{-i} \rceil$ of $\mathbf{z}$, there are $\left(2^i - (\ell \mod 2^i) + 1\right)$ positions of a window of size ℓ that allow us to observe this factor with different values of the counter. Similarly, there are $\left((\ell \mod 2^i) - 1\right)$ positions of the window that contain a given factor of $\mathbf{z}$ of size $\lceil \ell \cdot 2^{-i} \rceil + 1$. □

By taking $2 \leq \ell \leq 2^i$, we have that $\lceil \ell \cdot 2^{-i} \rceil = 1$. Therefore, only the number of factors of length 1 and 2 of $\mathbf{z}$ come into play. The formula can be further simplified into:

$$Fact_{\mathbf{d}}(\ell) = \ell \cdot \Big(Fact_{\mathbf{z}}(2) - Fact_{\mathbf{z}}(1) \Big) + (2^i + 1) \cdot Fact_{\mathbf{z}}(1) - Fact_2(\mathbf{z}).$$

For the Keränen sequence with $i = 13$, this gives: $Fact_{\mathbf{d}}(\ell) = 8 \cdot \ell + 32760$. Diluting over i bits makes the complexity 2^i times higher, but it does not change its asymptotic expression: it is still linear in ℓ, even though the constant term is bigger due to the counter. The cost of the attack is therefore:

$$2^{\frac{n}{2} + \frac{\ell}{2} + 2} + (8 \cdot \ell + 32760) \cdot 2^{n-\kappa} + 2^{n-\ell}.$$

At the same time, for any $\ell \leq 2^i$, the most frequent factor of $\mathbf{d}$ is $(\alpha, 0), (\alpha, 1), \ldots, (\alpha, \ell - 1)$ when α is the most frequent letter of the Keränen sequence. However, as shown in section 4.2.1.1, all the letters have the same frequency, so most frequent factor of the diluted Keränen sequence $\mathbf{d}$ has a frequency of 2^{-15}. Hence, the cost of the above attack is:

$$2^{n/2 + \ell/2 + 2} + 2^{n-\kappa+15} + 2^{n-\ell}.$$

This is an example where the most frequent factor has a frequency which is very close to the inverse of the number of factors (2^{-15} vs. $1/(8 \cdot \ell + 32760)$). In this specific case it may seem that the gain of using the most frequent element is small.

As before, if n is greater than 3κ (in this specific case $n \geq 3\kappa - 41$), the optimal value of ℓ is $\kappa - 15$, and the complexity of the attack is then approximately $2 \cdot 2^{n-\kappa+15}$. For settings corresponding to SHA-1, a second preimage can be found in expected time of 2^{120} (for $40 < \ell < 78$).

4.2.6 Countermeasures

We just observed that the presence of a counter increases the complexity of the attack. If we simply use a counter over i bits as the dithering sequence, the number of factors of size ℓ becomes $Fact(\ell) = 2^i$ (as long as $i \leq \ell$). The complexity of the attack would then become:

$$2^{n/2+\ell/2+2} + 2^{n-\kappa+i} + 2^{n-\ell}.$$

By taking $i = \kappa$, we obtain a scheme which is resistant to our attack. This is essentially the choice made by the designers of Haifa, but such a dithering sequence consumes κ bits of bandwidth. Using a counter (*i.e.*, a big alphabet) is a simple way to obtain a dithering sequence of high complexity. Another, somewhat orthogonal, possibility to improve the resistance of Rivest's dithered hashing to our attack is to use a dithering sequence of high complexity over a *small* alphabet (to preserve bandwidth).

Abelian Square-Free Sequences of Exponential Complexity. It is possible to construct an infinite abelian square-free sequence of exponential complexity, although we do not know how to do it without slightly enlarging the alphabet.

We start with the abelian square-free Keraïen sequence $\mathbf{k}$ over $\{a, b, c, d\}$, and with another sequence $\mathbf{u}$ over $\{0, 1\}$ that has an exponential complexity. For example, such a sequence can be built by concatenating the binary encoding of all the consecutive integers. Then we can create a sequence $\tilde{\mathbf{z}}$ over the union alphabet $\mathcal{A} = \{a, b, c, d, 0, 1\}$ by interleaving $\mathbf{k}$ and $\mathbf{u}$:

$$\tilde{\mathbf{z}} = \mathbf{k}[1].\mathbf{u}[1].\mathbf{k}[2].\mathbf{u}[2]\ldots.$$

The resulting shuffled sequence inherits both properties: it is still abelian square-free, and has a complexity of order $\Omega\left(2^{\ell/2}\right)$. Using this improved sequence, with $\ell = 2\kappa/3$, the total cost of the online attack is about $2^{n-2\kappa/3}$ (assuming $n > 8\kappa/3$). Even with this exponentially complex dithering sequence, our attack is still more efficient than brute-force in finding second preimages. Although it may be possible to find square-free sequences with even higher complexity, it is probably very difficult to achieve optimal protection, and the generation of the dithering sequences is likely to become more and more complex.

Pseudorandom Sequences. Another possible way to improve the resistance of Rivest's construction against our attack is to use a pseudo random sequence over a small alphabet. Even though it may not be repetition-free, its complexity is almost maximal. Suppose that the alphabet has size $\left|\mathcal{A}\right| = 2^i$. Then the expected number of ℓ-letter factors in a pseudo random word of size 2^κ is lower-bounded by: $2^{i\cdot\ell} \cdot \left(1 - \exp\left\{-2^{\kappa - i\cdot\ell}\right\}\right)$ (refer to [JLS04], theorem 2, for a proof of this claim). The total optimal cost of the online attack is then at least $2^{n-\kappa/(i+1)+2}$ and is obtained with $\ell = \kappa/(i+1)$. With 8-bit pseudorandom dithering symbols and $\kappa = 55$, the complexity of our attack is about 2^{n-5}, which still offers a small advantage over exhaustive search.

To conclude, we note that we show how to perform some attacks on dithering sequences over small alphabet in the precomputation model in §6.3 (*i.e.*, when a one-time precomputation of complexity 2^n subsequently allows to find arbitrary second preimages faster than exhaustive search).

4.3 Application to Shoup's UOWHF

We now adapt the attack of §4.1 to Shoup's UOWHF. To the best of our knowledge, no previous attack on this construction was known. The complexity of the attack matches the lower bound given in Shoup's security proof (cf §3.4.3), thus showing that both the attack and the proof are optimal.

We first give an alternate definition of the dithering sequence $\mathbf{z}$ used in Shoup's UOWHF. In fact, the alphabet over which the sequence $\mathbf{z}[i] = \nu_2(i)$ is built is not finite, as it is the whole $\mathbb{N}$. In any case, we define:

$$u_i = \begin{cases} 0 & \text{if } i = 1, \\ u_{i-1}.(i-1).u_{i-1} & \text{otherwise.} \end{cases}$$

As an example, we have $u_4 = 010201030102010$. The following facts about $\mathbf{z}$ are easy to establish:

i) $|u_i| = 2^i - 1$

ii) The number of occurrences of u_i in u_j (with $i < j$) is 2^{j-i}.

iii) The frequency of u_i in the (infinite) sequence $\mathbf{z}$ is 2^{-i}.

iv) The frequency of a factor is the frequency of its highest letter.

v) Any factor of $\mathbf{z}$ of size ℓ contains a letter greater or equal to $\lfloor \log_2(\ell) \rfloor$.

Let us consider a factor of size ℓ of $\mathbf{z}$. It follows from the previous considerations that its frequency is upper-bounded by $2^{-\lfloor \log_2(\ell) \rfloor - 1}$, and that this bound is reached by a prefix of size ℓ of $\mathbf{z}$. The frequency is lower-bounded by the nicer expression $1/(2 \cdot \ell)$.

Our attack can be applied against the eSec property of H as described above. Choose at random a (long) target message M. Once the key is chosen at random, build a collision tree using a prefix of $\mathbf{z}$ of size ℓ, and continue as described in §4.2.2. The cost of the attack is then upper-bounded by:

$$T = 2^{n/2+\ell/2+2} + 2 \cdot \ell \cdot 2^{n-\kappa} + 2^{n-\ell}.$$

The attack breaks the target collision resistance with a constant success probability (of about 63%). Therefore, with Shoup's security reduction, one can construct an adversary against f with running time T and probability of success $0.63/2^\kappa$. If f is a black box, the best attack against the eSec property of f is exhaustive search. Thus, the best adversary in time T against f has a success probability of $T/2^n$. When $n \geq 3\kappa$, $T \simeq (2\kappa + 2) \cdot 2^{n-\kappa}$ (with $\ell = \kappa - 1$), and thus the best adversary running in time T has success probability $\mathcal{O}(\kappa/2^\kappa)$ while the success probability of our attack is $0.63/2^\kappa$. This implies that there is no attack better than ours by a factor greater than $\mathcal{O}(\kappa)$ or, in other words, there is only a factor $\mathcal{O}(\kappa)$ between Shoup's security proof and our attack.

We note that in this case, there is a very large gap between the frequency of the most frequent factor and the upper-bound provided by the inverse of the number of factors. Indeed, it can be seen that:

$$Fact_{u_i}(\ell) = \begin{cases} 0 & \text{if } |u_i| < \ell \\ 2^i - \ell & \text{if } |u_{i-1}| < \ell \leq |u_i| \\ \ell + Fact_{u_{i-1}}(\ell) & \text{if } |u_{i-1}| \geq \ell \end{cases}$$

And the expression of the number of factors follows:

$$Fact_{u_\kappa}(\ell) = 2^{\lceil \log_2(\ell+1) \rceil} + \left(\kappa - \lceil \log_2(\ell+1) \rceil - 1\right) \cdot \ell$$

Hence, if all of them would appear with the same probability, the time complexity of the attack would have been

$$T = 2^{\frac{n}{2}+\frac{\ell}{2}+2} + \left(2^{\lceil \log_2(\ell+1) \rceil} + \left(\kappa - \lceil \log_2(\ell+1) \rceil - 1\right) \cdot \ell\right) \cdot 2^{n-\kappa} + 2^{n-\ell},$$

which is roughly κ times bigger than the previous expression.

4.3.1 Application of the Multi-Diamonds Attack

To apply the multi-diamond attack of §4.2.3, we need to identify a big enough suffix-friendly subset of the factors of $\mathbf{z}$ of a given size, and to compute its frequency.

We choose to have end diamonds of size $\ell = 2^{2^i-1}$. Let us keep in mind that ℓ and κ must generally be of the same order to achieve the optimal attack complexity, which suggest that i should be close to $\log_2 \log_2 \kappa$. Now, we need to identify a suffix-friendly set of factors of $\mathbf{z}$ in order to build a multi-diamond. In fact, we focus on the factors that have u_i as a suffix. It is straightforward to check that they form a suffix-friendly set. It now remains to estimate its size and its frequency.

Lemma 4.4. *let Ω_j be the set of words ω of size $\ell = 2^{2^i-1}$ such that $\omega.u_i$ is a factor of u_j. Then:*

i) If $\kappa \geq 2^i$, then $|\Omega_\kappa| = \left(\kappa - 2^i + 1\right) \cdot 2^{2^i-i-1}$

ii) There are 2^{2^i-i-1} (distinct) words in Ω_κ whose frequency is 2^{-j} (with $2^i \leq j \leq \kappa$).

Proof. We first evaluate the size of Ω, and for this we define $f_i(\kappa)$, the number of factors of u_κ that can be written $\omega.u_i$, with $|\omega| = 2^{2^i-1}$. We find:

$$|\Omega_\kappa| = \begin{cases} 0 & \text{if } 2^\kappa < 2^{2^i-1} + 2^i \\ |\Omega_{\kappa-1}| + 2^{2^i-i-1} & \text{if } 2^\kappa \geq 2^{2^i-1} + 2^i \end{cases} \tag{4.3}$$

The first case of this equality is rather obvious. The second case stems from the following observation: let x be a factor of u_j, for some j. Then either x is a factor of u_{j-1}, or u contains the letter "$j-1$" (both

Function (n, κ)		MD5 (128,55)	SHA-1 (160,55)	SHA-256 (256,118)	SHA-512 (512,118)
Single-Diamond	Offline:	2^{91}	2^{107}	2^{189}	2^{317}
	Online:	2^{80}	2^{112}	$2^{145.7}$	$2^{401.7}$
	Memory:	2^{50}	2^{50}	$2^{111.3}$	$2^{111.3}$
	Patch:	51	51	114	114
Multi-Diamond	Offline:	$2^{96.5}$	$2^{112.5}$	$2^{194.5}$	$2^{322.5}$
	Online:	$2^{76.9}$	$2^{108.9}$	2^{142}	2^{398}
	Memory:	2^{59}	2^{59}	2^{123}	2^{123}
	Patch:	60	60	124	124

Table 4.3: Comparison of the Time Complexity of Our Attacks on Shoup's UOWHF

cases are mutually exclusive). Thus, we only need to count the numbers of factors of Ω_κ containing the letter "$\kappa - 1$" to write a recurrence relation.

If $2^\kappa \geq 2^{2^i-1} + 2^i$, then u_i appears $2^{\kappa-i}$ times in u_κ, at indices that are multiples of 2^i. The unique occurrence of letter "$\kappa - 1$" in u_κ is at index $2^{\kappa-1} - 1$. Thus, elements of Ω_κ containing the letter "$\kappa - 1$" are present in u_κ at indices $2^{\kappa-1} - 2^{2^i-1} + \alpha \cdot 2^i$, with $0 \leq \alpha < 2^{2^i-i-1}$. Therefore, there are exactly 2^{2^i-i-1} distinct elements of Ω_κ containing "$\kappa - 1$" in u_κ (they are necessarily distinct because they all contain "$\kappa - 1$" only once and at different locations).

Now that (4.3) is established, we can unfold the recurrence relation. We note that we have for $i \geq 1$, $\left\lceil \log_2 \left(2^{2^i-1} + 2^i\right) \right\rceil = 2^i$, and thus we obtain (assuming that $\kappa \geq 2^i$):

$$|\Omega_\kappa| = \left(\kappa - 2^i + 1\right) \cdot 2^{2^i-i-1}$$

Also, for $2^i \leq j \leq \kappa$, Ω_κ contains precisely 2^{2^i-i-1} words whose greatest letter is "$j - 1$", and thus whose frequency in $\mathbf{z}$ is 2^{-j}. □

By just selecting the factors of Ω_κ of the highest frequency, we would herd together

$$2^{2^i-i-1} = \ell / \left(1 + \log_2 \ell\right)$$

diamonds, each one being of frequency $1/(2\ell)$. The frequency of the resulting multi-diamond then becomes $1/\left(2 + 2\log_2 \ell\right)$. The cost of the multi-diamond attack is thus roughly:

$$\frac{\ell}{1 + \log_2 \ell} \cdot \left(2^{(n+\ell)/2+2} + 2^{\frac{n}{2}}\right) + \left(1 + \log_2 \ell\right) \cdot 2^{n-\kappa+1} + 2^{n-\ell}.$$

If $n \gg 3\kappa$, the preprocessing will be negligible compared to the online time, and the cost of the attack is $\mathcal{O}\left(\log \kappa \cdot 2^{n-\kappa}\right)$. Therefore, with the same reasoning as before, we can show that there is a factor $\mathcal{O}\left(\log \kappa\right)$ between Shoup's security proof and our attack. Note that, depending on the parameters, this improved version of the attack may be worse than the basic version.

We outline the complexities out our attacks (the regular and the multi-diamond ones) against MD5, SHA-1, SHA-256, and SHA-512 in Table 4.3.

Chapter 5

Other Generic Attacks on Other Constructions

In this chapter we give more generic attacks on less standard modes of operations. Our results culminate with a generic second preimage attack on Merkle-Damgård-Again. This work has led to a joint publication with Elena Andreeva, Orr Dunkelman and John Kelsey at SAC 2009 [ABDK09].

Iterated-Concatenated-Expanded (ICE) hash functions generalize the Merkle-Damgård mode of operation in a natural way, by allowing an arbitrary number of permuted copies of the message to be hashed sequentially. The resulting construction is obviously non-streamable. the Merkle-Damgård-Again construction and the Zipper Hash are easily seen to be particular cases of this more general framework. A very general result of Hoch and Shamir [HS06], generalizing a preliminary result of Nandi and Stinson [NS07], states that any ICE hash function is susceptible to a generalization of Joux's multicollision attack, with essentially the same complexity.

In this chapter we aim at breaking some ICE hash functions using either a herding attack or a second preimage attack. In §5.1 we build the main tool that helps us achieve these goals, namely the ability to perform the herding attack on concatenated hashes. We then use this tool to find herding attacks on the Zipper Hash and on Merkle-Damgård-Again in §5.2. Building on this result we extend the second preimage attack of chapter 4 to Merkle-Damgård-Again in §5.3, combining Joux's multicollisions, Kelsey and Scheier's expandable messages and Kelsey and Kohno's diamond structure in a single attack, arguably *the* most sophisticated generic attack on hash functions so far. We conclude this chapter by presenting in §5.4 the *Trojan Message Attack*, a new kind of generic attack. Like Joux's multicollision attack, the trojan message attack only requires a few collisions, and is therefore practical on some legacy hash functions such as MD5. We did a public demonstration of this attack at the rump session of CRYPTO 2009.

5.1 Herding Concatenated Hashes

We start by showing how to adapt the herding attack to concatenated hashes. The main idea behind the new attack is to construct *multi-pipe diamond structures*, which can be done "on top" of multicollisions. We recall that a multicollision on $(k-1)$ pipes can be used to construct a collision on k pipes. In the same vein, we succeed in herding k pipes by building a k-pipe diamond using a $(k-1)$-pipe diamond and a $(k-1)$-pipe multicollision.

Assume that the adversary succeeded in herding $k-1$ pipes. Then, she faces the problem of herding the last pipe. Now, if the adversary tries to connect to a diamond structure in the k-th pipe with a random message block, she is very likely to lose the control over the previous pipes. However, if she uses a "block" which is part of a multicollision on the first $k-1$ pipes, she still maintains the control over the previous pipes, while offering enough freedom for herding the last pipe. The attack is described recursively, and the recurrence stops when there is only a single hash function left, in which case it is sufficient to apply the "normal" herding attack.

5.1.1 Precomputation Phase

In the precomputation phase, the adversary starts with the $(k-1)$-diamond which is already known and aims at building the k-diamond that would allow herding the k pipes. Figure 5.1 depicts the process for $k=2$.

Randomization Step. Given the concatenated chaining values, the adversary constructs a $2^{n-\ell}$-multicollision on the first $k-1$ pipes. Let the resulting chaining value be $\left(h^1, h^2, \ldots, h^{k-1}\right)$. This multicollision allows to randomize the chaining value in the last pipe while keeping the first $k-1$ pipes under full control.

Actual Diamond Construction. The adversary picks at random 2^ℓ values for $D_k = \{h_i^k\}$. Then, she generates a set of further $2^{n/2}$-multicollisions on the first $k-1$ pipes, starting from the intermediate val-

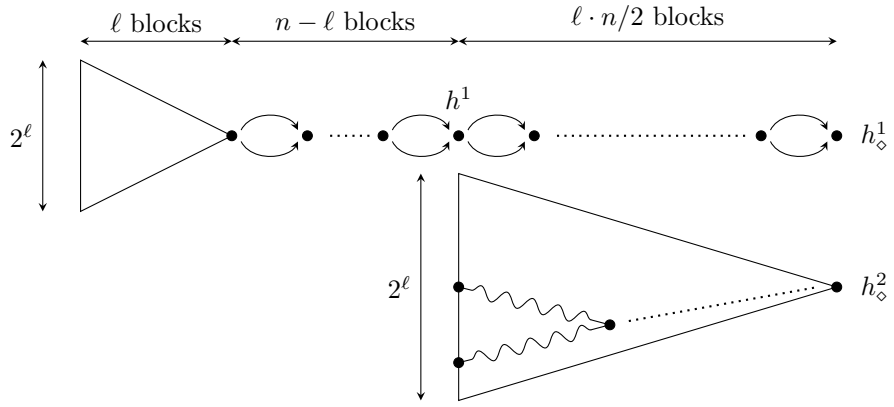


Figure 5.1: Diamond Structure on two pipes

ues $(h^1, h^2, \ldots, h^{k-1})$. The diamond in the last pipe will be built on top of this multicollision. For each possible message in the multicollision, and each starting point $(h_1^k, h_2^k, \ldots, h_{2^\ell}^k)$, the adversary computes the new chaining values, expecting to reach enough collisions, such that for any h_i^k, there exists a "message" $m_{\alpha(i)}$ (*i.e.*, a sequence of message blocks in the multicollision) such that

$$\left|\left\{{f_k}^*\left(h_i^k, m_{\alpha(i)}\right) : i \in \left\{1, \ldots, 2^\ell\right\}\right\}\right| = 2^{\ell-1}.$$

After this step, the same process is repeated as in the "normal" herding attack, until a fully converging tree has been built.

Running Time. The running time is dominated by the generation of the diamond structure in the last pipe. First, we need to generate a $2^{n-\ell+n\ell/2}$-multicollision on $k-1$ pipes, which requires:

$$\left(n - \ell + \frac{n\ell}{2}\right) \cdot (k-1) \cdot \left(\frac{n}{2}\right)^{k-2} \cdot 2^{n/2}$$

compression function calls. Then, we need to "hash" 2^ℓ values under $2^{\frac{n-\ell+1}{2}}$ message sequences (for the last layer of the diamond structure). While at a first glance it may seem that we need a very long time for each message sequence, it can be done efficiently if we take into consideration the fact that there is no need to recompute all the chaining values only if the last block was changed. Hence, the actual time required to construct the diamond structure is $2 \cdot 2^{n/2+\ell/2+2}$ (twice the time needed for a classic diamond structure). In total, the complexity of the preprocessing is:

$$\left(n - \ell + \frac{n\ell}{2}\right) \cdot (k-1) \cdot \left(\frac{n}{2}\right)^{k-2} \cdot 2^{n/2} + (2 \cdot k - 1) \cdot 2^{\frac{n+\ell}{2}+2}.$$

One may ask what is the reason for the randomization step. As demonstrated in the online phase of the attack, the need arises from the fact that herding the values in the first $k-1$ pipes fixes the value in the k-th pipe. Hence, we need enough "freedom" to randomize this chaining value, without affecting the already solved pipes.

5.1.2 Online Phase

The online phase of the herding attack is illustrated when $k = 2$ in Figure 5.2. Given a precomputed k-diamond structure, it is possible to apply the herding attack to k concatenated hash functions. The adversary is given a prefix P, and tries various message blocks $m_1^\nearrow$ until ${f_1}^*\left(IV_1, P \parallel m_1^\nearrow\right)$ gives one of the 2^ℓ chaining values in D_1 (the diamond structure on the first pipe). Then, the adversary traverses the first diamond structure to its root, finding the first part of the suffix S_1. Note that so far all computations have been done in the first pipe. At this point, the adversary computes ${f_2}^*(IV_2, P \parallel m_1^\nearrow \parallel S_1 \parallel m_2^\nearrow)$ for all the $2^{n-\ell}$ messages $m_2^\nearrow$ of the multicollision in the randomization path, until one of them hits one of the 2^ℓ values in D_2 (the diamond on the second pipe). The adversary can then use the paths inside this second diamond to reach $h_\diamond^2$ in the second pipe, while the first pipe reaches $h_\diamond^1$. This process can start again (with a randomization part, and traversing the diamond structure) until all k pipes were herded correctly.

We note that once a pipe is herded, there is no longer a need to evaluate it (as the multicollision predicts its value), and then it is possible to start analyzing the next pipe. In each new pipe, we need to evaluate $2^{n-\ell}$ "messages" (for all pipes but the first one, these messages are multicollisions on the previous pipes),

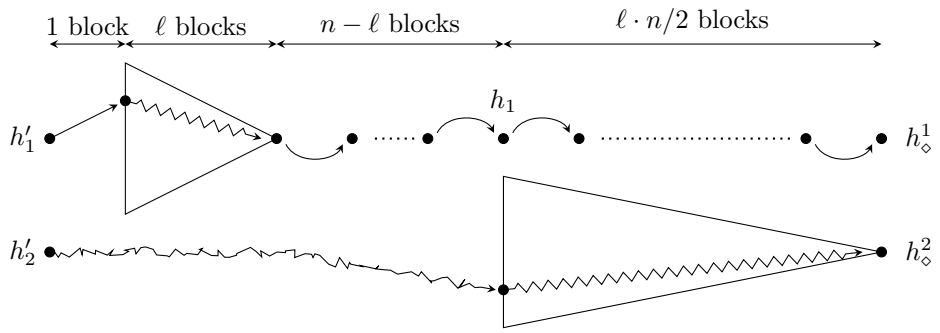


Figure 5.2: The Online Phase of the Herding Attack for $k = 2$

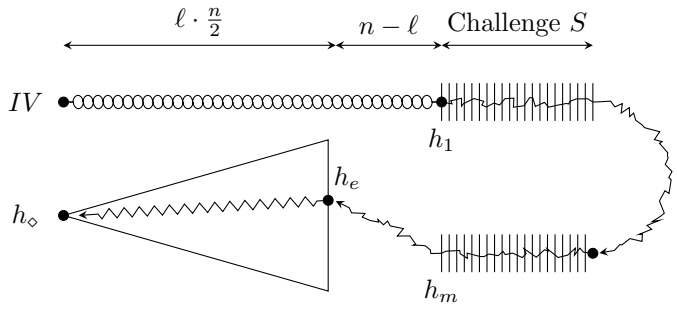


Figure 5.3: Herding the Zipper Hash

each takes on average (in an efficient implementation) two compression function calls (besides the first layer). Hence, the online time complexity of the attack is

$$2^{n-\ell} \cdot [1 + 2 \cdot (k-1)] = (2k-1) \cdot 2^{n-\ell}$$

compression function calls, essentially $2k$ times more than the normal herding attack.

5.2 Herding Some Non-Streamable Modes of Operations

In this section, we adapt the herding attack from the previous section to the Zipper Hash and to Merkle-Damgård-Again. This is not very difficult once these two construction are seen as the concatenation of two hash functions (where the IV in the second pipe is not fixed a priori).

5.2.1 Herding the Zipper Hash Function

We begin by observing that the "regular" herding attack is not feasible on the Zipper Hash, because the last message block going into the compression function is the first message block of the challenge. It follows that an adversary capable of doing the herding attack could be used to invert the compression function. This makes the Zipper Hash *provably resistant to the Herding attack*.

We therefore consider a variant of the herding attack where the challenge is placed at the end: the adversary commits to a hash value $h_\diamond$, then she is challenged with a suffix S and has to produce a prefix P such that $\mathcal{ZH}(P \,\|\, S) = h_\diamond$. If the hash function were a Random Oracle, this would obviously require 2^n queries.

Because each message block enters the hashing process twice, choosing a message block in the second pass may change not only the chaining value going *out* the second pass but also the chaining value going *in* the second pass. Choices of the message intended to affect the second pass must thus be done in a way that does not randomize the result of the first pass. This can be done using the techniques developed for the concatenation of two hash functions. The attack is illustrated by Figure 5.3.

5.2.1.1 Offline Phase

1. Starting from the IV, build a $2^{n\ell/2+n-\ell}$-multicollision that yields a chaining value h_1.

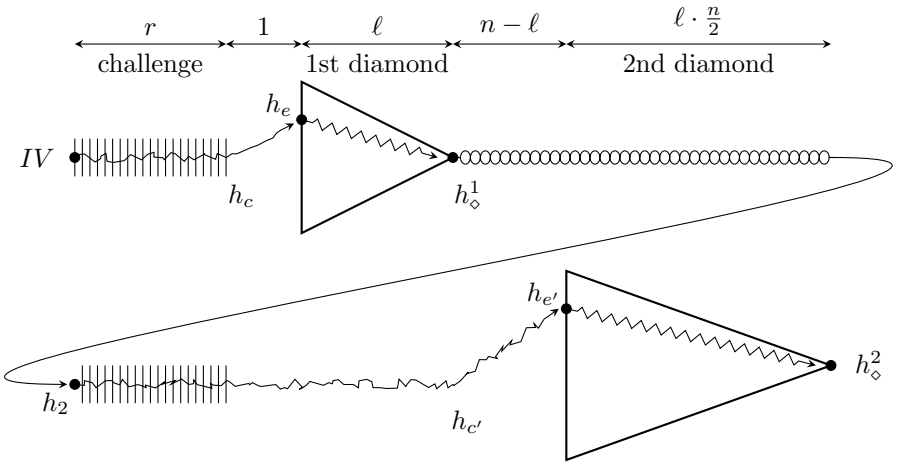


Figure 5.4: Herding Merkle-Damgård-Again

2. Build a diamond structure on top of the *reversed* multicollision (*i.e.*, where the order of colliding messages in the multicollision is reversed). The chaining value at the root of the second diamond is $h_\diamond$.
3. Commit to $h_\diamond$.

5.2.1.2 Online Phase

1. Given a challenge suffix S, compute the chaining value after the two copies of the challenge:

$$h_m = {f_2}^* \left({f_1}^* \left(h_1, S \right), \widetilde{S} \right).$$

2. From h_m, find a connecting path in the part of the (reversed) multicollision that is just before the diamond (in the second run) yielding a chaining value h_e that belongs to the diamond structure ($h_e \in D_1$). Then find a path inside the (reversed) diamond structure towards the committed hash $h_\diamond$.

We note that the fact that two different compression functions f_1 and f_2 are used in the two passes has no impact on our results, as the attack technique is independent of the actual functions used. The precomputation phase takes $2 \cdot 2^{(n+\ell)/2+2} + (n - \ell + \frac{n\ell}{2}) \cdot 2^{n/2}$ compression function evaluations, and the online computation takes $2 \cdot 2^{n-\ell}$ compression function calls.

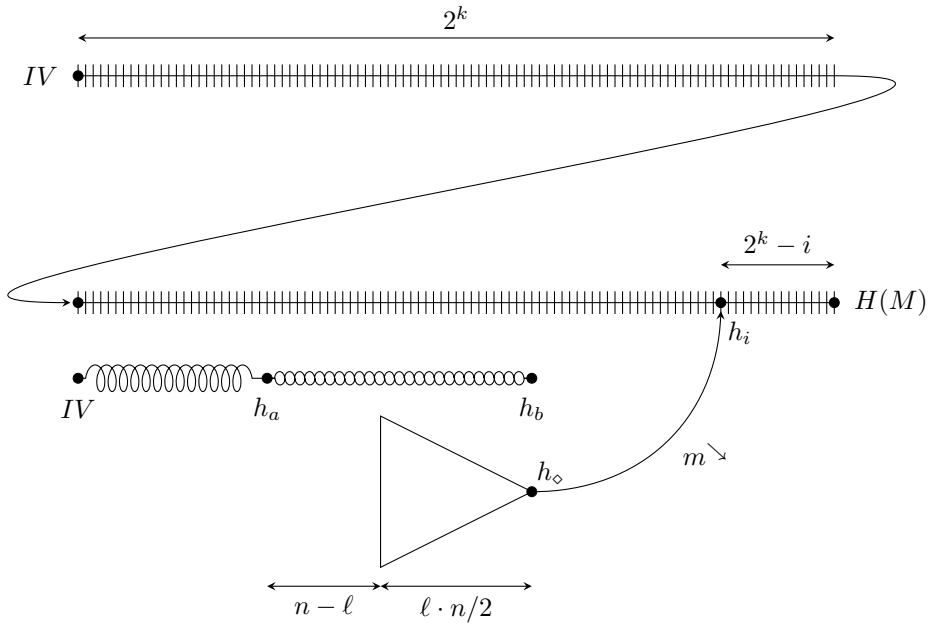
5.2.2 More Herding: Merkle-Damgård-Again

The attack is essentially the same as the one against the concatenation of two hash functions, as shown in figure 5.4. The adversary commits to $h^2_\diamond$, and is then being challenged with an unknown prefix P. Hashing the prefix yields a chaining value h_c. Starting from h_c, she chooses a message block $m_1^\nearrow$ connecting to a chaining value h_e which is one of the starting points of the first diamond, then a path S_1 inside it yields the chaining value $h^1_\diamond$ on the first pass, from which we traverse a precomputed $2^{n-\ell+n\ell/2}$-multicollision, producing h_2 as the input chaining value to the second pass. Starting from h_2, the challenge prefix P leads to a random chaining value $h_{c'}$ in the second pass. Then, the second pass can be herded without losing control of the chaining value in the first pipe thanks to the diamond built on top of the multicollision. Amongst the $2^{n-\ell}$ messages in the multicollision following the first diamond, we expect one to connect to the chaining value $h_{e'}$ in the starting points of the second diamond. We can then follow a path inside the second diamond, which is also a path in the multicollision of the first pipe, that yields the chaining value at the root of the second diamond, namely $h^2_\diamond$.

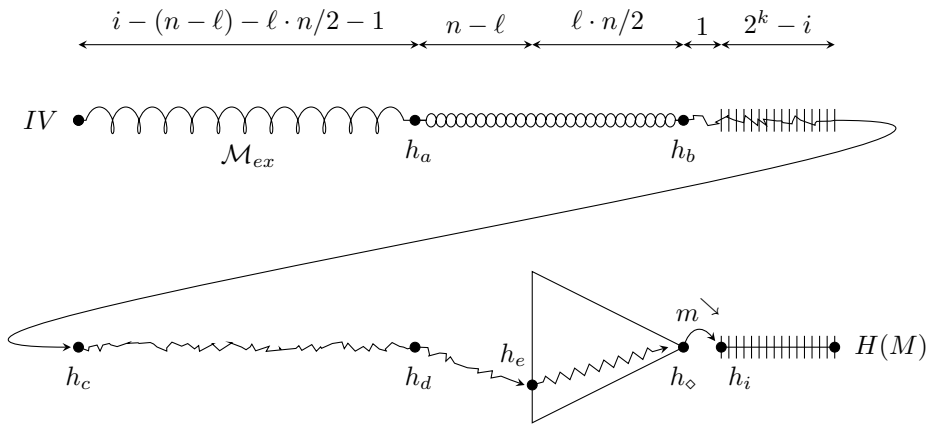
The offline complexity of the attack is the time required for generating a diamond structure of 2^ℓ starting points (which takes $2^{(n+\ell)/2+2}$), finding $(n - \ell) + n \cdot \ell/2$ collisions (which takes $[(n - \ell) + n \cdot \ell/2] \cdot 2^{n/2}$), and constructing a two-pipe diamond (which takes $2 \cdot 2^{(n+\ell)/2+2}$). The total offline complexity is therefore $3 \cdot 2^{(n+\ell)/2+2}$ compression function evaluations.

The online complexity is composed of finding two connecting "messages". The first search takes $2^{n-\ell}$, while the second takes $2 \cdot 2^{n-\ell}$, or a total of $3 \cdot 2^{n-\ell}$.

Attacks on Merkle-Damgård-Once-More. It is relatively clear that the attack can be generalized to the case where the message is hashed three or more times (by using multicollisions on 3 pipes, or the respective number of passes). The complexity of the attack becomes polynomially higher, though.



(a) First online step



(b) Online steps 2 to 5

Figure 5.5: Second preimage attack on Merkle-Damgård-Again

5.3 A Generic Second Preimage Attack on Merkle-Damgård-Again

If a construction is susceptible to the herding attack, then it is natural to ask whether the second preimage attack of chapter 4 is applicable. Recall that the general idea of this attack is to connect the root of the diamond structure to some chaining value encountered during the hashing of the target message, and then connect into the diamond structure (either from the corresponding location in the original message or from a random prefix). This ensures that the new message has the same length (foiling the Merkle-Damgård strengthening).

In this section, we present a second preimage attack against the Merkle-Damgård-Again construction. The general strategy is to build a diamond structure, and try to connect it to the challenge message (in the second pass). Some complications appear, because the connection may happen anywhere, and the diamond only works on top of a multicollision that has to be located somewhere in the first pass. However, we can use an *expandable message* (cf. §3.3.4) to move the multicollision (and therefore the diamond) around. Here is a complete description of the attack. Let us assume that the adversary is challenged with a message M of 2^κ blocks.

The offline procedure is as follows:

1. Generate an expandable message which can take any length between κ and $2^\kappa + \kappa - 1$, starting from

the IV and yielding a chaining value h_a.

2. Starting from h_a, generate a multicollision of length $(n-\ell)+\ell \cdot n/2$ blocks, yielding a chaining value h_b.
3. Build a diamond structure on top of the multicollision. It yields a chaining value $h_\diamond$. It is used to herd the second pass.

The online phase, given a message M, is as follows (depicted in Figure 5.5):

1. Given $h_\diamond$, select at random message blocks $m^{\searrow}$ until $f(h_\diamond, m^{\searrow})$ equals to a chaining value h_i appearing in the second pass.
2. To position the end of the diamond at the i-th block of M, generate the instance of the expandable message of length $i - n \cdot \ell/2 - (n - \ell)$ blocks, and let us denote it by $\mathcal{M}_{ex}$. Note that we have $h_a = f^*(IV, \mathcal{M}_{ex})$.
3. Let $h_c = f^*(h_b, m_{i+1}, m_{i+2}, \ldots, m_{2^\kappa-1})$. Compute the second pass on the expandable message, until h_d is reached: $h_d = f^*(h_c, \mathcal{M}_{ex})$. Now, using the freedom in the first $n-\ell$ blocks of the multicollision, find a message $m^{\nearrow}$ that sends h_d to a chaining value h_e occurring in the starting points of the diamond in the second pass.
4. Find a path T inside the diamond in the second pass (this is also a path inside the multicollision of the first pass). It yields the chaining value $h_\diamond$ at the root of the diamond in the second pipe.
5. Let $M' = \mathcal{M}_{ex} \,\|\, m^{\nearrow} \,\|\, T \,\|\, m^{\searrow} \,\|\, m_{i+1}, \ldots, m_{2^\kappa-1}$. We readily find that $H(M') = H(M)$.

Note that the message forged by assembling the right parts has the same length as M, therefore the padding scheme act the same way on both. The offline complexity of the attack is the mostly dominated by the need to construct a diamond structure on two pipes, *i.e.*, $2 \cdot 2^{(n+\ell)/2+2}$. The online time complexity is $2^{n-\kappa}$ for finding m, and $2 \cdot 2^{n-\ell}$ connecting to the diamond structure. Hence, the total online time is $2^{n-\kappa} + 2^{n+1-\ell}$, essentially the same as the attack on the regular Merkle-Damgård mode of operation.

5.4 The Trojan Message Attack

> Do not trust the horse, Trojans.
> Whatever it is, I fear the Greeks
> even when they bring gifts
>
> Virgil's Aeneid, Book 2, 19 BC

In this section, we introduce a new generic attack on many hash function constructions, called the *Trojan Message attack*. A Trojan message is a string S which is produced offline by an attacker, and is then provided to a victim. The victim then selects some prefix P from a constrained set of choices, and hashes $P \,\|\, S$. However, due to the way S was chosen, the attacker is now able to find a second preimage of $H(P \,\|\, S)$.

Given a Merkle-Damgård hash for which collisions may be found, Trojan messages may be produced. In general, the Trojan message requires at least one message input block, and one collision search, per possible value of P. If there are 1024 possible values of P, an attacker may produce a 1024-block Trojan message, requiring 1024 collision searches.

One can imagine a Trojan message attack being practical against applications which use MD5, and which permit an attacker to provide the victim with "boilerplate" text for the end of his document, while imposing a relatively constrained set of choice for his part of the document.

Against Merkle-Damgård hashes, Trojan message attacks take two forms:

1. If only the Collision procedure from algorithm 3.2 is available, second preimages for the full message keep the victim's choice of P, but introduce a limited change in S. That is, the attacker finds $S' \neq S$ such that $H(P \,\|\, S) = H(P \,\|\, S')$.
2. If however the Dual-IV-Colliision procedure is also practical, second preimages for the full message give the attacker a choice of P, and leave S mostly unchanged. That is, the attacker finds P' and S' such that $H(P \,\|\, S) = H(P' \,\|\, S')$.

Let $\mathcal{P} = \{P_1, \ldots, P_N\}$ be a set of N known prefix messages and $h_0^i = f^*(IV, P_i)$ be the intermediate chaining value resulting from the computation of P_i. Note, that without loss of generality, we can assume that all the prefixes have the same length (otherwise, we just consider padded versions). Therefore, we safely disregard strengthening and padding issues.

5.4.1 Collision-Variant

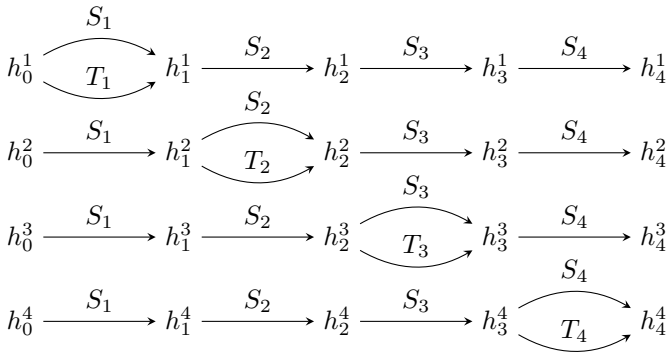
The collision variant of the Trojan message attack makes use of the Collision procedure of Algorithm 3.2 (or any *ad hoc* equivalent), which takes a chaining value as parameter and produces a pair of messages colliding from this chaining value. The attack proceeds as follows:

1. $\mathcal{A}$ computes N colliding message pairs (S_i, T_i) using Algorithm 5.1.
2. $\mathcal{A}$ sends $\mathcal{B}$ a suffix message $S = S_1 \,\|\, \ldots \,\|\, S_N$.
3. $\mathcal{B}$ commits to $h = H^f(P_i \,\|\, S)$ where P_i belongs to $\mathcal{P}$.
4. $\mathcal{A}$ finds out P_i through exhaustive search amongst the N possible choices and outputs:

$$M' = P_i \,\|\, S_1 \,\|\, \ldots \,\|\, T_i \,\|\, \ldots \,\|\, S_N$$

Algorithm 5.1 Trojan Message Attack, Collision Variant

```
1: function COLLISION-TROJAN-SUFFIX(P_1, ..., P_N)
2:     h_0 ← IV
3:     for i = 1 to N do
4:         (S_i, T_i) ← Collision(h_{i-1}^i)
5:         for j = 1 to N do
6:             h_i^j ← f(h_{i-1}^j, S_i)
7:         end for
8:     end for
9:     return (S_1, T_1), ..., (S_N, T_N)
10: end function
```



We indeed verify that $H^f(M') = h$. The hash of $P_i \,\|\, S$ and $P_i \,\|\, S'$ differs only when T_i replaces S_i, but because these two blocks collide, then the two hash processes do not diverge.

The only non-trivial part of the attack for $\mathcal{A}$ is the first step where she precomputes N collisions for each prefix from the set $\mathcal{P}$ (in time $N \cdot 2^{n/2}$), and evaluates the compression function N^2 times. If finding a collision for the hash function is easy, for instance on legacy hash functions such as MD5, then the attack can even be practical. It has recently been shown that finding a collision in MD5 takes about 2^{16} evaluations of the compression function [SSA+09]. For instance, one can forge in a matter of seconds a suffix S of 46720 bytes permitting to find second preimages for MD5 if the prefix set $\mathcal{P}$ is the set of the days of the year.

5.4.2 The Herding Trojan Attack

The herding variant of the trojan message attack is stronger, and allows for more freedom for the attacker. In exchange, the preprocessing and the online running times are larger.

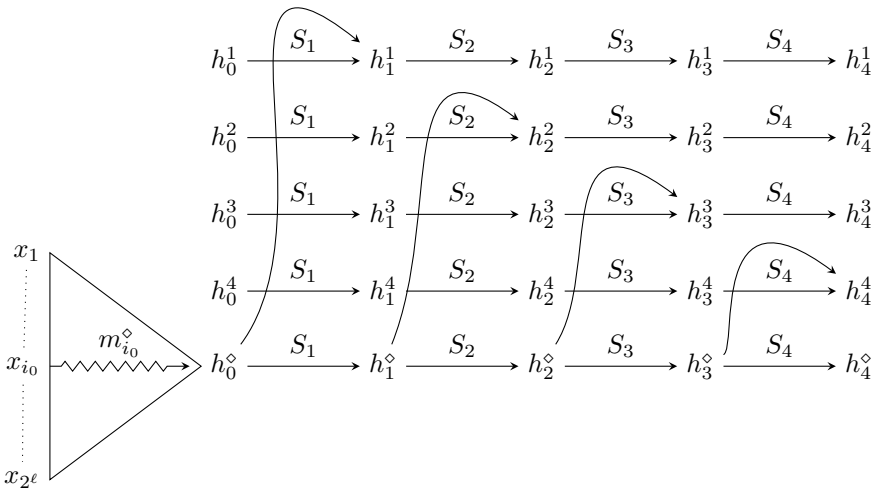
Let K denote the length of all possible prefixes in $\mathcal{P}$. We can extend K to be as large as we wish. The herding variant of the Trojan message attack makes use of a different, more sophisticated "chosen-prefix" procedure **Dual-IV-Collision**(h_1, h_2) from Algorithm 3.2, which returns two messages m_1 and m_2, such that $f(h_1, m_1) = f(h_2, m_2)$. In some specific cases this collision is harder to find (for instance in MD5, such collision takes 2^{41} compression function evaluations [SSA+09]).

Another difference between this variant and the previous one, is that in this variant, the adversary is challenged by a second prefix P', not controlled by him, which he has to herd to the same value as $H^f(P_i \,\|\, S)$. The attack proceeds as follows:

1. $\mathcal{A}$ computes a diamond structure with 2^ℓ entry points, denoted by $D_1 = \{x_i\}$, converging to the hash value $h_0^\diamond$, with the constraint that $\ell < K - 2$.
2. $\mathcal{A}$ generates N colliding message pairs using Algorithm 5.2.
3. $\mathcal{A}$ sends $\mathcal{B}$ a suffix message $S = S_1 \,\|\, \ldots \,\|\, S_N$.
4. $\mathcal{B}$ commits to $h = H^f(P_i \,\|\, S)$ where $P_i \in \mathcal{P}$.

Algorithm 5.2 Trojan Message Attack, Herding Variant

```
1: function HERDING-TROJAN-SUFFIX(P_1, ..., P_N)
2:     for i = 1 to N do
3:         (S_i, T_i) ← Dual-IV-Collision(h^i_{i-1}, h^◇_{i-1})
4:         for j = 1 to N do
5:             h^i_j ← f(h^{i-1}_j, S_i)
6:         end for
7:         h^◇_i ← f(h^◇_{i-1}, S_i)
8:     end for
9: end function
```



5. $\mathcal{A}$ is challenged with an arbitrary prefix P' of size at most $K - \ell - 1$ blocks, not necessarily in the known prefix set.
6. $\mathcal{A}$ finds (by random trials) a connecting message C of size $K - \ell - |P'|$ blocks such that $x_{i_0} = f^*(P' \,\|\, C) \in D_1$.
7. $\mathcal{A}$ forges a new prefix $Q = P' \,\|\, C \,\|\, m_{i_0}^\diamond$, which is such that $f^*(Q) = h_0^\diamond$.
8. As in the collision version, $\mathcal{A}$ outputs $Q \,\|\, S_1 \,\|\, \ldots \,\|\, T_i \,\|\, \ldots \,\|\, S_N$.

As in the collision variant, we have that $H^f(Q \,\|\, S') = h$. The reasoning to establish this fact is essentially the same. The workload of the attack is step one where $\mathcal{A}$ constructs a diamond structure with 2^ℓ starting points and N collisions for each prefix from the set P. Thus, the precomputation complexity is of order $2^{n/2+\ell/2+2} + N \cdot 2^{n/2}$. The online cost is the connection step for computing the prefix P' and is of order $2^{n-\ell}$.

Applications of the Trojan Attacks The trojan attack can be highly useful in instances with a set of predictable prefixes, and where the attacker is able to suggest a suffix to introduce to the message. Such a case is the X.509 certificate, where the adversary may generate a second certificate (with the same identification) but with different public keys. Another possible application is a time stamping service, which signs $H(ts, M)$ where ts is a time stamp and M is the document.

Chapter 6

Time-Memory Tradeoffs for Second Preimage Attacks

In this chapter, we briefly investigate how Time-Memory Tradeoff (TMTO) techniques apply to generic second preimage attacks. Joint work with Elena Andreeva, Orr Dunkelman, Jonathan Hoch and Adi Shamir

In the "standard" Time-Memory tradeoff (TMTO) discovered by Hellman [Hel80], the objective is to invert an arbitrary function $H : \{0,1\}^* \to \{0,1\}^n$. For this puprose, a precomputation requiring 2^n evaluations of H is used to generate a special data-structure of size $2^{2n/3}$, which subsequently allows to find a preimage of most points of $\{0,1\}^n$ with only $2^{2n/3}$ evaluations of H. This general technique allows, after an enormous precomputation, to find preimages (and thus *second*-preimages) of any message in (online) time $2^{2n/3}$. However, it cannot take advantage of the length of the target message.

We show in §6.2 that Time-Memory Tradeoffs can be applied to speed up all the known generic second preimage attacks: after a preprocessing of complexity $2^{n-\kappa}$, the adversary is able to find second preimages of long messages in online time $2^{2(n-\kappa)/3}$. We finally turn our attention to Rivest's Dithered Hashing again in §6.3, and we derive attacks capable of coping with arbitrary dithering sequences after a precomputation of order 2^n.

6.1 Hellman's Time-Memory Tradeoff Attack

Time-memory Tradeoff attacks (TMTO) were first introduced in 1980 by Hellman [Hel80]. The idea is to improve brute force attacks by trading the online time for memory and precomputation when inverting a function $f : \{0,1\}^n \to \{0,1\}^n$. Suppose we have an image element y and we wish to find a pre-image x such that $f(x) = y$. One extreme would be to go over all possible elements x until we find one such that $f(x) = y$, while the other extreme would be to precompute a huge table containing all the pairs $(x, f(x))$ sorted by the second element. Hellman's idea is to consider what happens when applying f iteratively. We start at a random element x_0 and compute $x_{i+1} = f(x_i)$ for t steps saving only the start and end points of the generated chain (x_0, x_t). We repeat this process with different initial points and generate a total of c chains. Given an input y, we start generating a chain starting from y and checking if we reached one of the saved endpoints. If we have, we generate the corresponding chain, starting from the suggested starting point and hope to find a preimage of y. Notice that as the number of chains, c, increases beyond $2^n/t^2$, the contribution (*i.e.*, the number of new values that can be inverted) from additional chains decreases. To counter this birthday paradox effect, Hellman suggested to construct a number of tables, each using a slightly different function f_i, such that knowing a preimage of y under f_i implies knowing such a preimage under f. Hellman's original suggestion, which works well in practice, is to use $f_i(x) = f(x \oplus i)$. Thus, if we create $d = 2^{n/3}$ tables each with different f_i's, such that each table contains $c = 2^{n/3}$ chains of length $t = 2^{n/3}$, about 80% of the 2^n points will be covered by at least one table. Notice that the online time complexity of Hellman's algorithm is $t \cdot d = 2^{2n/3}$ while the memory requirements are $d \cdot c = 2^{2n/3}$.

It is worth mentioning, that when multiple targets are given for inversion (*i.e.*, a set of possible targets $y_i = f(x_i)$), where it is sufficient to identify only one of the preimages (x_i for some i), one could offer better trade off curves. For example, given m possible targets, it is possible to reduce the number of tables stored by a factor of m, and trying for each of the possible targets, the attack (*i.e.*, apply the chain). This reduces the memory complexity (without affecting the online time complexity or success rate), as long as $m \leq d$ (see [BS00] for more details concerning this constraint).

6.2 Time-Memory-Data Tradeoffs For Second Preimage Attacks

All known generic second preimage attacks (including that of chapter 4) assume that the target message is very long (2^κ blocks). This enables the connection to the target message to be done with about $2^{n-\kappa}$ compression function calls by exhaustive search. This "connection" step can be seen as finding the inverse of a function.

Recall that in the second preimage attack of chapter 4, we search for a message block $m^{\searrow}$ such that $f(h_\diamond, m^{\searrow}) = h_i$. Something very similar happens in the attacks of Dean and Kelsey-Schneier. As there are 2^κ targets (and finding the preimage of only one h_i's is sufficient), then we can run a time-memory-data tradeoff attack with a search space of $N = 2^n$, $D = 2^\kappa$ available data points, time T, and memory M such that $N^2 = TM^2D^2$, after $P = N/D$ preprocessing (and $T \geq D^2$). Let 2^x be the online complexity of the time-memory-data tradeoff, with the constraint that $2^x \geq 2^{2\kappa}$, and the memory consumption is $2^{n-\kappa-x/2}$ blocks of memory. The resulting overall complexities are $2^{n/2+\ell/2+2} + 2^{n-\kappa}$ preprocessing, $2^x + 2^{n-\ell}$ online complexity, and $2^{\ell+1} + 2^{n-\kappa-x/2}$ memory, for messages of $2^{x/2}$ blocks.

Given the constraints on the online complexity (*i.e.*, $x \geq 2\kappa$), it can possibly be beneficial to consider shorter messages, *e.g.*, message of 2^λ blocks (with $\lambda \leq \kappa$). For such cases, the offline complexity is $2^{n/2+\ell/2+2} + 2^{n-\lambda}$, the online complexity is $2^x + 2^{n-\ell}$, and the memory consumption is $2^{n-\lambda-x/2} + 2^{\ell+1}$.

We can balance the online and memory complexities (as commonly done in time-memory-data tradeoff attacks) by picking x such that $2^x + 2^{n-\ell} \approx 2^{n-\lambda-x/2} + 2^{\ell+1}$. By picking $\lambda = n/4$, $x = 2\lambda = n/2$, and $\ell = n/2$, the online complexity is $2^{n/2+1}$, the memory complexity is $3 \cdot 2^{n/2}$, and the offline complexity is $5 \cdot 2^{3n/4}$. This of course holds as long as $n/4 = \lambda \leq \kappa$, or equivalently $n \leq 4\kappa$.

We can for instance find a second preimage for a 2^{40}-block long message in SHA-1, with an online time of 2^{81} operations, $2^{81.6}$ blocks of memory, and 2^{120} steps of precomputation. The equivalent Kelsey-Schneier attack takes 2^{120} online steps (and about 2^{71} offline computation). The standard time-memory tradeoff for finding preimages on an n-bit function allows, after a preprocessing of complexity 2^n, to find preimages using time 2^x and memory $2^{n-x/2}$. Hence, for the same 2^{40}-block message and $2^{81.6}$ blocks of memory, the online computation is about $2^{156.8}$ SHA-1 compression function calls, only marginally faster than exhaustive search.

Second Preimage Attack with Multiple Targets The existing generic second preimage attacks can be applied efficiently to multiple target messages. The work needed for these attacks depends on the number of intermediate hash values of the target message, as this determines the work needed to find a linking message from the data structure used to forge a prefix of the challenge. A set of 2^R messages, each of 2^κ blocks, has the same number of intermediate hash values as a single message of $2^{R+\kappa}$ blocks, and so the difficulty of finding a second preimage for one of a set of 2^R such messages is no greater than that of finding a second preimage for a single $2^{R+\kappa}$ block target message. In general, for the older second preimage attacks, the total work to find one second preimage falls linearly in the number of target messages; for our attack, it falls also linearly as long as the total number of message blocks, 2^S, satisfies $S < (n-4)/3$.

Consider for example an application which has used SHA-1 to hash 2^{30} different messages, each of 2^{20} message blocks. Finding a second preimage for a given one of these messages using the attack of Kelsey and Schneier requires about 2^{141} work. However, finding a second preimage for *one* of these of these 2^{30} target messages requires 2^{111} work. (Naturally, the adversary cannot control for *which* target message he finds a second preimage.)

This works because we can consider each intermediate hash value in each message as a potential target to which the root of the collision tree (or an expandable message) can be connected, regardless of the message it belongs to, and regardless of its length. Once we connect to an intermediate value, we have to determine to which particular target message it belongs. Then we can compute the second preimage of that message.

This observation is important for two reasons: first, simply restricting the length of messages processed by a hash function is not sufficient to block the long-message attack. Second, this observation allows long-message second preimage attacks to be applied to target messages of practical length. A second preimage attack which is feasible only for a message of 2^{50} blocks has no practical relevance, as there are probably no applications which use messages of this length. A second preimage attack which can be applied to a large set of messages of, say, 2^{24} blocks each, might have some practical impact. While the computational requirements of these attacks are still infeasible, this observation shows that the attacks can apply to messages of practical length. Moreover, for hashes which use the same dithering sequence $\mathbf{z}$ in all invocations, this has an affect on the frequency of the most common factors (especially when the most common factor is relatively in the beginning of the dithering sequence, *e.g.*, Shoup's UOWHF with the same set of keys).

6.3 Dealing with High Complexity Dithering Sequences

As discussed before, a possible improvement to the Dithered Hash that would prevent our second preimage attack would be to use a very-high complexity sequence. In this section, we give several attacks against Dithered Hashing *in the precomputation model*, and whose complexity is independent of the dithering sequence. In the precomputation model, the adversary is allowed to perform a very expensive precomputation, whose complexity may be of order 2^n, after which she may store a data-structure of size strictly less than 2^n. When presented with a challenge, the adversary may make use of this data-structure to find a solution faster. The idea is that the precomputation may be amortized over any number of subsequent forgeries.

We explore various techniques to break arbitrary dithering sequences. We start with a simple generalization of our proposed attack. We follow with two new attacks which have an expensive precomputation, in exchange for a much faster online phase: The kite generator and a variant of Dean's attack tailored to these settings.

6.3.1 Generalization of the Attack on Dithered Merkle-Damgård

The main limiting factor of our attack on Dithered Hashing is the fact that the diamond structure can only be positioned in specific locations of the target message. Once the sequence is of high enough complexity (and assuming that there is no real bias in the factor distribution), then there are not enough "good" positions where the attack can be performed. To overcome this difficulty, a possible solution is to build a more powerful herding device that works at all the possible locations in the target message.

To build this "dithering-oblivious" diamond structure, we generate a converging tree in which each node is a $2|\mathcal{A}|$-collision. Specifically, for a pair of starting points w_0 and w_1 we find a $2|\mathcal{A}|$-collision under different dithering letters, *i.e.*, we find $m_0^1, \ldots, m_0^{|\mathcal{A}|}$ and $m_1^1, \ldots, m_1^{|\mathcal{A}|}$ such that:

$$\begin{aligned} & f(w_0, m_0^1, \alpha_1) = f(w_0, m_0^2, \alpha_2) = \cdots = f\left(w_0, m_0^{|\mathcal{A}|}, \alpha^{|\mathcal{A}|}\right) \\ = \quad & f(w_1, m_1^1, \alpha_1) = f(w_1, m_1^2, \alpha_2) = \cdots = f\left(w_1, m_1^{|\mathcal{A}|}, \alpha^{|\mathcal{A}|}\right) \end{aligned}$$

This way, we can position the diamond structure in any position, unrelated to the actual dithering sequence, as we are assured that there is a path from any leaf to the root labeled by any possible combination of letters.

To build the required diamond structure we propose the following algorithm: first for each starting point (out of the 2^ℓ) find a $|\mathcal{A}|$-collision (under the different dithering letters). Now, it is possible to find collisions between different starting points (just like in the original diamond structure, where we use a $|\mathcal{A}|$-collision rather than one message). Hence, the total number of $|\mathcal{A}|$-collisions which are needed from one specific starting point (in order to build the next layer of the collision tree) is $2^{n/2-\ell/2}$. The cost for building this number of $|\mathcal{A}|$ collisions is $2^{\frac{2|\mathcal{A}|-1}{2|\mathcal{A}|}n-\frac{\ell}{2|\mathcal{A}|}}$, or a total of $2^{\frac{2|\mathcal{A}|-1}{2|\mathcal{A}|}(n+\ell)+2}$ for the preprocessing step.

After the computation of the diamond structure (which may take more than 2^n), one can connect to any point in the message, independent of the used dithering letter. Hence, from the root of the diamond structure we try the most common dithering letter, and try to connect to all possible locations (this takes time $2^{n-\kappa+H_\infty(\mathbf{z},1)} \leq \left|\mathcal{A}\right| \cdot 2^{n-\kappa}$). Connecting from the message to the diamond structure takes $2^{n-\ell}$ as before.

The memory required for storing the diamond structure is $\mathcal{O}\left(|\mathcal{A}| \cdot 2^\ell\right)$. We note that the generation of the $|\mathcal{A}|$-collision can be done using the results of [JL09], which allow balancing between the preprocessing's time and its memory consumption.

6.3.2 The Kite Generator—Dealing with Small Dithering Alphabets

Even though the previous attack could handle any dithering sequence, its online complexity is about the same as that of the simpler second preimage attacks on Merkle-Damgård. We could in addition try to leverage the huge precomputation required by the attack to further reduce the online complexity. To this end, we introduce a new technique, the *kite generator*. The kite generator shows that a small dithering alphabet is an inherent weakness, and after a $\mathcal{O}\left(2^n\right)$ preprocessing, second preimages can be found for messages of length $2^\kappa \leq 2^{n/4}$ in $\mathcal{O}\left(2^{2\cdot(n-\kappa)/3}\right)$ time and space for any dithering sequence (even of maximal complexity). Second preimages for longer messages can be found in time $\max\left(\mathcal{O}\left(2^k\right), \mathcal{O}\left(2^{n/2}\right)\right)$ and memory $\mathcal{O}\left(\left|\mathcal{A}\right| \cdot 2^{n-k}\right)$.

6.3.2.1 Outline of the Attack.

The kite generator uses a different approach, where the connections to and from the message are done for free, regardless of the dithering sequence.

During the precomputation phase the adversary builds a static data structure, the kite generator: she picks a set of $2^{n-\kappa}$ chaining values, B, that contains the IV. For each chaining value $x \in B$ and any dither

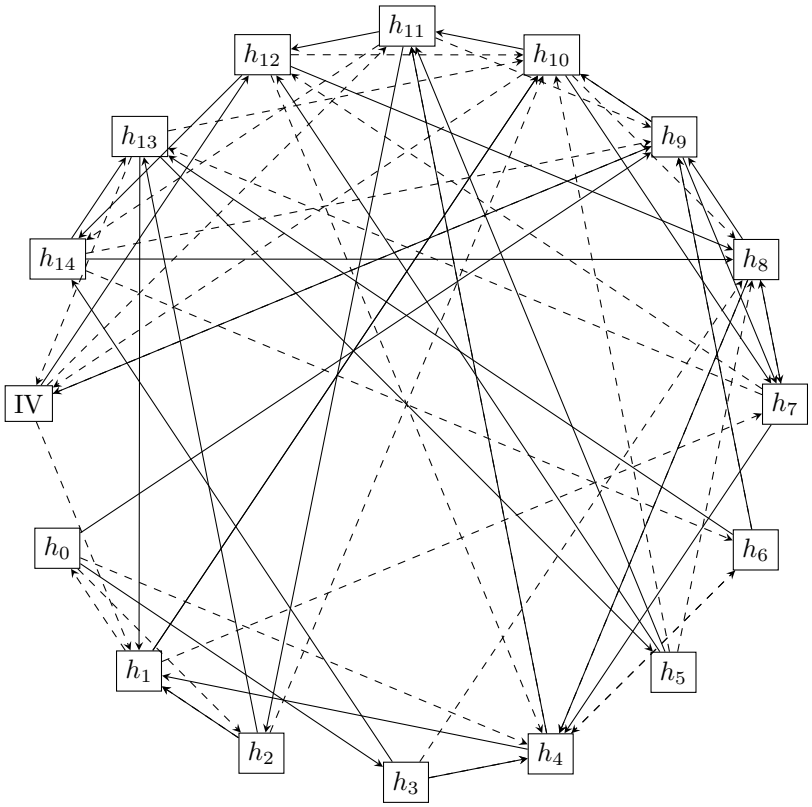


Figure 6.1: A toy "Kite-Generator" with 16 nodes over a binary alphabet. Edges are labelled with a message block and a letter. Hard edges correspond to the first letter, and dashed edges to the second letter. This big straw ball unfolds into kites!

letter $\alpha \in \mathcal{A}$, the adversary finds two message blocks $m_{x,\alpha,1}$ and $m_{x,\alpha,2}$, such that both $f(x, m_{x,\alpha,1}, \alpha)$ and $f(x, m_{x,\alpha,2}, \alpha)$ belong to B. The adversary then stores all $m_{x,\alpha,1}$ and all $m_{x,\alpha,2}$ in the data structure. Figure 6.1 shows a toy kite generator.

In the online phase of the attack, given a message M, the adversary computes $H(M)$, and finds with high probability (thanks to the birthday paradox) an intermediate chaining value $h_i \in B$ (for $2^\kappa > i > n - \kappa$). The next step of the attack is to find a sequence of i blocks from the IV that leads to this h_i. This is done in two steps. In the first step, the adversary perform a random walk in the kite generator, by just picking random $m_{x,\alpha,i}$ one after the other (in conformance with the dither sequence), until $h'_{i-(n-\kappa)}$ is computed (this $h'_{i-(n-\kappa)}$ is independent of the value found in the computation of $H(M)$). At this point, the adversary stops her random walk, and computes from $h_{i-(n-\kappa)}$ all the possible $2^{(n-\kappa)/2}$ chaining values reachable through any sequence of $m_{x,\alpha,1}$ or $m_{x,\alpha,2}$ (which agrees with the dithering sequence)—this amounts to consider all the paths starting from where the random walk stopped inside the kite generator and trying all the paths whose labels agree with the dithering sequence. Then, the adversary computes the "inverse" tree, starting from h_i, and listing the expected $2^{(n-\kappa)/2}$ values [1] that may lead to it following the dither sequence. If there is a collision between the two lists (which happens with high probability due to the birthday paradox), then the adversary just found the required path—she "connected" the IV to h_i. Figure 6.2 illustrates the process.

The precomputation takes $\mathcal{O}\left(\left|\mathcal{A}\right| \cdot 2^{n-\kappa} \cdot 2^\kappa\right) = \mathcal{O}\left(\left|\mathcal{A}\right| \cdot 2^n\right)$. The memory used to store the kite generator is $\mathcal{O}\left(\left|\mathcal{A}\right| \cdot 2^{n-\kappa}\right)$. The online phase requires $\mathcal{O}\left(2^\kappa\right)$ compression function calls to compute the chaining values associated with M, and $\mathcal{O}\left(2^{(n-\kappa)/2}\right)$ memory and time for the meet-in-the-middle phase. [2] We conclude that the online time is $\max\left(\mathcal{O}\left(2^\kappa\right), \mathcal{O}\left(2^{(n-\kappa)/2}\right)\right)$ and the total used space is $\mathcal{O}\left(\left|\mathcal{A}\right| \cdot 2^{n-\kappa}\right)$. For the SHA-1 parameters of $n = 160$ and $\kappa = 55$, the time complexity of the new attack is 2^{55}, which is just the time needed to hash the original message. However, the size of the kite generator for the above parameters exceeds 2^{110}.

To some extent, the "converging" part of the kite generator can be treated as a diamond structure (for each end point, we can precompute this "structure"). Similarly, the expanding part, can be treated as the

1. See [Fel71] for a formal justification of this claim.
2. The meet-in-the-middle can be done using memoryless variants as well.

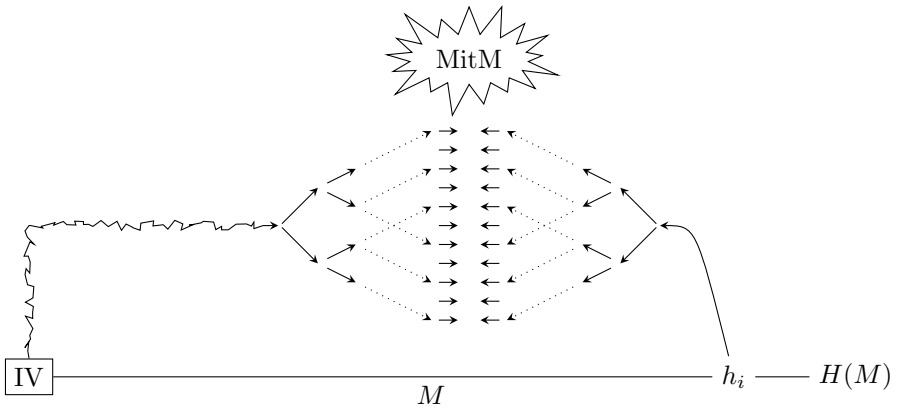


Figure 6.2: A "Kite" connected to and from the message. The "tail" is the random walk inside the kite generator. Then comes the divergent tree and the convergent tree (which is very reminiscent of a diamond structure) leading to h_i, the intermediate chaining value occurring in both M and in the kite generator. There exist a path between both trees with high probability.

trials to connect to this diamond structure from $h'_{i-(n-\kappa)}$.

We note that the attack can also be applied when the IV is unknown in advance (*e.g.*, when the IV is time dependent or nonce), with essentially the same complexity. When we hash the original long message, we have to find two intermediate hash values h_i and h_j (instead of IV and h_i) which are contained in the kite generator and connect them by a properly dithered kite-shaped structure of the same length.

The main problem of this technique is that for the typical case in which $\kappa < n/2$, it uses more space than time, and if we try to equalize them by reducing the size of the kite generator, we are unlikely to find any common chaining values between the given message and the kite generator.

6.3.3 A Variant of Dean's Attack for Small Dither Alphabet

In fact, the kite generator can be seen as an expandable message tolerating the dithering sequence, and we can use it in a more "traditional" way. We first pick a special chaining value N in the kite generator. From this N we are going to connect to the message (following the approaches suggested earlier, as if N were the root of a diamond structure). Then, it is possible to connect from the IV to N inside the kite generator, for any possible dithering sequence

For a kite containing 2^ℓ points, the offline complexity is $\mathcal{O}\left(\left|\mathcal{A}\right| \cdot 2^n\right)$, and the online complexity is

$$2^{n-\kappa+H_\infty(\mathbf{z},1)} + 2^\kappa + 2^{\ell/2+1}.$$

The memory required for the attack is $\mathcal{O}\left(2^\ell\right)$. It follows that we have nothing to lose to setting $\ell = 0$, and interestingly this degenerate case of the kite generator can be considered as an adaptation of Dean's attack to the case of a small dithering alphabet.

Assume that the kite generator contains only one chaining value, namely, IV. For each dither letter α, we find x_α such that $f(IV, x_\alpha, \alpha) = IV$. Then, we can "move" from IV to IV under any dithering letter. At this point, we connect from the IV to the message (either directly, or using time-memory-data tradeoff), and "traverse" the degenerate kite generator under the different dithering letters.

Hence, a standard implementation of this approach would require $\mathcal{O}\left(\left|\mathcal{A}\right| \cdot 2^n\right)$ precomputation and $2^{n-\kappa+H_\infty(\mathbf{z},1)}$ online computation (with $\left|\mathcal{A}\right|$ memory). A time-memory-data variant can reduce the online computation to $2^{2(n-t-\kappa+H_\infty(\mathbf{z},1))}$ in exchange for 2^t memory (as long as $t \leq n - 2(\kappa - H_\infty(\mathbf{z},1))$). Table 6.1 compares all the techniques suggested for dithered hashing.

Attack	Complexity			Avg. Patch
	Offline	Online	Memory	
Single-$\diamond$ (§4.2.2)	$2^{(n+\ell)/2+2}$	$2^{n-\kappa+H_\infty(\mathbf{z},\ell+1)} + 2^{n-\ell}$	$2^{\ell+1}$	$\ell+2$
Multi-$\diamond$ (§4.2.3)	$2^k(2^{(n+\ell)/2+2} + 2^{n/2})$	$2^{n-\kappa+H^k_\infty(\mathbf{z},\ell+1)} + 2^{n-\ell}$	$2^{k+\ell+1}$	$k+\ell+2$
general. (§6.3.1)	$2^{\frac{2\lvert\mathcal{A}\rvert-1}{2\lvert\mathcal{A}\rvert}\cdot(n+\ell)+2}$	$2^{n-\kappa+H_\infty(\mathbf{z},1)} + 2^{n-\ell}$	$\lvert\mathcal{A}\rvert \cdot 2^{\ell+1}$	$\ell+2$
Kite gen. (§6.3.2)	$\lvert\mathcal{A}\rvert \cdot 2^n$	$2^\kappa + 2^{(n-\kappa)/2+1}$	$\lvert\mathcal{A}\rvert \cdot 2^{n-\kappa+1}$	$2^{\kappa-1}$
"Self-loop" (§6.3.3)	$\lvert\mathcal{A}\rvert \cdot 2^n$	$2^{n-\kappa+H_\infty(\mathbf{z},1)}$	$\lvert A\rvert$	$2^{\kappa-1}$

$H^k_\infty(\mathbf{z},\ell+1)$ — the maximal sum of probabilities for 2^k suffix-friendly set of length $\ell+1$.

Table 6.1: Comparison of Long Message Second Preimage Attacks on Dithered Hashing

Chapter 7

Provable Security for Modes of Operations

This chapter presents some results on the provable security of hash function modes of operation. This is the result of a joint work with Gaëtan Leurent and Sébastien Zimmer. The results of §7.3 have been published at SAC'2010 [BFL10].

The previous chapters presented several generic attacks against hash functions, *i.e.*, attacks on their modes of operation. This chapter concludes our study modes of operation of hash function by looking at them from the *provable security* angle. One of the most distinctive feature of the Merkle-Damgård mode of operation is that it *provably* promotes a collision-resistant compression function into a collision-resistant hash function. The situation is very different for second-preimage resistance, since even when the compression function is ideal, the generic second preimage attacks of Dean, Kelsey and Schneier, and our own (chapter 4) show that the iteration is not as secure as a Random Oracle.

The exact resistance of Merkle-Damgård to generic second preimage attacks is thus unknown. The known attacks give an upper-bound of order $2^n/\ell$ for messages of size ℓ, above the birthday paradox, and the fact that a second preimage is also a collision gives a birthday lower-bound (because of the collision-resistance preservation). In addition, the resistance of Haifa to generic second-preimage attacks lies somewhere between $2^{n/2}$ and 2^n compression function evaluations, since no attack faster than exhaustive search is known.

Closing the Gap. It would be desirable to close this gap between attacks and proofs. But how? Recently, the *indifferentiability framework* has had some success in proving the absence of generic attacks, in particular by allowing to prove that hash function with an internal state of $2n$ bits resist all possible generic attacks up to $2^n/n$ queries [CN08]. Unfortunately, this proof technique cannot be used for our purpose. First of all, it cannot say anything about Merkle-Damgård: this venerable mode of operation is not even indistiguishable from a Random Oracle, because of the length extension attack. More fundamentally, indifferentiability can only be proved up to the birthday bound when the internal state is n-bit large, so that we cannot hope to obtain interesting results on Haifa. This follows from the fact that once a collisions on the compression function has been found, then it can often be used to build efficiently several pairs of colliding message by exploiting the iterated nature of the process, and thus allowing a distinguisher to tell apart the iteration from a random function. Obtaining provable security against second preimage adversaries therefore requires an *ad hoc* approach. In §7.1, we prove lower-bounds on the resistance of Merkle-Damgård and Haifa to second preimage attacks under the assumption that the compression function is a random function. More precisely, an adversary querying the compression function q times can find a second-preimage of an ℓ-block message on Merkle-Damgård (resp. Haifa) with probability $q\ell/2^n$ (resp. $2q/2^n$). This shows that the upper-bounds given above are tight, and that the known generic attacks are optimal.

Moving to the Standard Model. These results rely crucially on the assumption that the compression function is *random*. This shows that any attack beating the bounds of our proofs must exploit in a way or another a bias in the compression function, as it could be used to "distinguish" the compression function from a random function, inasmuch as this means anything. While this guarantees that the modes of operation are free of some bad flaw, does it mean that it will result in a secure hash function when an actual compression function will be plugged in? Results such as the collision-resistance preservation of Merkle-Damgård are stronger, as they result in an actual break of a well-defined security notion of the compression function when the full hash function is broken. Thus, as long as the compression function is secure, then the iteration is secure.

Are there modes of operation provably resistant to second preimage attacks beyond the birthday bound in the standard model? The answer is yes: the wide-pipe hash of Lucks achieves that (cf. §3.4). Are there non-wide-pipe modes of operation provably resistant to second preimage attacks in the standard model? The answer is again yes: UOWHFs are precisely *keyed* hash functions provably achieving second preimage resistance in the standard model. Shoup's UOWHF resists eSec adversaries up to $2^n/\ell$ queries when the compression function is as good as it gets. In addition, it is noteworthy that two other modes of operation

were presented in 2008, which obtain beyond-birthday second preimage security up to $2^n/\ell$ queries in the standard model: Yasuda's split-padding [Yas08] as well as Andreeva and Preneel's Three-property-secure hash function [AP08].

Unavoidable Security Loss. It seems that all narrow-pipe provably second-preimage resistant constructions only achieve security up to $2^n/\ell$ queries for messages of ℓ blocks, even when the compression function is ideal. On all these constructions, this follows from the fact a generic second preimage attack applies, be it either that of Kelsey and Schneier or our own. This is not completely the end of the story though, because these constructions could easily be patched to avoid generic second preimage attacks (by adding a round counter as in Haifa), yet it would not be possible to prove them secure above $2^n/\ell$ queries. We believe that this is caused by the proof technique itself: we show in §7.2 that any narrow-pipe mode of operation (belonging to a large and natural class) whose second-preimage resistance is reduced to the second-preimage resistance of the compression function cannot be provably secure beyond $\mathcal{O}(2^n/\ell)$ elementary operations (it *might* be more secure, but not *provably*). In other terms, there is an *unavoidable security loss* in second-preimage resistance proofs by black-box reductions.

Coping With Broken Compression Functions. As mentionned above, indifferentiability proofs are great because they guarantee the absence of generic attacks. In these proofs, the compression function is assumed to be ideal. What happens when the compression function turns out to have a flaw, or a specific property making it "obviously" different from a random function? For instance, what happens if there exist two differences Δ_i and Δ_o such that $f(x \oplus \Delta_i) = f(x) \oplus \Delta_o$ for all possible x's? Can the iteration of such a compression function be secure? Researchers exhibiting "distinguishers" (whatever this means) for the compression functions often argue that their findings voids the security proof of the full hash function. Following the ideas of the SHABAL team [BCCM+09] we show in §7.3 that this is not necessarily true, and that the Prefix-Free-Merkle-Damgård iteration of a non-ideal compression function can be indifferentiable from a Random Oracle. The security level assured by the proof degrades when the compression function deviates more and more from a random function.

In this whole chapter, we will assume that the maximal size of hashed messages, 2^κ blocks, is much smaller than $2^{n/2}$. This allows us to assume that all the intermediate chaining values are different with very high probability.

7.1 Second-Preimage Resistance in the Random Oracle Model

The proofs presented in this section assume that the compression function is an *ideal primitive* (*i.e.*, a fixed-input length Random Oracle), to which *computationally unbounded* adversaries have *oracle access*. The only obstacle they are facing is the randomness of the query responses. The number of queries sent to the primitive is then a meaningful complexity measure, because the adversary cannot obtain any kind of advantage by computation alone without querying the random function. In any case, it gives a lower-bound on the time complexity of the adversary. This setting is very similar to the analysis of block cipher-based constructions in the ideal cipher model by Black, Rogaway and Shrimpton [BRS02].

We usually denote by q the number of queries sent to the compression function f by an adversary $\mathcal{A}$. For the sake of convenience, we enforce second preimage adversaries not to abort, to always return a message M', even if they do not win the security game, and to evaluate $H^f(M')$ before terminating, by issuing the corresponding queries to the compression function. We also enforce adversaries not to ask the same query twice. We say that an adversary (q, ℓ, ε)-*breaks* a hash function H^f if the adversary is being challenged with an ℓ-block message, issues q queries to f and succeeds in finding a second preimage with probability ε.

The main idea common to all the proofs presented here is almost directly adapted from the existing generic second preimage attacks: we lower-bound the complexity of one particular step common to all these attacks, namely when some kind of a possible prefix has to be "connected" to the target message.

7.1.1 The Existing Attacks Against Merkle-Damgård are Optimal

We now prove that known generic second preimage attacks against Merkle-Damgård, whose complexity is of order $2^n/\ell$ for messages of length ℓ, are optimal.

Theorem 7.1. *Let f be a random compression function, H^f be the Merkle-Damgård iteration of f, and $\mathcal{A}$ be a second preimage adversary against H^f which (q, ℓ, ε)-break H^f. Then $\varepsilon \leq q \cdot \ell / 2^n$.*

Proof. Consider an adversary $\mathcal{A}$ that (q, ℓ, ε)-breaks the second preimage resistance of H^f. We denote by h_i, for $1 \leq i \leq \ell$, the chaining values obtained while hashing M, according to the description in Algorithm 3.1. If $\mathcal{A}$ succeeds in finding a second preimage, then in particular $\mathcal{A}$ has found a collision. As argued in §3.2.1,

in the presence of the Merkle-Damgård *strengthening*, this implies a collision on the compression function f. In our case, there exists an index i such that one of the colliding chaining value is h_i. This collision on f is therefore actually a second preimage of h_i for f. Note that because f is a random function, all the chaining values are random. [1]

Every time $\mathcal{A}$ submits a new query to the oracle, it receives a uniformly-distributed random value. The probability that $\mathcal{A}$ wins thanks to this particular query is upper-bounded by the probability that this random value is one of the h_i's, and this probability is exactly $\ell \cdot 2^{-n}$. Since $\mathcal{A}$ sends at most q queries, $\mathcal{A}$ wins with probability at most $q \cdot \ell \cdot 2^{-n}$. □

It must be noted that this proof is fairly general, because it reduces the problem of finding a second preimage for H^f to the problem of finding a second preimage of one out of many random chaining values for f. It actually covers nearly all the existing iterated hash functions.

7.1.2 The Round Counter Prevents Generic Second Preimage Attacks

The inventors of Haifa claimed that Haifa has optimal resistance against generic second preimage attacks but they did not prove it. The bound given by theorem 7.1 is also not strong enough to back up their claim. A slightly more involved proof technique is required to prove that Haifa achieves optimal second preimage resistance. The key ingredient of Haifa that makes it more secure than Merkle-Damgård is the round counter.

Theorem 7.2. *Let f be a random compression function, H^f be the* Haifa*-iteration of f, and $\mathcal{A}$ be a second preimage adversary that (q, ℓ, ε)-break H^f. Then $\varepsilon \leq q/2^{n-1}$.*

Proof. We simulate the execution of the adversary $\mathcal{A}$, and bookmark the queries sent by A to f: it is a set S of tuples (x, m, c, y), with $y = f(x, m, c)$. We suppose that $\mathcal{A}$ evaluates $H^f(M)$, so $\mathcal{A}$ sends the corresponding queries to the oracles at some point. Let us denote these particular queries $(h_{i-1}, m_i, c_i, h_i)_{1 \leq i \leq \ell}$. In particular, $H^f(M) = h_\ell$.

Suppose now that $\mathcal{A}$ wins. Then one of the two following situations arises:

1. If $\big|M\big| \neq \big|M'\big|$, then the values of the counter entering the compression function in its last invocation are different. Therefore, $\mathcal{A}$ has found a second preimage of h_ℓ on f. Each query has a probability 2^{-n} to realize this event, because f is a random function.
2. Otherwise, $\big|M\big| = \big|M'\big|$. This means that $\mathcal{A}$ has found a collision with M.

Because of the round-counter, collisions now have a special property.

Lemma 7.3. *Let H^f be the* Haifa *iteration of an arbitrary compression function f. If there are two messages M and M' such that $H^f(M) = H^f(M')$ with $M \neq M'$ and $\big|M\big| = \big|M'\big|$, then there is a collision on f, with the same value of the counter entering f.*

Proof. let us write the two (padded) messages $M = x_1, \ldots, x_r$ and $M' = x'_1, \ldots, x'_r$, the two initialization vectors $h_{-1} = h'_{-1} = IV$, and for $i \geq 0$, the chaining values $h_i = F(h_{i-1}, x_i, i)$ and $h'_i = f(h'_{i-1}, x'_i, i)$.

Since $h_r = h'_r$, either there is a collision on f (with counter value r), or $(x_r, h_{r-1}) = (x'_r, h'_{r-1})$. In the latter case, either there is a collision for f (with counter value $r-1$) or we again find that $(x_{r-1}, h_{r-2}) = (x'_{r-1}, h'_{r-2})$. This argument repeats. Since $\big|M\big| = \big|M'\big|$, then either there is a collision for f at some point (with the same counter value), or $x_i = x'_i$, for all i, $1 \leq i \leq r$. In the latter case, $M = M'$, which is impossible. This completes the proof of the lemma. □

This lemma means that if the adversary finds a second preimage of M, then there is in S a query (x, m, i_0, h_{i_0}) for a given value of i_0 (recall that h_{i_0} is the i_0-th chaining value obtained in the process of hashing M), and such that $(x, m) \neq (h_{i_0}, m_{i_0})$. We now upper-bound the probability that this event is realized.

When $\mathcal{A}$ submits its i-th query to the compression function, a random value is chosen and returned to $\mathcal{A}$. Our event is realized if and only if this value is h_i, and this happens with probability 2^{-n}. This query may also allow $\mathcal{A}$ to invert h_ℓ with probability 2^{-n}. Each query allows $\mathcal{A}$ to win with probability $2^{-(n-1)}$, and there are q queries, which completes the proof. □

1. We note that this claim is not necessarily true when the message is long and there are collision between the various chaining values. However, as this has a non-negligible probability only when $\ell \geq O(2^{n/2})$, we allow ourselves to disregard such very long messages.

7.2 Unavoidable Security Loss in Black-Box Reductions

Theorems 7.1 and 7.2 give lower-bounds on the complexity of *generic* attacks, *i.e.*, attacks that apply even when the attacker only has black-box access to the (ideal) compression function. Security against generic attacks is certainly a nice feature of modes of operations, but as we argued before it is not entirely satisfactory. In particular, it does not formally shows that there does not exist actual adversaries against the hash function. For instance, theorems 3.3 and 3.9 are stronger statements, as they show that any Turing Machine breaking a security property of the actual hash function can be transformed into another Turing machine that breaks a security property of the compression function. Thus, security of the iteration with respect to a well-defined security property relies on a well-defined security property of the compression function (as opposed to its "randomness"), hence the label "provable security in the standard model".

In this section, we will focus our attention on the Sec security notion, but our reasoning carries over to the other second-preimage resistance notions. In the standard model, time is the right measure of complexity. We will say that an adversary (t, ε)-break a (keyed) function F if given M and K, the adversary returns a message M' such that $M \neq M'$ and $F_K(M) = F_K(M')$ with probability ε. To compare such adversaries, the relevant measure is the ratio t/ε: it tells us how much time do we have to wait before actually seeing the adversary succeed.

We are aware of at least three narrow-pipe modes of operation that provably resist second preimage adversaries beyond the birthday bound in the standard model: we already know Shoup's UOWHF, but there are also Andreeva and Preneel's Three Property Secure Hash Function [AP08], and Yasuda's Split Padding [Yas08]. These three constructions use the same proof technique: they give an explicit black-box reduction $\mathcal{R}(\cdot, \cdot)$ that, given an adversary $\mathcal{A}_H$ against the full hash function and a compression function f, yields an adversary $\mathcal{R}(f, \mathcal{A}_H)$ against f. Thus, if the full hash function gets broken, the compression function gets broken as well. It is striking in the three modes of operation mentioned above, $\mathcal{R}(f, \mathcal{A}_H)$ only succeeds with probability $1/\ell$ (the inverse of the number of blocks of the challenge message), even when the adversary $\mathcal{A}_H$ always succeeds. The main result of this section is to show that this is not a coincidence, and that any such reduction cannot achieve a better success probability. Because this statement is very general, we begin by defining a whole class of modes of operations that we wish to take into account in out reasoning.

7.2.1 Abstract Narrow-Pipe Modes of Operations

We will consider that a *narrow-pipe mode of operation* $H^{(\cdot)}$ is a circuit that takes as its input M (the full message), K (the key, if present), h_i (the current chaining value) and i (the block counter). This circuit is responsible for preparing the input to the compression function. The next chaining value h_{i+1} is the output of the compression function on the input prepared by the circuit. The output of the whole hash function is the output of the compression function on its last invocation. The circuit activate a special wire "last call" to indicate that the hash process is terminated. We will assume that the number of calls to the compression function does not depend on the input of the hash function (*i.e.*, on M and K).

The incoming chaining value is set to a predefined value (say, zero) on the first invocation. This particular class of modes of operation imposes that the output of the full hash function comes out of the compression function without post-treatment, in particular without truncation. This, coupled with the fact that the circuit has no internal memory makes it a narrow-pipe mode of operation. Appart from that, H may include a "final transformation", or process each message block multiple times. Formally, the hash process works according to the pseudo-code shown in Algorithm 7.1.

Algorithm 7.1 Formal definition of the hash process with an abstract mode of operation

```
function Abstract-Mode-Of-Operation(M, K)
    h_{-1} ← 0
    i ← 0
    while not finished do
        x_i ← H^(·)(M, K, i, h_{i-1})
        h_i ← f(x_i)
        i ← i + 1
    end while
    return h_{i-1}
end function
```

There are constructions that are apparently not narrow-pipe, but that still fit in this framework, such as the GOST hash function (the checksum can be computed in the last invocation, and do not need to be transmitted between each invocation of the compression function).

Note that by choosing $H^{(\cdot)}$ to be a *circuit*, we implicitly admit the existence of an upper-bound on the size of the messages (if only because the block counter comes on a finite number of wires). In the sequel, by "mode of operation", we implicitly mean "a narrow-pipe mode of operation that fits the above framework". We are not aware of any narrow-pipe construction that does not fit the above definition though.

7.2.2 Collision-Resistance Preserving Modes of Operation

While we tried to make the definition of a mode of operation as generic as it gets, we are not interested in *really bad* modes of operation. We are not interested in non-collision resistant constructions, for instance. In this section, we characterize a few properties modes of operation should have not to be totally worthless.

We say that a mode of operation *is strengthened* if the binary encoding of the size of the processed message is contained in the input to the last invocation of the compression function. It is well-known that the Merkle-Damgård mode of operation is strengthened, which is the key in establishing its important collision-resistance preservation (theorem 3.3). However, in general, being strenghtened is not completely sufficient to be collision-resistance preserving. Some further technicalities are required.

We say that a mode of operation is *message-injective* if for all functions f and all keys K, the function that maps the message M to the sequence of compression-function inputs (x_i) is injective. This implies that hashing two different messages M and M' cannot generate the same sequence of inputs (x_i). This property is necessary for collision-resistance preservation: if H is not message-injective, there there exist a function f and a key K such that there exist two colliding messages M and M' generating the same hash, without causing a collision in the compression function.

We also say that a mode of operation is *chaining-value-injective* if for all f and all K, there exist a (deterministic...) function that maps x_i to h_{i-1}. The combination of these three properties is sufficient to ensure collision-resistance preservation

Lemma 7.4. *A mode of operation H simultaneously message-injective, chaining-value-injective and strengthened is collision-resistance preserving.*

This is essentially a more abstract version of theorem 3.3, and because the proof is awfully similar, we omit it. Because of lemma 7.4, we call a mode H "collision-resistance preserving" if it satisfies these three conditions.

7.2.3 Some particular Provably Second-Preimage Secure Modes of Operations

We briefly describe the three modes of operations mentionned in the introduction, and we show that they fit in out framework. In passing, we will argue that they satisfy two more properties: they *allow for embedding* and they are *suffix-clonable*.

Definition 7.1. A mode of operation $H^{(\cdot)}(K, M, i, h)$ allows for embedding if it is always possible (and computationally easy) to forge M and K such that the input of the i-th invocation of the compression function takes a chosen value (*i.e.*, if x_i's takes a specified value in Algorithm 7.1).

Definition 7.2. A mode of operation is suffix-clonable if given an $(\ell+1)$-block message M, a key K and an integer $0 \leq i \leq \ell$ it is always possible to find a different $(\ell+1)$-block message M' such that if $i > 0$:

$$H^{(\cdot)}(K, M, i-1, h_{i-2}) \neq H^{\cdot}(K, M', i-1, h_{i-2})$$

and always:

$$\forall j \in \{i, \ldots, \ell\}, \quad H^{(\cdot)}(K, M, j, h_{j-1}) = H^{(\cdot)}(K, M', j, h_{j-1})$$

Shoup's UOWHF. We have already described Shoup's UOWHF in §3.4.3. This construction promotes the eSec security of the compression function to that of the whole hash function. The key of the iterated hash function is logarithmic in the maximal size of the messages that can be hashed. In our framework, this construction is simple to describe:

1: **function** SHOUP-UOWHF(M, K, i, h_{i-1})
2: **let** $(k, \mu_0, \ldots, \mu_\kappa) \leftarrow K$
3: **let** $(m_0, \ldots, m_\ell) \leftarrow \mathbf{Pad}(M)$
4: **return** $\big(k, h_{i-1} \oplus \mu_{\nu_2(i)}, m_i\big)$
5: **end function**

It is *not* easy to see that Shoup's UOWHF allows for embedding, and we refer the reader to [Sho00a] (the key idea is that at all points in the iteration there is always a mask that has been used only once). In addition, this construction is also easily seen to be suffix-clonable: it suffices to leave K untouched an to modify the begining of M. The security of the construction is guaranteed by theorem 3.9.

The Backwards Chaining Mode. Andreeva and Preneel described in [AP08] the *Backwards chaining mode* which promotes the second-preimage resistance of an unkeyed compression function to the Sec security of the (keyed) iterated hash function. We will assume for the sake of simplicity that the message block and the chaining values have the same size, and in addition that a simplified padding scheme is applied: the last block is padded with zeroes, and the message length in bits is included in an extra block. The iteration is keyed, and the key is formed by a triplet (K_1, K_2, K_3) of n-bit strings (note that its size is independent of κ).

1: **function** BCM(M, K, i, h_{i-1})
2: **let** $(K_1, K_2, K_3) \leftarrow K$
3: **let** $(m_0, \ldots, m_\ell) \leftarrow$ **Pad**(M)
4: **if** $0 \leq i < \ell - 1$ **then return** $(h_{i-1} \oplus m_{i+1}, m_i)$
5: **if** $i = \ell - 1$ **then return** $(h_{\ell-2} \oplus m_\ell \oplus K_2, m_\ell \oplus K_1)$
6: **if** $i = \ell$ **then return** $(h_{\ell-1} \oplus K_3, m_\ell \oplus K_2)$
7: **end function**

The backwards chaining mode allows for embedding. To embed at any index smaller than $\ell - 1$, just choose m_i and m_{i+1} with care. To embed at index $\ell - 1$ or ℓ, pick the message at random and choose K_1, K_2 and/or K_3 accordingly (the keys are necessary to embed in the last blocks because of the padding scheme). Again, this construction is also easily seen to be suffix-clonable, with the same argument as before.

Theorem 7.5 ([AP08])**.** *If an adversary is able to break the Sec$[\ell]$ notion of H^f with probability ε in time T, then one can construct an adversary that breaks the Spr notion of f in time $T + \mathcal{O}(\ell)$, with probability ε/ℓ.*

The Split Padding. Yasuda's *Split Padding* [Yas08] is a minor tweak to the Merkle-Damgård-strenghtening. For the sake of simplicity, we will assume that the message block is twice bigger than the chaining values (*i.e.*, it is $2n$-bit wide). The tweak ensures that any message block going into the compression function contains at least n bits from the original message (this is not necessarily the case in the last block of the usual Merkle-Damgård padding scheme).

It promotes a kind of eSec security of the compression function to the Spr security of the (unkeyed) iteration. More precisely, the security notion required of the compression function is the following: the adversary chooses a chaining value h and the first n bits of the message block m_1, and is then challenged with the last n bits of the message block m_2. She has to find a new pair $(h', m') \neq (h, m_1 \,\|\, m_2)$ such that $f(h, m_1 \,\|\, m_2) = f(h', m')$. To some extent, this is the eSec security notion, but here the "key" of the compression function is the last n bits of the message block.

Theorem 7.6 ([Yas08])**.** *If an adversary is able to break the Spr$[\ell]$ notion of H^f with probability ε in time T, then one can construct an adversary that breaks the eSec-like notion of f in time $T + \mathcal{O}(\lambda)$, with probability ε/ℓ.*

The split-padding easily fits in our framework, because besides the padding scheme it is just the normal Merkle-Damgård mode.

1: **function** Split-padding(M, K, i, h_{i-1})
2: **let** $(m_0, \ldots, m_\ell) \leftarrow$ **Special-Pad**(M)
3: **return** (h_{i-1}, m_i)
4: **end function**

The split-padding does not allows for our previous definition of embedding, but it allows to embed n bits of message block into any block. It is easily suffix-clonable, as it suffices to change a prefix of the message.

7.2.4 Proofs by Reduction

Resistance against second preimage attacks in the standard model of a mode of operation $H^{(\cdot)}$ is often announced by theorem formulated similar to the following.

Theorem (typical)**.** *There exist a black-box reduction $\mathcal{R}(\cdot, \cdot)$ such that $\mathcal{R}(f, \mathcal{A}_H)$ is an adversary against the compression function f that $(t + t', \alpha \cdot \varepsilon + \beta)$-breaks f, for all compression functions f and all adversaries $\mathcal{A}_H$ that (t, ε)-break H^f.*

Theorems 3.9, 7.5 and 7.6 are clearly instances of this general setting. Note that the reduction is given *black-box* access to both the adversary and the compression function f. This is a way of formalizing that the reduction must work for any adversary and any compression function. For the sake of simplicity, we allow the reduction to issue *only one query* to the adversary. To some extent, this narrows our study a little, but all the reductions we are aware of (in [Sho00a, Yas08, AP08]) fit into this category.

In the setting of the security theorem above, there are three parties: the challenger, the reduction and the adversary. The challenger sends the reduction a challenge made of an input x to f, and a "key" k for f, whatever this means (chaining value or additional input). The reduction has to find a distinct input x' such that $f_k(x) = f_k(x')$. For this purpose, the reduction may use the $\mathcal{A}_H$ adversary: the reduction sends the adversary a challenge made of a message M of at most ℓ message blocks, and a key K. The adversary may either returns a message M' such that $H^f(K, M) = H^f(K, M')$ or fail. The precise sequence of interactions is the following:

$$\begin{array}{llll|ll} \text{Challenger} & \xrightarrow{x,k} & \text{Reduction} & & & \\ & & \text{Reduction} & \xrightarrow{M,K} & \text{Adversary} & \\ & & \text{Reduction} & \xleftarrow{M'} & \text{Adversary} & M \neq M' \quad H_K(M) = H_K(M') \\ \text{Challenger} & \xleftarrow{x'} & \text{Reduction} & & & x \neq x' \qquad f_k(x) = f_k(x') \end{array}$$

Reductions are generally assumed to have to simulate the legitimate input challenge distribution the adversary is normally expecting. In our case, this means that the distribution of the challenges M, K must be indistinguishable from random. Note that if M, K were biased, then the adversary could detect that it is "being used", and fail deterministically. In any case, when we mention the success probability ε of the adversary $\mathcal{A}_H$, we assume that its input distribution is uniformly random.

The interest of such reductions is that if the compression function f is secure, then for any adversary $\mathcal{A}_H$ against H^f, one must have: $(t + t')/(\alpha \cdot \varepsilon + \beta) \geq 2^n$, and therefore the global complexity of $\mathcal{A}_H$ is lower-bounded by:

$$\frac{t}{\varepsilon} \geq 2^n \alpha + \frac{2^n \beta - t'}{\varepsilon}. \tag{7.1}$$

The right-hand side of equation (7.1) is the *provable security level* that the reduction offers. Note that it bizarrely depends on the success probability of the adversary, but this seems unavoidable. When considering a single run of the reduction, its success probability should depends very much on whether the adversary succeeds or not. Therefore, it makes sense to write:

$$\mathbb{P}\left[\mathcal{R} \text{ succeeds}\right] = \varepsilon \cdot \mathbb{P}\left[\mathcal{R} \text{ succeeds} \;\middle|\; \mathcal{A}_H \text{ succeeds}\right] + (1 - \varepsilon) \cdot \mathbb{P}\left[\mathcal{R} \text{ succeeds} \;\middle|\; \mathcal{A}_H \text{ fails}\right]$$

This justifies why we assumed the success probability of the reduction to be of the form $\alpha \cdot \varepsilon + \beta$, and in fact we have:

$$\begin{aligned} \alpha &= \mathbb{P}\left[\mathcal{R} \text{ succeeds} \;\middle|\; \mathcal{A}_H \text{ succeeds}\right] - \mathbb{P}\left[\mathcal{R} \text{ succeeds} \;\middle|\; \mathcal{A}_H \text{ fails}\right] \\ \beta &= \mathbb{P}\left[\mathcal{R} \text{ succeeds} \;\middle|\; \mathcal{A}_H \text{ fails}\right] \end{aligned}$$

Now, while our objective is to understand what happens when $\mathcal{A}_H$ *succeeds*, it is easier to get a glimpse of what happens when $\mathcal{A}_H$ *fails*. In this setting, the reduction is just a randomized turing machine trying to find second preimage resistance on an arbitrary black-box function, which cannot be done faster than exhaustive search. For instance, f could be a Pseudo-Random Function with a randomly-chosen secret key. We could even use our Random Oracle Web Service to simulate a "truly" random function. In any case, it follows that $\beta \leq t'/2^n$. The provable security level offered by a reduction is thus upper-bounded by $\alpha \cdot 2^n$. We will thus say that a reduction is *useable* if $\alpha > t'/2^n$, as this implies that the reduction offers a provable security level better than that of exhaustive search (or equivalently, that the reduction actually makes use of the adversary).

7.2.5 How do Reductions Use the Adversary ?

In the sequel, we will make the natural assumption that the $\mathcal{A}_H$ adversary the reduction has access to has non-zero success probability. We will also restrict our attention to useable reductions. By doing so we rule out modes of operation for which no useable reduction is known such as the Merkle-Damgård construction.

Let us consider a provably secure mode mode of operation H also satisfying the hypotheses of lemma 7.4 (*i.e.*, injective, extractable and strengthened). We will also assume that it is suffix-clonable. Since the mode is provably secure, there exist a reduction $\mathcal{R}$ with a reasonably high success probability. Our objective, and the main technical contribution of this section, is to show the following theorem:

Theorem 7.7. *We always have $\alpha \leq 1/\ell + t'/2^n$. It follows that the provable security level offered by $\mathcal{R}$ cannot be higher than $2^n/\ell + t'$.*

The remaining of this section is devoted to the proof of this result. The general idea of the proof is to built an environnment around the reduction $\mathcal{R}$ that simulates a legitimate "world" for $\mathcal{R}$, but in which it is easy to see that $\mathcal{R}$ has a low success probability. Then because $\mathcal{R}$ has to work in all legitimate environments, it follows that in general $\mathcal{R}$ cannot succeed with higher probability.

Connection Point. Before going any further, let us observe what happens when the adversary finds a second preimage. Let us denote by x_i and h_i (resp. x'_i and h'_i) the inputs and the outputs of f while evaluating $H^f(M)$ (resp. $H^f(M')$). Since M and M' collide, and because H satisfies the hypotheses of lemma 7.4, then a second preimage of one of the x_i input values can be readily obtained from M'. Let us take a look back at the proof of theorem 3.3. If $\big|M\big| \neq \big|M'\big|$, then we obtain a second preimage of f on the last invocation. Otherwise, there exist an index i such that $f(x_i) = f(x'_i)$ and $x_i \neq x'_i$. In the sequel, we call this particular index i the "connection point", and we note that at this particular index a second preimage of x_i for f is revealed, which we call "the second preimage at connection point".

Embedding. The strategy used by all the reductions we are aware of consists in *embedding* the small challenge (x, k) into the big challenge (M, K). Following our definition, we say that (x, k) is *embedded into* (M, K) *at location* i if and only if $f_k(x)$ is evaluated during the i-th iteration of the main loop of Algorithm 7.1 during the evaluation of $H^k(K, M)$. We will show that *the second preimage returned by the adversary can only be used by the reduction if the second preimage at connection points directly gives a solution to the small challenge*. Let us denote by ♣ the condition "the second preimage at connection point is a second preimage of the small challenge sent by the Challenger to $\mathcal{R}$". Formally, this means that:

$$\mathbb{P}\left[\clubsuit\right] = \mathbb{P}\left[\Big(\exists i.\ \ x_i = (x,k) \text{ in Algorithm 7.1}\Big) \wedge (x_i \neq x'_i) \wedge (h_i = h'_i)\right]$$

We can then write:

$$\begin{aligned}\mathbb{P}\big[\mathcal{R} \text{ succeeds} \;\big|\; \mathcal{A} \text{ succeeds}\big] = \quad & \mathbb{P}\big[\mathcal{R} \text{ succeeds} \;\big|\; \mathcal{A} \text{ succeeds} \wedge \clubsuit\big] \cdot \mathbb{P}\left[\clubsuit\right] \\ + \quad & \mathbb{P}\big[\mathcal{R} \text{ succeeds} \;\big|\; \mathcal{A} \text{ succeeds} \wedge \neg\clubsuit\big] \cdot \mathbb{P}\left[\neg\clubsuit\right] \end{aligned} \tag{7.2}$$

We first argue that the challenge cannot be embedded more than once. If the challenge were embedded twice or more, the input distribution of the adversary would not be random, because we would have $x_i = x_j$ for $i \neq j$ in Algorithm 7.1, something that is highly unlikely when M and K are drawn at random. This is not allowed in the first place, and the adversaries could straightforwardly detect it and abort.

Next, we claim that in order to be usable, a reduction *must* embed the challenge (x, k) into (M, K). This is consistent with our observation that the three schemes of interest all allow some form of embedding. To establish this result, we show that a legitimate world with various interesting properties can be built around the reduction. When we argued that β is small, we used the (informal) argument that f could be implemented by a Random Function simulator similar to the "Random Oracle Web Service" of the introduction, and that inverting such a function faster than exhaustive search is impossible. We now make this argument more formal, with the additionnal feature that we will be able to choose whether the adversary succeeds or fails, and where it connects.

Simulation. The easy case is when we want $\mathcal{A}_H$ to fail, as it is sufficient to let f simulate an arbitrary random function, and let $\mathcal{A}_H$ return some random junk. The more interesting case is when we want $\mathcal{A}_H$ to succeed. The difficulty comes from the fact that the view of the reduction must be consistent: after having received M' from the $\mathcal{A}_H$, the reduction must be able to check that $H_K(M) = H_K(M')$ by querying f. This is in fact quite easy to achieve, by *programming* the function f. Here is how we simulate the execution of the reduction:

Algorithm 7.2 A dummy random function simulator

```
function f-Simulator(x, k)
    if Log[x, k] = ⊥ then Log[x, k] ←$ {0,1}^n
    return Log[x, k]
end function
```

1. Before $\mathcal{R}$ sends its query (M, K) to $\mathcal{A}_H$, we simulate f by generating random answers and storing them (for consistency), "implementing" f with Algorithm 7.2
2. When $\mathcal{R}$ sends its query (M, K) to $\mathcal{A}_H$, we choose a uniformly random integer $i \in \{0, \ldots, \ell\}$ (this will be the connection point), and we use the suffix-clonability property of the mode of operation to generate a different message $M' \neq M$ satisfying the conditions of definition 7.2.
3. We evaluate $H^f(M')$ in a special way. On the first $i-1$ iterations we use Algorithm 7.2 in place of f. On the i-th iteration we program f so that $f(x'_i) = h_i$, thus "connecting" M' to M in iteration i.
4. We return M' as the answer of $\mathcal{A}_H$ to the reduction, and keep simulating f.

We acknowledge that there is a negligible probability of failure in the third step, but it does not really matter. What matters is that the view of the reduction is consistent and legitimate. In this environment, we are able to choose the connection point at will. For instance, we can make sure that the ♣ event never happens. In this case, the reduction, even though it knows a collision on f, cannot find a second preimage on f faster than exhaustive search (because each new query to f returns an independent random answer, and thus each query yields a second preimage with probability 2^{-n}).

It follows that if a reduction does not embed its challenge, then it cannot be usable. We conclude that a usable reduction must embed its challenge exactly once with non-zero probability. As a matter of fact, the reductions of the three schemes considered in the introduction published in the litterature embed their challenge with probability one. Equation (7.2) then gives:

$$\mathbb{P}\left[\mathcal{R} \text{ succeeds} \mid \mathcal{A} \text{ succeeds}\right] \leq \mathbb{P}\left[\mathcal{R} \text{ succeeds} \mid \mathcal{A} \text{ succeeds} \wedge \clubsuit\right] \cdot \mathbb{P}\left[\clubsuit\right] + \frac{t'}{2^n} \tag{7.3}$$

Now, to prove theorem 7.7, we upper-bound the probability that the ♣ condition occur. The reduction cannot control "where" the adversary will "connect" to the big challenge M. Conversely, if the adversary could guess where the challenge is embedded, then she could systematically refuse to connect precisely there. In fact, we need not even worry about this complication, since the adversary can foil all the reduction's plan by connecting randomly. In our simulation procedure, if we choose the connection point uniformly at random between 0 and ℓ, then the ♣ event only happens with probability $1/\ell$. Combining this with equation (7.3) yields:

$$\mathbb{P}\left[\mathcal{R} \text{ succeeds} \mid \mathcal{A}_H \text{ succeeds}\right] \leq \frac{1}{\ell} + \frac{t'}{2^n}$$

And this is exactly what we needed to complete the proof of theorem 7.7. We conclude by pondering on this intringuing situation, where some narrow-pipe modes of operations are provably resistant to generic second preimage attacks, yet this cannot be shown in the standard model...

7.3 Indifferentiability in the Presence of Distinguishers

In this final section, we discuss the security of the prefix-free Merkle-Damgård iteration of non-ideal compression functions. While our primary objective was to show that the distinguisher found by Leurent [BFL10] for the compression function of SIMD [LBF08] did not void the security proof of SIMD, the reasoning and the proof presented here are pretty general and could very well be adapted to other functions.

Here, $\mathcal{H} = \{0,1\}^n$ still denotes the set of chaining values, $\mathcal{M} = \{0,1\}^m$ still denotes the set of message blocks, and $\mathcal{F}$ denotes the set of all functions $\mathcal{H} \times \mathcal{M} \to \mathcal{H}$. Let $f \in \mathcal{F}$ be a compression function taking as input an n-bit chaining value and an m-bit message block.

The Prefix-Free-Merkle-Damgård mode has been shown to be indifferentiable from a random oracle up to the birthday bound by Coron, Dodis, Malinaud and Puniya [CDMP05]. This is done by constructing a simulator $\mathcal{S}$ such that any distinguisher $\mathcal{D}$ cannot tell apart (H^f, f) and $(RO, \mathcal{S})$ without a considerable effort, where RO is a variable-input-length random oracle (VIL-RO, for short). Informally, if f is ideal, then H^f is (supposedly) secure up to the level offered by the indifferentiability proof. More precisely, if $H^{(\cdot)}$ is $(t_\mathcal{D}, t_\mathcal{S}, q_S, q_0, \varepsilon)$-indifferentiable from a VIL-RO when the compression function is assumed to be a Fixed Input Length Random Oracle (FIL-RO), then this means that there exists a simulator running in time $t_\mathcal{S}$, such that any distinguisher running in time $t_\mathcal{D}$ and issuing at most q_S (resp. q_0) queries to the FIL-RO (resp. VIL-RO) has success probability at most ε.

A drawback of this methodology is that it is not very "failure-friendly", because the security argument offered by the indifferentiability proof becomes vacuous as soon as the compression function used in a hash function "turns out" to be non-ideal. For instance, distinguishers exhibiting a "non-random" behavior of the compression function are usually advertised by their authors to nullify the security proof of the full hash function.

This problematic situation was first tackled by the designers of Shabal, who provided a security proof taking into account the existence of an efficient distinguisher on the internal permutation of their proposal [BCCM+09]. We will follow their track and demonstrate that the security of Prefix-Free Merkle-Damgård can be proved despite the existence of an efficient distinguisher on its compression function.

Let H^f therefore denote the *prefix-free*-Merkle-Damgård iteration of f. Formally, the function $g : \{0,1\}^* \to \mathcal{M}^*$ is a *prefix-free encoding* if for all x, x', $g(x)$ is not a prefix of $g(x')$. The mode of operation $H^{(\cdot)}$ simply applies the Merkle-Damgård iteration of f to the prefix-free encoding of the message.

Theorem 7.8 (ideal case, [CDMP05]). *The* Prefix-Free-*Merkle-Damgård mode of operation is* $(t_\mathcal{D}, t_\mathcal{S}, q_S, q_O, \varepsilon)$-*indifferentiable from a VIL-RO when the compression function is modeled by a FIL-RO, for any running time* $t_\mathcal{D}$ *of the distinguisher, and* $t_\mathcal{S} = \mathcal{O}\left((q_O + \ell \cdot q_S)^2\right)$ *where* ℓ *is an upper-bound on the size of the queries*

sent to the VIL-RO. If $q = q_S + 2^\kappa \cdot q_O + 1$, where 2^κ is again an upper-bound on the size (in block) of hashed message, then the success probability of the distinguisher is upper-bounded by:

$$\varepsilon = 8 \cdot \frac{q^2}{2^n}$$

To restore the security argument damaged by the existence of distinguishers, we will show that the prefix-free iteration of a non-ideal compression function is to some extent still indifferentiable from a VIL-RO.

7.3.1 Deterministic Distinguishers for the Compression Function

Let us consider a non-ideal compression function f.

- For instance, it may have *weak states*, that are such that querying f thereon with a well-chosen message block produces a "special" output allowing to distinguish f from random in one query. Known examples include for instance the symmetry on the compression function of Lesamnta [BDFL10] and SIMD [BFL10], as well as on the internal permutation of CubeHash [ABM+09, FLM10].
- But f can also have *bad second-order properties*, meaning that the output of f on correlated input states (with well-chosen message blocks) produces correlated outputs, allowing to distinguish f from random in two queries. A notable example of this property include the existence of differential paths with probability one in the compression function of Shabal [AMM09]. Symmetry properties also give second order relations, which means that Lesamnta, CubeHash and SIMD have bad second-order properties as well.

Relational Constraints. Following the methodology introduced in [BCCM+09], we model this situation by saying that there are two relations $\mathcal{R}_1$ and $\mathcal{R}_2$ such that:

$$\forall (h, m) \in \mathcal{H} \times \mathcal{M} : \qquad \mathcal{R}_1(h, m, f(h, m)) = 1$$
$$\forall (h_1, h_2, m_1, m_2) \in \mathcal{H}^2 \times \mathcal{M}^2 : \qquad \mathcal{R}_2(h_1, m_1, h_2, m_2, f(h_1, m_1), f(h_2, m_2)) = 1$$

We denote by $\mathcal{R}$ the relation formed by the union of $\mathcal{R}_1$ and $\mathcal{R}_2$, and we will denote by $\mathcal{F}[\mathcal{R}]$ the subset of $\mathcal{F}$ such that the above two equations hold. We require the relations to be efficiently checkable, *i.e.*, that given h, m and h', it is efficient to check whether $\mathcal{R}_1(h, m, h') = 1$. The relation can thus be used as an efficient distinguishing algorithm that tells $\mathcal{F}[\mathcal{R}]$ apart from $\mathcal{F}$.

A *weak state* is a state on which it is possible to falsify the relation $\mathcal{R}_1$. We formally define the set of weak states for $\mathcal{R}_1$ in the following way:

$$\mathcal{W} = \{h \in \mathcal{H} \mid \exists m, h' \in \mathcal{M} \times \mathcal{H} \text{ such that } \mathcal{R}_1(h, m, h') = 0\}$$

The set of weak states $\mathcal{W}$ should be a relatively small subset of $\mathcal{H}$ because the loss of security will be related to the size of $\mathcal{W}$. Moreover, we require that the IV is not in $\mathcal{W}$.

In the same vein, a *weak pair* is a pair of states on which it is possible to falsify the relation $\mathcal{R}_2$. We therefore define the set of *weak pairs* for $\mathcal{R}_2$ by an undirected graph $G_{\mathcal{R}_2} = (\mathcal{H}, \mathcal{WP})$, where the set of edges $\mathcal{WP}$ is defined by:

$$\mathcal{WP} = \left\{h_1 \leftrightarrow h_2 \mid \exists m_1, m_2, h'_1, h'_2 \in \mathcal{M}^2 \times \mathcal{H}^2 \text{ such that } \mathcal{R}_2(h_1, m_1, h_2, m_2, h'_1, h'_2) = 0\right\}$$

Similarly, $\mathcal{WP}$ should be a relatively small subset of $\mathcal{H}^2$ because the security loss will be related to the size of $\mathcal{WP}$. For the sake of expressing things conveniently, we define a variant of the same graph, $G'_{\mathcal{R}_2} = (\mathcal{H} \times \mathcal{M}, \mathcal{WP}')$, where $\mathcal{WP}'$ is defined by:

$$\mathcal{WP}' = \left\{(h_1, m_1) \leftrightarrow (h_2, m_2) \mid \exists h'_1, h'_2 \in \mathcal{H}^2 \text{ such that } \mathcal{R}_2(h_1, m_1, h_2, m_2, h'_1, h'_2) = 0\right\}$$

To simplify the proof, we also require that *the connected component of $G'_{\mathcal{R}_2}$ have size at most two.* This rules out some second-order relations, but it includes for instance the existence of a differential path with probability one with a non-zero difference in the input chaining value, as well as the symmetry in the compression function of SIMD or Lesamnta. This restriction is somewhat artificial and could likely be lifted, but there will be a loss of security related to the square of the size of connected components.

We also require the existence of *sampling algorithms* for $\mathcal{R}$, namely of two efficient algorithms $\textsc{Sampler}_1$ and $\textsc{Sampler}_2$ such that:

$$\textsc{Sampler}_1(h, m) \quad : \quad f \xleftarrow{\$} \mathcal{F}[\mathcal{R}]; \quad \textbf{return } f(h, m)$$
$$\textsc{Sampler}_2(h_1, m_1, h_2, m_2, h'_1) \quad : \quad f \xleftarrow{\$} \{f \in \mathcal{F}[\mathcal{R}] : f(h_1, m_1) = h'_1\}; \quad \textbf{return } f(h_2, m_2)$$

Informally, the sampling algorithms should produce an output that looks as if it were produced by a random function constrained to conform to $\mathcal{R}$. There are several ways to measure the constraint put on f by $\mathcal{R}$. Examples include:

1. The advantage of $\mathcal{R}$ to distinguish $\mathcal{F}[\mathcal{R}]$ from $\mathcal{F}$
2. The ratio between the cardinality of $\mathcal{F}[\mathcal{R}]$ and that of $\mathcal{F}$
3. The advantage with which the output of the samplers can be predicted
4. $|\mathcal{W}|$ and $|\mathcal{WP}|$.

All these quantities are related, and in this section, we will use the cardinality of $\mathcal{W}$ and $\mathcal{WP}$, as it appears naturally in our results.

7.3.2 Adapting the Indifferentiability Proof to Non-Ideal Compression Functions

We now assume that the compression function is a public function chosen uniformly at random in $\mathcal{F}[\mathcal{R}]$, and for the sake of convenience we will call it a "biased FIL-RO". We show that the prefix-free iteration of biased FIL-RO is indifferentiable from a VIL-RO. In fact, we extend Theorem 7.8 to the case where the compression function is biased.

Theorem 7.9 (biased case)**.** PREFIX-FREE-*Merkle-Damgård is* $(t_{\mathcal{D}}, t_{\mathcal{S}}, q_S, q_O, \varepsilon)$*-indifferentiable from a VIL-RO, when the compression function is modeled by a biased FIL-RO conforming to the relation* $\mathcal{R}$*, for any running time* $t_{\mathcal{D}}$ *of the distinguisher, and* $t_{\mathcal{S}} = \mathcal{O}\left((q_O + \ell \cdot q_S)^2\right)$ *where* ℓ *is an upper-bound on the size of the queries sent to the VIL-RO. If* $q = q_S + 2^\kappa \cdot q_o + 1$*, where* 2^κ *is an upper-bound on the size of hashed messages, then the probability of success of the distinguisher is upper-bounded by:*

$$\varepsilon = 16 \cdot \frac{q^2}{2^n} + 4 \cdot |\mathcal{W}| \cdot \frac{q}{2^n,} + 4 \cdot |\mathcal{WP}| \cdot \frac{q^2}{\left(2^n - q\right)^2}$$

The first term of the expression of ε is similar to the result given in Theorem 7.8, when the compression function is ideal (up to a factor two that could be avoided by making the argument slightly more involved). The two other terms reflect the fact that the compression function is biased. The relation induces a loss in provable security if $|\mathcal{W}|$ is at least of order $2^{n/2}$, or if $|\mathcal{WP}|$ is at least of order 2^n. Informally, it seems possible to iterate compression functions having a relatively high bias in a secure way.

Free-start Differential Attacks. Let us assume that the compression function is weak because of the existence of a good differential path with a non-zero difference in the input chaining value. Even if the probability of the differential path is 1, this has a very limited effect on the security of the hash function: this leads to $\mathcal{W} = \emptyset$ and $|\mathcal{WP}| = 2^{n-1}$. The advantage of the distinguisher is at most twice as high, compared to the iteration of an ideal FIL-RO.

SIMD. In SIMD-256 (resp. SIMD-512), the internal state has $n = 512$ bits (resp. $n = 1024$ bits), and the distinguisher given in [BFL10] yields $|\mathcal{W}| = 2^{n/2+16}$, $|\mathcal{WP}| = 2^{n+32}$ (resp. $|\mathcal{W}| = 2^{n/2+32}$, $|\mathcal{WP}| = 2^{n+64}$). Therefore the advantage of any distinguisher in telling apart SIMD-256 from a VIL-RO with q queries is upper-bounded by:

$$\varepsilon = 16 \cdot \frac{q^2}{2^n} + 4 \cdot \frac{2^{n/2+16} \cdot q}{2^n} + 4 \cdot 2^{n+32} \cdot \frac{q^2}{\left(2^n - q\right)^2}$$

The mode of operation of SIMD-256 is then secure against generic attacks up to roughly 2^{256-16} queries, even if the attacker is able to exploit the symmetries of the compression function. Similarly, for SIMD-512, the bound goes up to 2^{512-32} queries.

Lesamnta. Lesamnta uses the prefix-free Merkle-Damgård mode of operation due to its special finalization function. An efficient distinguisher based on symmetries was shown in [BDFL10], with $|\mathcal{W}| = 2^{n/2}$ and $|\mathcal{WP}| = 2^{n-1}$. According to Theorem 7.9, the advantage of any distinguisher in telling apart Lesamnta-256 from a random oracle with q queries is upper-bounded by:

$$\varepsilon = 16 \cdot \frac{q^2}{2^n} + 4 \cdot \frac{2^{n/2} \cdot q}{2^n} + 4 \cdot 2^{n-1} \cdot \frac{q^2}{\left(2^n - q\right)^2} \approx 22 \cdot \frac{q}{2^{n/2}}$$

Our result shows that Lesamnta remains secure against generic attacks up to the birthday bound. This is the best achievable proof for Lesamnta, since it does not behave as a good narrow-pipe hash function beyond that bound: a dedicated herding attack based on the symmetry property is shown in [BDFL10], with complexity $2^{n/2}$.

To establish theorem 7.9, we will proceed in two steps. We will first give a proof of theorem 7.8 (dealing with the case where the compression function is ideal). Only then we will give a proof of theorem 7.9, by "patching" the proof of the ideal case.

7.3.3 In the Ideal Case : Proof of Theorem 7.8

We now show that the prefix-free iteration of an ideal compression function is indifferentiable from a random oracle, thus proving Theorem 7.8. The content of this section borrows very much to the proof in the extended version of [CDMP05].

We consider a *simulator* $\mathcal{S}$, which has oracle access to a random oracle $RO : \{0,1\}^* \to \{0,1\}^n$, and whose task is to simulate a random compression function. The pseudo-code of the simulator is shown in Algorithm 7.3, and here are a few comments. The simulator maintains a log of the queries it has answered to. This knowledge is maintained under the form of a graph $G = (V, E)$, where the set of vertices V is a subset of $\mathcal{H}$, and where the edges are labelled by message blocks from $\mathcal{M}$. The meaning of this graph is that there is an edge labelled by m between h and h' if the simulator lets the distinguisher know that $f(h, m) = h'$. We denote this by $h \xrightarrow{m} h'$. Initially, the graph contains only a single vertex IV. The simulator also maintains a subset of V denoted by Reach, consisting of the vertices that are reachable from IV. It also associates to each vertex v in Reach an *ancestor* in Reach. This allows, given v, to efficiently reconstruct the sequence of message blocks that maps IV to v. We will note $IV \xrightarrow[*]{M} v$ when there is such a path between IV and v. At the beginning, Reach only contains the IV.

Algorithm 7.3 Pseudo-code of the Simulator $\mathcal{S}$

```
1:  function FreshValue(h, m)
2:      if IV -M/*-> h ∈ Reach then
3:          if there exist M' such that M||m = g(M') then
4:              h' ← RO(M')
5:          else
6:              h' <-$ H
7:          end if
8:          Reach ← Reach ∪ {h -m-> h'}
9:      else
10:         h' <-$ H
11:     end if
12:     V ← V ∪ {h, h'}
13:     E ← E ∪ {h -m-> h'}
14:     return h'
15: end function

16: function Simulator(h, m)
17:     if there exist a vertex h' ∈ V and an edge h -m-> h' in E then
18:         return this h'
19:     else
20:         h' ← FreshValue(h, m)
21:         return h'
22:     end if
23: end function
```

Let f be a random compression function. Now, a *distinguisher* $\mathcal{D}$ interacts with either H^f and f (we say that it is in the "construction world"), or with RO and $\mathcal{S}$ (and we say that it is in the "random oracle world"), and it has to tell in which world it is. More formally, $\mathcal{D}$ is a Turing machine that has two interfaces. It should output "1" if H^f and f are answering its oracle queries, and "0" if RO and $\mathcal{S}$ are. Our objective is to show that the following holds for a small ε:

$$\left| \mathbb{P}\left[\mathcal{D}^{H^f, f} = 1\right] - \mathbb{P}\left[\mathcal{D}^{RO, \mathcal{S}} = 1\right] \right| \leq \varepsilon$$

The proof uses a hybrid argument through a sequence of games. We will denote by q_S and q_O the number of queries sent to the Simulator and the Oracle respectively, by the distinguisher.

Game 1: The distinguisher is in the random oracle world. It has access to RO and $\mathcal{S}$. Let G_1 be the event that $\mathcal{D}$ outputs "1" in this setting:

$$\mathbb{P}\left[G_1\right] = \mathbb{P}\left[\mathcal{D}^{RO, \mathcal{S}} = 1\right]$$

Game 2: We introduce a dummy *relay algorithm* $\mathcal{T}$, which has oracle access to RO. Given a random oracle query from the distinguisher, $\mathcal{T}$ just send the query to RO, and transmits the answer of RO back to $\mathcal{D}$.

Let G_2 be the event that $\mathcal{D}$ outputs "1" in this case. Since the view of $\mathcal{D}$ is left unchanged, we have:

$$\mathbb{P}[G_2] = \mathbb{P}\left[\mathcal{D}^{\mathcal{T}^{RO},\mathcal{S}} = 1\right] = \mathbb{P}[G_1]$$

Game 3: In this game, we modify the simulator $\mathcal{S}$. In particular, we restrict the responses of the simulator such that they never satisfy certain specific failure conditions. If the simulator comes up with a response that would result in an inconsistent state, then it fails explicitly instead of sending that response. The failure conditions describe certain situations that could be exploited by the distinguisher, such as collisions on the internal state. We replace the FreshValue function by the one shown in Algorithm 7.4.

Algorithm 7.4 Patched simulator for Game 3

```
1: function FreshValue(h, m)
2:     if IV -M->* h ∈ Reach then
3:         if there exist M' such that M‖m = g(M') then
4:             h' ← RO(M')
5:         else
6:             h' <-$ H
7:         end if
8:         if h' ∈ V then
9:             Abort
10:        end if
11:        Reach ← Reach ∪ {h -m-> h'}
12:    else
13:        h' <-$ H
14:    end if
15:    V ← V ∪ {h, h'}
16:    E ← E ∪ {h -m-> h'}
17:    return h'
18: end function
```

It should be clear that until no abort occur, the subgraph Reach is in fact a tree rooted in IV. This follows from the fact that the simulator aborts as soon as a collision in the internal state is detected. The new value h' is always drawn uniformly at random. It should be clear that as long as the simulator does not abort, the number of nodes in V is upper-bounded by $2q_S + 1$.

Therefore, for a given query, the probability of failure is upper-bounded by $(2q_S + 1)/2^n$. For all the q_S queries sent by the distinguisher to the simulator, the probability of failure is therefore less than $q_S \cdot (2 \cdot q_s + 1)/2^n$. Let G_3 be the event that $\mathcal{D}$ outputs "1" in this case. Since the view of $\mathcal{D}$ only changes when the simulator aborts, we have:

$$|\mathbb{P}[G_3] - \mathbb{P}[G_2]| \leq 2 \cdot \frac{(q_S + 1)^2}{2^n}$$

In the sequel, we denote the patched simulator by $\mathcal{S}_0$.

Game 4: In this game, we modify the relay algorithm and leave the simulator unchanged. The underlying idea is to make the responses of the relay algorithm directly dependent on the simulator. Thus, instead of giving the new relay algorithm $\mathcal{T}_1$ an oracle access to the random oracle RO, it is now given oracle access to the simulator $\mathcal{S}_0$. On a random oracle query X, the relay algorithm $\mathcal{T}_1$ computes $g(X)$, the prefix-free encoding of X. It then applies the Merkle-Damgård construction to $g(X)$ and queries the simulator $\mathcal{S}_0$ to evaluate the compression function. Thus the relay algorithm $\mathcal{T}_1$ is essentially the same as the random oracle construction pf-MD, except that it is based on the simulator $\mathcal{S}_0$ instead of random function f. Let G_4 denote the event that the distinguisher $\mathcal{D}$ outputs "1" when given oracle access to $\mathcal{T}_1$ and $\mathcal{S}_0$ in this game. Thus, we know that

$$\mathbb{P}[G_4] = \mathbb{P}\left[\mathcal{D}^{\mathcal{T}_1,\mathcal{S}_0} = 1\right]$$

Before going further, we establish two key properties of $\mathcal{S}_0$. Let us consider the sequence $\mathcal{Q}$ of queries (h_i, m_i, h'_i) sent to $\mathcal{S}_0$, where h'_i is the answer and (h_i, m_i) is the question. We say that the IV is *reachable*, and at a given point in the simulation h'_i is reachable if there has been a previous query (h_i, m_i, h'_i) where h_i was reachable. Then:

(*i*) Until $\mathcal{S}_0$ fails, Reach precisely describes the set of reachable chaining values.

(*ii*) Until $\mathcal{S}_0$ fails, Reach describes a tree.

These two properties are easy to establish by induction on the number of queries. When the simulator detects that h_i is reachable, it puts its answer h'_i in Reach. What guarantees that our two properties hold is that $\mathcal{S}_0$ aborts if h'_i was already "known". Thus, the set of reachable values can only be extended by one element, namely h'_i, and Reach is updated accordingly.

Next, we claim that the following three statements hold:

(*i*) In Game 3, *i.e.*, when $\mathcal{D}$ interacts with $\left(\mathcal{T}^{RO}, \mathcal{S}_0\right)$, the answers of $\mathcal{S}_0$ are consistent with those of RO as long as $\mathcal{S}_0$ does not abort.

(*ii*) In Game 4, *i.e.*, when $\mathcal{D}$ interacts with $\left(\mathcal{T}_1^{pf-MD(\mathcal{S}_0)}, \mathcal{S}_0\right)$, the answers of $\mathcal{S}_0$ are consistent with those of RO as long as $\mathcal{S}_0$ does not abort.

(*iii*) $\mathcal{T}^{RO}$ and $\mathcal{T}_1^{pf-MD(\mathcal{S}_0)}$ give the same answers until the simulator aborts.

From these three points, we can deduce that the view of the distinguisher $\mathcal{D}$ remains unchanged from game 3 to game 4 if the simulator $\mathcal{S}_0$ does not fail in either of the two games.

Proof. (*i*) To detect an inconsistency between $\mathcal{S}_0$ and RO, one has to build a chain of queries corresponding to a valid message, and compare with the output of RO with the last query of the chain. Note that if the chain is built out-of-order, then the simulator will abort. Therefore the last query to be sent to $\mathcal{S}_0$ is the final block of the prefix free encoding of M. When $\mathcal{S}_0$ detects the final block of a message, it queries RO on the decoded message, which is unique because Reach is a tree. The answers of RO and $\mathcal{S}_0$ are thus consistent.

(*ii*) The justification is the same as in the previous point. The fact that $\mathcal{T}_1$ sends extra queries does not change the fact that $\mathcal{S}_0$ answers are consistent with the Random Oracle.

(*iii*) Since $\mathcal{S}_0$ is consistent with the VIL-RO, the relay algorithm $\mathcal{T}_1$ does in fact return $RO(M)$ by applying the pf-MD construction with $\mathcal{S}_0$.

□

We can finally complete the argument by observing that if the maximum length of the prefix-free encoding of a random oracle query made by $\mathcal{D}$ is ℓ blocks, then,

$$\begin{aligned} |\mathbb{P}[G_4] - \mathbb{P}[G_3]| &\leq \mathbb{P}[\mathcal{S}_0 \text{ fails in Game 3}] + \mathbb{P}[\mathcal{S}_0 \text{ fails in Game 4}] \\ &\leq 2 \cdot \frac{(q_S+1)^2 + (q_S + \ell \cdot q_O + 1)^2}{2^n} \end{aligned}$$

Game 5: In this game, we modify the simulator $\mathcal{S}_0$ so as to make its responses independent of the random oracle RO. For this purpose, we remove the random oracle RO from this game entirely and the new simulator $\mathcal{S}_1$ always chooses a random n-bit response itself, even in situations where $\mathcal{S}_0$ would have consulted RO. We also remove all the failure conditions from the new simulator $\mathcal{S}_1$. More precisely, we replace the FreshValue function by the one in Algorithm 7.5

Algorithm 7.5 Patched simulator for Game 5

```
1: function FreshValue(h, m)
2:     h' ←$ H
3:     V ← V ∪ {h, h'}
4:     E ← E ∪ {h —m→ h'}
5:     return h'
6: end function
```

The responses of these two simulators are identical except from the failure conditions which are used by $\mathcal{S}_0$ and not by $\mathcal{S}_1$: even when $\mathcal{S}_0$ consults the random oracle, its responses are still uniformly distributed. Thus, the distinguisher does not notice a difference between these games if in game 4, the simulator $\mathcal{S}_0$ does not fail.

Let G_5 denote the event that the distinguisher $\mathcal{D}$ outputs "1" in game 5, so that

$$\mathbb{P}[G_5] = \mathbb{P}\left[\mathcal{D}^{\mathcal{T}_1, \mathcal{S}_1} = 1\right]$$

Then we can deduce that:

$$\begin{aligned} |\mathbb{P}[G_5] - \mathbb{P}[G_4]| &\leq \mathbb{P}[\mathcal{S}_0 \text{ fails in game 4}] \\ &\leq 2 \cdot \frac{(q_S + \ell \cdot q_O + 1)^2}{2^n} \end{aligned}$$

Game 6: This is the final game of our argument. Here we finally replace the simulator $\mathcal{S}_1$ with the random function f. Since the relay algorithm $\mathcal{T}_1$ simply implemented the prefix-free Merkle-Damgård construction, then the view of the distinguisher is in fact the construction world. Now, by combining games 1 to 6, we can show that

$$\left|\mathbb{P}\left[\mathcal{D}^{H^f,f}=1\right]-\mathbb{P}\left[\mathcal{D}^{RO,\mathcal{S}}=1\right]\right| \leq 4\cdot\frac{(q_S+1)^2+(q_S+\ell\cdot q_O+1)^2}{2^n}$$

7.3.4 Non-Ideal Case: Proof of Theorem 7.9

Algorithm 7.6 Pseudo-code of the Simulator $\mathcal{S}_0$ for the non-ideal case, with abort conditions

1: **function** SIMULATOR(h, m)
2: **if** there exist a vertex $h' \in V$ and an edge $h \xrightarrow{m} h'$ in E **then**
3: **return** this h'
4: **else**
5: **return** FRESHVALUE(h, m)
6: **end if**
7: **end function**

8: **function** FRESHVALUE(h, m)
9: **if** there exist $(u, v) \leftrightarrow (h, m) \in G'_{\mathcal{R}_2}$ **then** $(\overline{h}, \overline{m}) \leftarrow (u, v)$
10: **if** $IV \xrightarrow[*]{M} \overline{h} \in \mathsf{Reach}$ **then** Swap (h, m) and $(\overline{h}, \overline{m})$ ▷ (only if $\overline{h}$ is defined)
11: **if** $IV \xrightarrow[*]{M} h \in \mathsf{Reach}$ **then**
12: **if** there exist M' such that $M\|m = g(M')$ **then**
13: $h' \leftarrow RO(M')$
14: **else**
15: $h' \xleftarrow{\$}$ HEIGHT
16: **end if**
17: $\overline{h'} \leftarrow \text{SAMPLER}_2\left(h, m, \overline{h}, \overline{m}, h'\right)$ ▷ (only if $\overline{h}$ is defined)
18: **if** $h' \in \mathcal{W}$ **or** $h' \in V$ **or** $\mathsf{Reach} \cup \{h'\}$ covers an edge of $G_{\mathcal{R}_2}$ **then Abort**
19: $\mathsf{Reach} \leftarrow \mathsf{Reach} \cup \left\{h \xrightarrow{m} h'\right\}$
20: **else**
21: $h' \leftarrow \text{SAMPLER}_1(h, m)$
22: $\overline{h}' \leftarrow \text{SAMPLER}_2\left(h, m, \overline{h}, \overline{m}, h'\right)$ ▷ (only if $\overline{h}$ is defined)
23: **end if**
24: $V \leftarrow V \cup \left\{h, h', \overline{h}, \overline{h'}\right\}$ ▷ (only add $\overline{h}$ and $\overline{h'}$ if defined)
25: $E \leftarrow E \cup \left\{h \xrightarrow{m} h', \overline{h} \xrightarrow{\overline{m}} \overline{h'}\right\}$ ▷ idem.
26: **return** h' (or $\overline{h'}$ if they were swapped in line 10)
27: **end function**

The proof proceed in the same way as the proof of Theorem 7.8. The simulator is shown in Figure 7.6. The pseudo-code shows $\mathcal{S}_0$ with the failure conditions. Before going further, a few comments on $\mathcal{S}_0$ are in order. When it receives a query (h, m), it checks whether there exist a *symmetric query* $\left(\overline{h}, \overline{m}\right)$, that would trigger the symmetry relation (*i.e.*, it checks whether the node (h, m) is connected to something in $G'_{\mathcal{R}_2}$). If such a query exist, then both are treated simultaneously in a "symmetric" way. In particular, if either one of these concerns a reachable state, then it is treated specially, even if it not the original query, but the "symmetric" one. The first two games are identical to those of the ideal case.

Game 3: Let us discuss the probability that $\mathcal{S}_0$ fails. It can only happen if h is reachable, which in turn means that h' is randomly distributed in $\mathcal{H}$. $\mathcal{S}_0$ aborts when $h' \in \mathcal{W}$, $h' \in V$ or when $\mathsf{Reach} \cup \{h'\}$ covers an edge of $G_{\mathcal{R}_2}$. The probability that $h' \in \mathcal{W}$ is $|\mathcal{W}|/2^n$, and the probability that $h' \in V$ is upper-bounded by $(4 \cdot q_S + 1)/2^n$, since $|V| \leq 4 \cdot q_s + 1$. Let us now discuss the probability that an edge of $G_{\mathcal{R}_2}$ is covered by Reach.

A simple induction on the number of queries shows that the chaining values in Reach are all randomly and independently distributed in $\mathcal{H}$ (this is because Reach is always extended by h' on line 19, and h' is itself always generated randomly). If we ignore the abort conditions, Reach is a random subset of $\mathcal{H}$ of size $k \leq q_s + 1$ after q_S queries. There are $\binom{2^n}{k}$ such subsets, and amongst these $\binom{2^n}{k-2}$ cover a given edge. The probability that at least one edge out of $|\mathcal{WP}|$ is covered is thus upper-bounded by $|\mathcal{WP}| \cdot \binom{2^n}{k-2}/\binom{2^n}{k}$,

which it itself upper-bounded by $|\mathcal{WP}| \cdot k^2/(2^n - k)^2$. After q_s queries, the probability of failure is therefore bounded by:

$$|\mathbb{P}[G_3] - \mathbb{P}[G_2]| \leq 4 \cdot \frac{(q_S+1)^2}{2^n} + |\mathcal{W}| \cdot \frac{q_S+1}{2^n} + |\mathcal{WP}| \cdot \frac{q_S+1^2}{(2^n - q_S - 1)^2}$$

Game 4: We claim that the following four statements hold:

(*i*) In Game 3, *i.e.*, when $\mathcal{D}$ interacts with $\left(\mathcal{T}^{RO}, \mathcal{S}_0\right)$, the answers of $\mathcal{S}_0$ are consistent with those of *RO* as long as $\mathcal{S}_0$ does not abort.

(*ii*) In Game 4, *i.e.*, when $\mathcal{D}$ interacts with $\left(\mathcal{T}_1^{pf-MD(\mathcal{S}_0)}, \mathcal{S}_0\right)$, the answers of $\mathcal{S}_0$ are consistent with those of *RO* as long as $\mathcal{S}_0$ does not abort.

(*iii*) $\mathcal{T}^{RO}$ and $\mathcal{T}_1^{pf-MD(\mathcal{S}_0)}$ give the same answers until the simulator aborts.

(*iv*) As long as it does not abort, the answer of $\mathcal{S}_0$ always comply with the relation $\mathcal{R}$.

Consistency with the VIL-RO. Establishing the first three points can be done in the same as it was done in the ideal case. The simulator relies on the fact that Reach is a tree, and that it exactly describes the reachable chaining values in V. This can be established by arguing that if $\mathcal{S}$ does not abort, then h' is the only new reachable chaining value created by the current invocation of FreshValue. Note that $\overline{h}$, if it exists, is not reachable.

Conformance to the Relation. The main point is that the relation can never be falsified on reachable states, and that the samplers are used on non-reachable states to ensure that the answers are consistent with the relation. More precisely, the simulator aborts as soon as a state in $\mathcal{W}$ becomes reachable, or a pair of states in $\mathcal{WP}$ becomes reachable.

Let us assume that the distinguisher can find a query (h, m, h') with $h \xrightarrow{m} h'$ such that $\mathcal{R}_1(h, m, h')$ does not hold. Then we have $h \in \mathcal{W}$ by definition of $\mathcal{W}$, therefore h cannot be reachable and h' has necessarily been build by Sampler_1. By definition of Sampler_1, $\mathcal{R}_1(h, m, h')$ must hold.

Similarly, let us assume that the distinguisher finds $h_1 \xrightarrow{m_1} h'_1$ and $h_2 \xrightarrow{m_2} h'_2$ such that $\mathcal{R}_2(h_1, m_1, h_2, m_2, h'_1, h'_2)$ does not hold. By definition of $\mathcal{WP}$ we have $(h_1, h_2) \in \mathcal{WP}$ therefore h_1 and h_2 cannot both be reachable. Moreover, we have $(h_1, m_1) \leftrightarrow (h_2, m_2) \in G'_{\mathcal{R}_2}$. Without loos of generality, we assume that h_2 is not reachable. When the first query involving (h_1, m_1) or (h_2, m_2) was sent to $\mathcal{S}_0$, the simulator built the second query. If h_1 was reachable, h'_1 has been built by calling the VIL-RO and h'_2 has been built by Sampler_2, with assures that $\mathcal{R}_2(h_1, m_1, h_2, m_2, h'_1, h'_2)$ holds. Similarly, if h_1 is not reachable, h'_1 has been built by Sampler_1, and h'_2 by Sampler_2. We note that if h_1 was not reachable at the time when it was queried, it cannot become reachable later without causing the simulator to abort.

Finally, we obtain that the view of the distinguisher does not change as long as the simulator does not abort:

$$\begin{aligned}
|\mathbb{P}[G_4] - \mathbb{P}[G_3]| &\leq \mathbb{P}[\mathcal{S}_0 \text{ fails in Game 3}] + \mathbb{P}[\mathcal{S}_0 \text{ fails in Game 4}] \\
&\leq 2 \cdot \mathbb{P}[\mathcal{S}_0 \text{ fails in Game 4}] \\
&\leq 8 \cdot \frac{(q_S + \kappa \cdot q_O)^2}{2^n} + 2|\mathcal{W}| \cdot \frac{q_S + \kappa \cdot q_O}{2^n} + 2|\mathcal{WP}| \cdot \frac{(q_S + \kappa \cdot q_O)^2}{(2^n - q_S - \kappa \cdot q_O)^2}
\end{aligned}$$

And we conclude:

$$\begin{aligned}
\varepsilon &= \left|\mathbb{P}\left[\mathcal{D}^{H^F, F} = 1\right] - \mathbb{P}\left[\mathcal{D}^{RO, \mathcal{S}} = 1\right]\right| \\
&\leq |\mathbb{P}[G_1] - \mathbb{P}[G_6]| \\
&\leq 4\mathbb{P}[\mathcal{S}_0 \text{ fails in Game 4}] \\
&\leq 16 \cdot \frac{(q_S + \kappa \cdot q_O)^2}{2^n} + 4 \cdot |\mathcal{W}| \cdot \frac{q_S + \kappa \cdot q_O}{2^n} + 4 \cdot |\mathcal{WP}| \cdot \frac{(q_S + \kappa \cdot q_O)^2}{(2^n - q_S - \kappa \cdot q_O)^2}
\end{aligned}$$

Conclusion

Our work on hash functions modes of operations was mostly centered on second preimage attacks, and provable second preimage resistance. These questions are essentially of theoretical interest, because generic attacks are usually unlikely to ever become practical. This applies in particular to second preimage attacks which do not just rely on a black-box collision finder that could efficient with the passage of time. The Trojan Message Attack is a rare example of a generic attack becoming practical.

The complexity of generic attacks is usually above the birthday bound, but it is striking that as soon as the adversary is allowed to perform at least $2^{n/2}$ operations, where n denotes the size of the internal state of the hash function, uncontrollable phenomena happen. This is typically exemplified by the generic attacks on concatenated or iterated hashes, where sophisticated herding techniques allows to keep an arbitrary large number of parallel hash processes under control.

All-in-all, the avalanche of generic attacks on the Merkle-Damgård construction stimulated the search for alternatives. From a theoretical standpoint, this question has been settled: the wide-pipe mode of operation and the sponge construction both offers unquestionable benefits over Merkle-Damgård, most notably the provable absence of generic attacks. The SHA-3 competition has also demonstrated that they can be efficient and practical.

There might nevertheless be some practical scenario (*e.g.*, very constrained memory) where narrow-pipe constructions could be a better fit. In addition, investigating the properties of narrow-pipe hash functions is more interesting, because the situation is much less clear. We have shown that there are narrow-pipe hash function modes of operations immune to generic second preimage attacks. Are there narrow-pipe modes of operation provably resistant against herding attacks? The Zipper Hash seems to be a promising candidate, but it is very unpractical.

The designers of new hash function modes of operation will have to justify their work in light of the existence of provably secure and decently practical constructions. We thus believe that these new modes will have to provide some additional feature, *e.g.*, stronger provable security in the standard model. The sponge paradigm [BDPA08] is a striking example: it provides more features than most other constructions (it can produce digests of any sizes, for instance). It has already gained respectability even though it only appeared three years ago, because it was demonstrated that it could be simultaneously quite practical and yield apparently secure constructions.

Finally, the interplay between attacks and security proofs is a fascinating subject. Our proofs of resistance to second preimage attacks directly exploit the way the attacks work. Constructing simulators for indifferentiability proofs is often a (challenging) trial-and-error process, where the designer, sometimes with the help of a few colleagues, puts her own algorithm to the test of carefully crafted distinguishers. An important phase of the elaboration of the security proof therefore consists in the elaboration of attacks...

Computer-Aided Cryptanalysis of Byte-Oriented Primitives

Introduction

The field of block cipher design has advanced greatly in the last two decades. New strategies of designing secure block ciphers were proposed, and following the increase in computing power, designers could offer larger security margins with reduced performance penalties. As a result, practical attacks on block ciphers became extremely rare, and even "certificational attacks" (that is, attacks which are not practical but are still faster than exhaustive key search on the full version of the cipher), are not very common. As we already argued in the introduction of part I, the situation of hash functions (and also stream ciphers) is dramatically different. Several commonly used hash functions were practically broken in recent years [SSA+09, MP08], and practical attacks on new stream cipher designs appear every several months [SHJ09, HJ11].

This led to two approaches in the block-cipher cryptanalysis community. The first is to concentrate on attacking reduced-round variants of block ciphers, where the usual goal of the adversary is to maximize the number of rounds that can be broken, using less data than the entire codebook and less time than exhaustive key search. This approach usually leads to attacks with extremely high data and time complexities, such as those shown in [DS08, LDKK08]. The second approach is to allow the adversary more degrees of freedom in his control. Examples of this approach are attacks requiring adaptive chosen plaintext and ciphertext queries, the related-key model, the related-subkey model, and even the known-key model [BK09, DKS10b, KR07]. This approach allows to achieve practical complexities even against widely used block ciphers such as the AES, but the practicality of the models themselves in real-life situations can be questioned.

Attacks following each of these approaches are of great importance, as they ensure that the block ciphers are strong enough, almost independently of the way in which they are deployed. Moreover, they help to establish the security margins offered by the ciphers. A block cipher which is resistant to attacks when the adversary has a strong control and almost unrestricted resources offers larger security margins than a block cipher which does not possess this resistance.

At the same time, concentrating the cryptanalytic attention only on such attacks may prove insufficient to truly understand the security of the analyzed block cipher. It seems desirable to also consider other approaches, such as restricting the resources available to the adversary in order to adhere to "real-life" scenarios. For example, one may study the maximal number of rounds that can be broken with *practical* data and time complexity, as considered for instance in [BDK+10] with respect to the related-key model.

Low Data Complexity Attacks. In this second part we pursue this direction of research, but we concentrate on another restriction of the adversary's resources. In the attacks we consider, the time complexity is not restricted (besides the natural bound of exhaustive search), but the data complexity is restricted to only a few known or chosen plaintexts. At first glance, this scenario may seem far fetched. However, it makes sense to question whether it is easier in practice for an attacker to perform 2^{50} elementary operations or to acquire 50 plaintext/ciphertext pairs. We are inclined to believe that in many actual-life situations, performing the computation is easier. In addition, the EMV protocol for credit cards specifies that at most 2^{16} signatures shall be issued by a given chip, which practically restricts the quantity of available data. It also turns out that this setup is very natural in the context of several classes of attacks:

- *Slide attacks* [BW99]: This class of attacks is especially designed against block ciphers whose rounds are very similar to each other. The main feature of slide attacks is that they are independent of the number of rounds, and thus, the common countermeasure of increasing the number of rounds is not effective against them. Since most other attack techniques can be easily undermined by adding a few rounds, this makes the slide attacks one of the most powerful attacks against modern block cipher designs. The main idea of the slide attacks is to reduce the attack on the entire block cipher to an attack on a single round, where the data available to the adversary is only *two* known plaintext/ciphertext pairs. Hence, the scenario considered in our paper is exactly the one faced by the adversary in the slide attack. We note that several variants of slide attacks suggested methods to increase the amount of data available to the adversary [BW00, BDK07, Fur01]. However, all these methods either require the knowledge of large portion of the codebook or perform in the adaptively chosen plaintext model.
- *Attacks based on fixed point properties* [CBW08]: In this class of attacks, the adversary looks for a fixed point of *some part of* the encryption process. For such a fixed point, the cipher is reduced to a

smaller variant, which can (sometimes) be attacked efficiently. Since usually the number of fixed points is extremely small (*e.g.*, one or two), the adversary's goal is to attack a reduced-round variant of the cipher given a few known plaintexts.
- *Side channel attacks*: In this class of attacks, the adversary has access to some information on the internal states during the encryption process. Usually, due to practical restrictions, the amount of data available to the adversary is extremely low. In the (somewhat unlikely) case where the information available to the adversary is the full intermediate state after a few rounds, the scenario the adversary faces is exactly the one considered in our paper. We note that the complementary scenario, where the adversary has access to a small part of the internal state in multiple encryptions, was studied in [DS09].
- *Building block in more complex attacks*: As we demonstrate on the example of AES, an attack on 2-round AES with two known plaintexts can be leveraged to a known plaintext attack on 6-round AES. The attack uses a meet-in-the-middle approach combined with a low probability differential. Also, the block cipher GOST was recently broken by such an attack, where an attack against the full 32 rounds is reduced to an attack against 8 rounds using four known plaintext [Iso11]. We expect that such "leveraging" attacks are applicable against other block ciphers as well.
- *Attacks on other primitives based on the block cipher*: In recent years, many designs of stream ciphers (*e.g.*, Sosemanuk [BBAC$^+$05]), hash functions (*e.g.*, Hamsi [OK09]), MACs (*e.g.*, ALPHA-MAC [DR05a]), etc., use a small number of rounds of a block cipher as one of their components. Some of these constructions can be broken when internal collisions are found and thus the attackers are in a setting in which they have to exploit a very limited quantity of data (the colliding inputs). The attacks we consider can be used against these primitives, once collisions are found (which may require a large quantity of data though).

The AES, a Natural Target. In order to make our results concrete, we have chosen to concentrate on a single block cipher — the AES, the *Advanced Encryption Standard* [NIS01]. The AES is a 128-bit block cipher with a variable key length (128, 192, and 256-bit keys are supported). Since its selection, AES gradually became one of the most widely used block ciphers and received a great deal of cryptanalytic attention, both during the AES process, and even more after its selection. Studying reduced-round versions of AES is also motivated by the recent blossom of many AES-based primitives for hashing or authentication, such as the Grøstl, ECHO, SHAVite-3 and LANE hash functions, the LEX [Bir06a] stream cipher, or the Alpha-MAC [DR05a] and Pelican-MAC [DR05b] message authentication codes. In these construction, AES rounds (and sometimes the full AES) are used as internal permutations. A possible explanation of this fancy is that the AES enjoys very interesting security properties against statistical attacks: two rounds achieve full diffusion, and there exist very good differential and linear lower bounds for the best differential on four rounds [KMT01b, KMT01a, Kel04]. It results that in some applications (such as authentication or hashing), only a small number of AES rounds are sufficient to yield a reasonable internal permutation. The relative weakness of the permutation is then usually compensated by the fact that the internal state is either hidden from the adversary (in MACs, because of the secret key), or only partially accessible to the adversary (because only certain parts can be seen and modified). Furthermore, in some particular attacks, such as side-channel attacks, only a small number of rounds of the cipher needs to be studied [PQ03, BK07].

Lastly, much attention has been recently devoted to the AES block cipher as a by-product of the NIST SHA-3 competition. The low diffusion property of the key schedule has been used to mount several related-key attacks [BKN09, BK09, BDK$^+$10, KBN09] and differential characteristic developed for hash functions have been used to also improve single-key attacks [DKS10a]. The AES is therefore a relevant and interesting case study to demonstrate our techniques.

Algebraic Structure. Since the introduction of the AES in 2001, it has been questioned whether its simple algebraic structure could be exploited by cryptanalysts. Soon after its publication as a standard [NIS01], Murphy and Robshaw showed in 2002 an interesting algebraic property: the AES encryption process can be described only with simple algebraic operations in $GF\left(2^8\right)$ [MR02]. Such a result paved the way for multivariate algebraic techniques [CP02, Cid04] since the AES encryption function can also be described by a very sparse overdetermined multivariate quadratic system over $GF(2)$. However, so far this approach has not been so promising [MV04, CL05], and the initial objective of this simple structure, providing good security protections against differential and linear cryptanalysis, has been fulfilled.

Techniques. Our concentration on attacks with an extremely small data complexity affects the techniques available to us. The small amount of available data makes the usual statistical attacks, such as differential and linear cryptanalysis, almost irrelevant. Algebraic attacks seem to be more well-suited, even though it seems that they perform better when a lot of data is available. Anyway, such attacks using either SAT solvers or Gröbner basis algorithms [MR02, BPW06], have never been able, so far, to endanger even very reduced

versions of the AES. These attacks encode the problem into a system of equations, then feeds the equations to a generic, sometimes off-the-shelf equation solver, such as a SAT-solver or a Gröbner basis algorithm [2]. The main obstacle in these approaches is the non-linear component of the AES (the S-box), that only admits "bad" representations (for instance, it is a high degree polynomial over the AES finite field), and increases the complexity of the equations, even though low degree implicit equations may also exist.

Our attacks have an algebraic flavor, but are mostly based on the meet-in-the-middle approach, combined with guess-and-determine and differential-type ideas, as well as vast exploitation of the key schedule of the analyzed block cipher (a not-so-common feature in single-key attacks).

Automated Tools. Because of their special nature, finding low data complexity attacks "by hand" is difficult. A guess and determine attack against three AES rounds for instance involves about 50 deduction steps. Even on a single AES rounds, where attacks should be easier to find, some subtle shortcuts are not easy to find at all. To alleviate the task of cryptanalyst, a natural solution is to create *automated tools* tailored to find an attack of some kind on a given primitive. This approach has been successful many times: nice examples include the cryptanalyses of Grindhal by Peyrin [Pey07] and of RadioGatùn by Fuhr and Peyrin [FP09]: in both case a custom-made program found a truncated or symmetric differential characteristic leading to a collision. Biryukov and Nikolić designed a tool to automatically find related-key attacks in the AES [BN10], and along with Khovratovich they designed a tool to search for collision attacks on byte-oriented hash functions [KBN09]. Much earlier, Matsui designed a tool to find differential characteristics and linear approximations [Mat93]. Leurent developed a tool to find good differential paths in MD4 and MD5 [FLN07], while De cannière and Rechberger developed a tool to find good differential characteristics in SHA-1 [CR06], etc.

This part devoted to low data complexity attacks is made of two chapters. In chapter 8 we describe two tools we designed to find low-data complexity attacks against the AES. Both tools outperformed (well-known) human cryptanalysts in several occasions. In chapter 9, we present a collection of low-data complexity attacks reduced-round versions of the AES, and against AES-based primitives such as LEX, Alpha-MAC or Pelican-MAC [DR05b]. Some of these attacks were found by hand, and some others were found by our automated tool.

2. Gröbner Bases will be discussed more in-depth in part III

Chapter 8

Automated Tools For Low Data Complexity Attacks on AES Derivatives

This chapter presents two software tools we have designed to find low-data complexity attacks. These results are the product of a joint effort with Patrick Derbez, and they have been published at CRYPTO'2011 [BDF11].

In this chapter, we describe two automated tools we designed to find low data complexity attacks on the AES. The global strategy is to encode the cryptanalytic problem at hand (key-recovery, state-recovery, differential pair-finding, etc.) as a system of equations over $\mathbb{F}_{2^8}$. Our tools can then be seen as *equation solvers* of a special kind, because they take the equations describing the problem and try to find all the solutions of these equations in an efficient way. The tools are thus somewhat generic, as they are not specialized to a particular block cipher.

This approach is yet quite different from the usual algebraic attacks. Our tools do not try to solve the equations using a generic "one-size-fits-all" equation solver, but they first run a search for an *ad hoc* solver tailored for the equations to solve. The tools generate the source code of this solver, build it (using standard compilers) and run it in order to obtain the actual solutions. The resulting standalone program is, in fact, the attack. It can be run independently of the tools that were used to find it. While the complexity of the tools seems difficult to upper-bound, the tools give a decently precise approximation of the expected run-time of the attacks they produce.

These tools can be applied to systems of linear equations containing non-linear permutations of the field, such as bijective S-boxes. Our idea is to consider the S-box as a black box permutation. We only use few properties of this function and our attacks work for *any good instantiation* of the S-box.

This approach is reminiscent of the ideas used by Khovratovich, Biryukov and Nicolić to find collisions in an AES-based hash function (more precisely, a hash function using a large version of Rijndael in Davies-Meyer mode) [KBN09]. They first found a "good" colliding truncated differential path, and they were facing the problem of finding a conforming pair to obtain an actual collision. The basic strategy for finding a message pair conforming to a differential characteristic consists in exhaustively trying all possible input values and checking if the constraints are satisfied. In order to speed up the collision search, these authors used a message modification technique: they described the hash function using a system of linear equations with an S-box, and added equations to enforce that the message and chaining value follow their truncated differential characteristic inside the function. Solving the equations would yield a collision, and the approach they proposed is to look automatically for constraints that could be satisfied by setting a particular variable to a particular value without violating other constraints. To this end, they use linear algebra, and essentially consider x and $S(x)$ to be independent variables, and then greedily satisfy constraints. This method is however limited in that when the greedy strategy aborts, *i.e.*, when no easily-satisfiable constraints remain, then probabilistic trials is the only fallback.

Our tools exploit the algebraic simplicity of the AES, or, more precisely, the fact that none of its operations interact in a disastrous way with the algebraic structure of $\mathbb{F}_{2^8}$. The S-box operates on a single element of $\mathbb{F}_{2^8}$, and the only operation that involves more than one element of the field is the XOR, which is precisely the addition of the field. The situation would be much less algebraically "clean" if, for instance, each column were rotated by one bit: this particular operation is difficult to describe over $\mathbb{F}_{2^8}$.

We also exploit the fact that the cipher can be described by *sparse* equations over $\mathbb{F}_{2^8}$. Both features stems from the fact that the AES is an (efficient) byte-oriented cipher. We believe that it is the first time that the algebraic simplicity of the AES is effectively harnessed by cryptanalysts.

Techniques. Our tools try to find attacks automatically by searching some classes of guess-and-determine and meet-in-the-middle attacks. They take as input a system of equations describing the cryptographic primitive and some constraints on the plaintext and ciphertext variables (*e.g.*, a differential relation). They

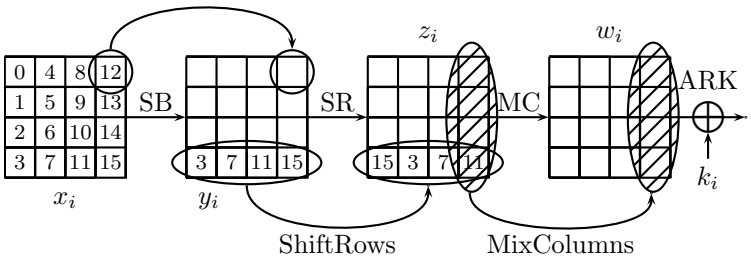


Figure 8.1: An AES round

then solve the equations by first running a (potentially exponential) search for a *customized* solver for the input system. The solver is built, run, and the solutions are computed.

We describe two tools. Our preliminary tool uses a depth-first branch-and-bound search to find "good" guess-and-determine attacks. It has been used to generate some of the attacks published in [BDD+10] and described in the next chapter, and outperformed human cryptanalysts in several occasions. The class of attacks searched for by this preliminary tool is quite restricted, and it fails to take into account important differential properties of the S-box. It is therefore mostly suited to the case where a *single known plaintext* is available.

Many ideas developed for this preliminary tool could be reused in our second, more advanced tool, which finds more powerful attacks, such as Meet-in-the-Middle attacks. For instance, this improved tool automatically exploits the useful fact that an input and output difference on the S-box determine uniquely on average the actual input and output values. Because the search procedure of our improved tool is essentially a *saturation* procedure, the algorithmic techniques we use are reminiscent of (and inspired by) the Buchberger algorithm [Buc65], as well as many techniques used in automated theorem proving.

We have applied these algorithms to reduced-round versions of the AES, to the stream cipher LEX, to Pelican-MAC, to the block cipher SQUARE [DKR97], to SkipJack, etc. Some of the attacks found by these algorithms are described in chapter 9. Once given a very little bit of human knowledge, our tools automatically rediscover the best known attacks on Alpha-MAC and LEX. They also discovered the best known attack on Pelican-MAC (a different attack with the same complexity was independently discovered at about the same time by Dunkelman, Keller and Shamir [DKS11]). The tools also helped us to find the best known attack on the LEX stream cipher. The tools can output the source code of a program performing the attack, but can also provide a somewhat human-readable description of the attack procedure, which was used in some cases to understand and describe the attacks.

8.1 Description of the AES

The Advanced Encryption Standard [NIS01] is a Substitution-Permutation network that supports key sizes of 128, 192, and 256 bits. A 128-bit plaintext (resp. a 128-bit key or internal state) is treated as a byte matrix of size 4×4, where each byte represents a value in $\mathbb{F}_{2^8}$. An AES round applies four operations to the state matrix:

- `SubBytes` (SB) — applying the same 8-bit to 8-bit invertible S-box 16 times in parallel on each byte of the state,
- `ShiftRows` (SR) — cyclic shift of each row (the i'th row is shifted by i bytes to the left),
- `MixColumns` (MC) — multiplication of each column by a constant 4×4 matrix over $\mathbb{F}_{2^8}$, and
- `AddRoundKey` (ARK) — XORing the state with a 128-bit subkey.

We outline an AES round in Figure 8.1. Before the first round, an additional `AddRoundKey` operation (using a whitening key) is applied, and in the last round the `MixColumns` operation is omitted. The number of rounds depends on the key length: 10 rounds for 128-bit keys, 12 rounds for 192-bit keys, and 14 rounds for 256-bit keys. We use the round numbers $1, \ldots, Nr$, where Nr is the number of rounds ($Nr \in \{10, 12, 14\}$). We only consider the AES with 128-bit keys and 10 rounds. Because the final AES round is different from the others, we use the term "r.5 rounds AES" to denote the AES reduced to $(r+1)$ rounds, including the final round. We use "r rounds AES" to denote the AES reduced to r identical full rounds. In our terminology, the "normal" 128-bit AES has 9.5 rounds.

Let $\mathbb{F}_{2^8}$ denote the finite field with 256 elements used in the AES, namely $\mathbb{F}_2[X]/\mathbb{X}^8 + X^4 + X^3 + X + 1$. We denote the S-box of the `SubBytes` transformation by $S : \mathbb{F}_{2^8} \to \mathbb{F}_{2^8}$. In a 4×4 matrix, we use the following numbering of bytes: byte zero is the top-left corner, the first column is made of bytes 0-3, while the last column is made of bytes 12-15, with byte 15 in the bottom-right corner (this is illustrated by Figure 8.1). We denote the four columns of a 4×4 matrix M by $M[0..3], M[4..7], M[8..11]$ and $M[12..15]$ respectively.

As we consider only the AES with 128-bit key, we shall describe only its key schedule algorithm. The key schedule of the other variants can be found in [NIS01]. The key schedule of AES-128 takes the 128-bit master key k_0 and extends it into 10 subkeys $k_1, \ldots, k_{10}$ of 128 bits each using a key-schedule algorithm described by the following equations:

$$KS_i : \begin{cases} k_i[j] + k_i[j-4] + k_{i-1}[j] = 0, & j = 4, \ldots, 15 \\ k_i[0] + k_{i-1}[0] + S\left(k_{i-1}[13]\right) + \texttt{RCON}_i = 0 \\ k_i[1] + k_{i-1}[1] + S\left(k_{i-1}[14]\right) = 0 \\ k_i[2] + k_{i-1}[2] + S\left(k_{i-1}[15]\right) = 0 \\ k_i[3] + k_{i-1}[3] + S\left(k_{i-1}[12]\right) = 0 \end{cases}$$

We denote by x_i the internal state entering round i (*i.e.*, before `SubBytes`), by y_i the internal state between the `SubBytes` and `ShiftRows` operations, while z_i and w_i denote the internal state before and after the `MixColumns` operation, respectively. The plaintext is denoted by P, and the ciphertext is denoted by C. One round is represented by these equations:

$$R_i : \begin{cases} y_i + S(x_i) = 0 \\ w_i + \begin{pmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{pmatrix} \times \begin{pmatrix} y_i[0] & y_i[4] & y_i[8] & y_i[12] \\ y_i[5] & y_i[9] & y_i[13] & y_i[1] \\ y_i[10] & y_i[14] & y_i[2] & y_i[6] \\ y_i[15] & y_i[3] & y_i[7] & y_i[11] \end{pmatrix} = 0 \\ x_{i+1} + w_i + k_{1+i} = 0 \end{cases}$$

It is straightforward to form the system of equations $\mathbb{E}$ describing the full encryption process along with the key schedule: we just have to concatenate some KS_i's and some R_i's (without forgetting the initial key addition). Since the right-hand side of all these equations are zero, we stop representing it from now on.

These equations are extremely sparse, containing at most 5 terms. Each variable occurs in at most 3 equations. These equation form a constrained linear system, where the constraints are that for all variables, the values of x and $S(x)$ are not independent.

Let us denote by $\mathcal{V}(X)$ the vector space spanned by $1, x, S(x)$ for all $x \in X$, for any set of variables X. If we denote by $\mathbb{X}$ the set of all key and internal state variables, then the cipher equations span a subspace of $\mathcal{V}(\mathbb{X})$. Any basis of this subspace describes an equivalent system of equations. Therefore, by an abuse of notation we identify the set of equations describing the block cipher with the vector space formed by all the linear combinations of the equations, and we still denote it by $\mathbb{E}$. We also introduce the notation $\mathcal{S}(\mathbb{E})$ to denote the set of solutions of the equations $\mathbb{E}$.

In some cases, we are interested in interchanging the order of the `MixColumns` and `AddRoundKey` operations. As these operations are linear they can be interchanged, by first XORing the data with an equivalent key and only then applying the `MixColumns` operation. We denote the equivalent subkey for the altered version by:

$$u_i = MC^{-1}(k_i) = \begin{pmatrix} 0e & 0b & 0d & 09 \\ 09 & 0e & 0b & 0d \\ 0d & 09 & 0e & 0b \\ 0b & 0d & 09 & 0e \end{pmatrix} \times k_i$$

8.2 A preliminary Tool for Simple Guess-And-Determine Attacks

Confronted with a system of equations in $\mathcal{V}(\mathbb{X})$ (possibly describing a cryptographic problem), the most naive way to obtain its solutions consists in enumerating all the possible values of the variables and retaining only the combinations satisfying all the equations. However, equations in $\mathcal{V}(\mathbb{X})$ are such that, in a given equation, once all the terms but one are known then the last one can be found efficiently. This is especially useful when the equations are sparse (efficient cryptographic primitives usually result in sparse equations). This enables more or less efficient *guess-and-determine* techniques to solve the equations. In a cryptographic setting, guess-and-determine attacks are often found when data is very scarce, and statistic attacks are therefore impossible. Guess-and-determine attacks can be more or less sophisticated, but the simplest ones typically take the following form:

1: **for all** values of some part of the (unknown) internal state **do**
2: Compute the full internal state
3: Retrieve the secrets
4: Check compatibility of secrets with available data
5: **if** match available data **then return** secrets
6: **end for**

The difficulty in finding such an attack is to find which parts of the internal state to enumerate, and to find how to recover the rest. In this section, we present a preliminary tool that finds such attacks automatically. It takes as an input a system of equations $\mathbb{E} \subset \mathcal{V}(\mathbb{X})$ and a set $\mathbb{K}_0 \subset \mathbb{X}$ of initially *known* variables—these are the variables corresponding to the available data, for instance the plaintext, the ciphertext, the keystream, etc. The preliminary tool returns a C++ function (the "solver") which enumerates the solutions of $\mathbb{E}$ (using negligible memory), given the actual values of the known variables. The tool also returns the exact number of elementary operations the solver performs in the worst case.

This preliminary tool has been developed while performing the research that led to the results presented in the next chapter, and which have been published in [BDD+10]. The preliminary tool has for instance been used to find *one known plaintext* attacks against 1, 1.5, 2, 2.5 and 3 AES rounds, systematically beating the best results found manually. For instance, prior to the publication of [BDD+10], the best attack on one (full) AES round was a guess-and-determine attack with complexity 2^{48} described in [DK10b]. The preliminary tool found in less than a second an attack of complexity 2^{40} and generated its implementation, which we verified to be correct.

8.2.1 Knowledge Propagation.

The core idea of this preliminary tool is quite simple: if there is a linear combination of the equations in which the values of all terms are known except one, then the value of this last term can be determined efficiently.

When applied to the AES, this simple procedure automatically harnesses the simple and clean algebraic structure of the cipher. It automatically exploits the linear relations existing in the key-schedule, as well as the MDS property of the `MixColumns` operation: if $y = \texttt{MixColumns}(x)$ then knowlege of any four bytes in (x, y) is sufficient to recover the remaining four efficiently.

An "Algebraic" Point of View. The acquisition of further knowledge, either by "guessing" or "determining" has a simplifying effect on the equations (it removes an active variable whose value is unknown from the problem). This simplification of the original equations in fact has a clean algebraic description.

Let $\mathbb{K} \subset \mathbb{X}$ be a set of variables whose values are known. If we substituted the values of known variables into the original equations $\mathbb{E}$, we would indeed get a system with less variables. In fact, this reduced system is essentially the *quotient space* of $\mathbb{E}$ by $\mathcal{V}(\mathbb{K})$: starting from an equation $f \in \mathbb{E}$, its equivalence class $[f]$ in the quotient contains a representative where all the variables in $\mathbb{K}$ have disappeared. Alternatively, the variable x can be deduced from $\mathbb{K}$ if either $[x]$ or $[S(x)]$ belong to the quotient of $\mathbb{E}$ by $\mathcal{V}(\mathbb{K})$, and we will write $x \in \textsc{Propagate}(\mathbb{K})$ when it is the case. To see why, observe that $[x] \in \mathcal{V}(\mathbb{E})/\mathcal{V}(\mathbb{K})$ (or $[S(x)] \in \mathcal{V}(\mathbb{E})/\mathcal{V}(\mathbb{K})$) means that there exists $k \in \mathcal{V}(\mathbb{K})$ such that $x + k \in \mathcal{V}(\mathbb{E})$. In other terms, there is a linear combination of the equations $\mathbb{E}$ that can be written $x + k$ (respectively, $S(x) + k$). It follows that in any solution of the equations $\mathbb{E}$, the value of x is the value of k. There is therefore a straight-line program of size $\mathcal{O}(|\mathbb{K}|)$ that uniquely determines the value of x given the values of the variables in $\mathbb{K}$—it just has to evaluate k.

Observe in passing that it is not difficult to check whether $x \in \textsc{Propagate}$: it comes down to solving a system of $2|\mathbb{X}|$ linear equations in $|\mathbb{E}|$ variables over $\mathbb{F}_{2^8}$.

8.2.2 Automatic Search for a Minimal Number of Guesses

Given a set of "known" variables $\mathbb{K} = \mathbb{K}_0$, we may propagate knowledge and obtain the value of new variables, yielding a new set of known variables $\mathbb{K}_1$. But it may turn out that new variables may again be obtained from $\mathbb{K}_1$. We therefore define the function $\textsc{Propagate}^*(X)$ which returns the least fixed point of $\textsc{Propagate}$ containing X:

$$\textsc{Propagate}^*(X) = \left\{ \begin{array}{l} \textbf{let } Y = \textsc{Propagate}(X) \textbf{ in} \\ \quad \textbf{if } X = Y \textbf{ then return } Y \textbf{ else return } \textsc{Propagate}^*(Y) \end{array} \right.$$

Note that this definition is well-founded, because $\textsc{Propagate}$ is both monotonic and bounded. Indeed, it is very easy to check that $X \subseteq Y$ implies $\textsc{Propagate}(X) \subseteq \textsc{Propagate}(Y)$. It also follows that $\textsc{Propagate}^*$ is also monotonic.

A guess-and-determine solver has been found as soon as we have found a set $\mathbb{G}$ of "guesses" such that $\textsc{Propagate}^*(\mathbb{G}) = \mathbb{X}$. In that case, we will say that $\mathbb{G}$ is *sufficient*. The problem thus comes down to automatically finding a sufficient set of minimal size.

The process of exhaustively searching such a guess-and-determine attack can be seen as the exploration of a Directed Acyclic Graph (DAG) whose nodes are sets of variables. The starting node is the set $\mathbb{K}_0$, and the terminal node is $\mathbb{X}$. For any set of variables X, and any variable $y \notin X$ there is an edge $X \xrightarrow{y} X \cup \{y\}$, meaning that we may always choose to "guess" the value of y to gain knowledge. Finally, for any set of

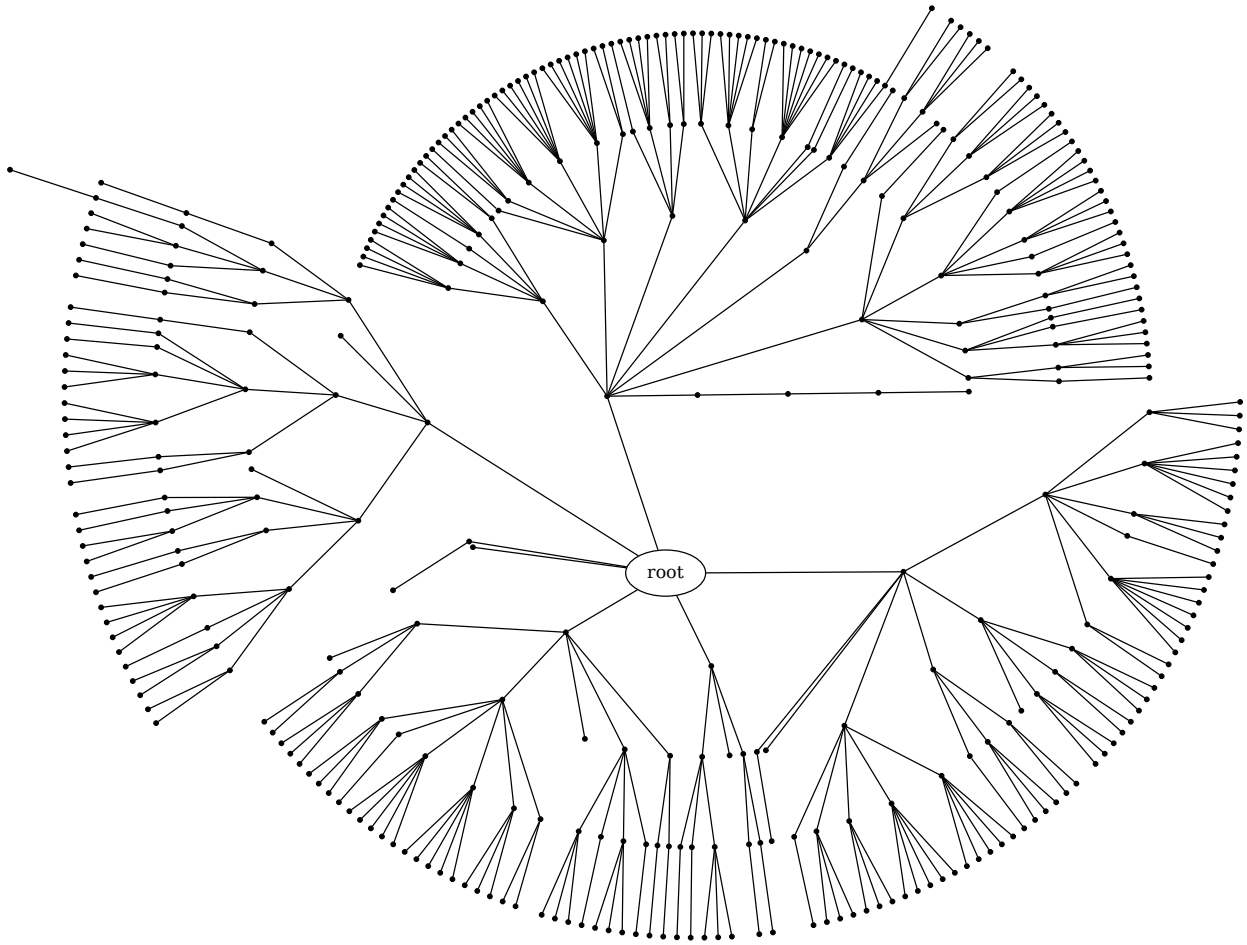


Figure 8.2: The possible sets of guessed variables explored by the tool to find an attack one full AES round. Each descendant has one more guess than its parent.

variables X, there is an edge $X \to \textsc{Propagate}^*(X)$, symbolizing the fact that increase our knowledge by propagation.

In this setting, the objective of the preliminary tool is to find a path from $\mathbb{K}$ to $\mathbb{X}$ going through a small (if not the smallest) number of "guess" edges. Indeed, the cost of the resulting attack is exponential in the number of traversed "guess edges". The problem is that the size of the DAG is exponential in the number of variables.

The search works in a depth-first branch-and-bound fashion reminiscent of the DPLL procedure implemented in many SAT-solvers. The pseudo-code of the search procedure is shown in Algorithm 8.1. The function $\textsc{Explore}(\mathbb{K}, \mathbb{G}, \mathbb{B})$ [1] returns a minimal set of variables to guess in order to be able to recover the entire internal state. Here $\mathbb{K}$ denotes the set of *currently known* variables (*i.e.*, the current node of the DAG), $\mathbb{G}$ denotes the set of variables that have been guessed so far, and $\mathbb{B}$ denotes the set of variables that have been guessed in the best known solution. This implicit assumption is that $|\mathbb{G}| < |\mathbb{B}|$, and that the result of Explore has cardinality smaller than or equal to $\mathbb{B}$. Evaluating $\textsc{Explore}(\mathbb{K}_0, \emptyset, \mathbb{X})$ returns a minimal solution.

8.2.3 Pruning Strategies

An obvious way to speed up the DAG exploration is to avoid guessing a permutation of a set of guesses previously tested. This can be easily enforced by choosing a total order relation $<_{\mathbb{X}}$ between variables and only guessing variables in increasing order. In order to further speed-up the search procedure, we used several *pruning strategies* that remove "guess" edges from the DAG without modifying its reachability properties. These pruning strategies appear in Algorithm 8.1 under the form of the FilterGuesses function, which only returns a subset of its argument.

1. no pun intended

Algorithm 8.1 Pseudo-code of the Preliminary Tool.

```
1: function Explore(𝕂, 𝔾, 𝔹)
2:     if 𝕂 = 𝕏 then return 𝔾
3:     if 𝕂 → Propagate*(𝕂) then
4:         return Explore(Propagate*(𝕂), 𝔾, 𝔹)
5:     if |𝔾| = |𝔹| − 1 then return 𝔹
6:     for all x ∈ FilterGuesses(𝕂) do
7:         recursive ← Explore(𝕂 ∪ {x}, 𝔾 ∪ {x}, 𝔹)
8:         if |recursive| < 𝔹 then 𝔹 ← recursive
9:         if |𝔾| = |𝔹| − 1 then return 𝔹
10:    end for
11:    return 𝔹
12: end function
```

Local Pruning. In simple words, if we need to guess the value of a new variable, and if guessing the value of x allows to deduce the value of y, then it is always better to guess x instead of y. More formally, we see that if $y \in \textsc{Propagate}^*(\mathbb{K} \cup \{x\})$, then:

$$\textsc{Propagate}^*(\mathbb{K} \cup \{y\}) \subseteq \textsc{Propagate}^*(\mathbb{K} \cup \{x\})$$

Given a set of known variables, this translates to a partial order relation on variables:

$$x \succcurlyeq_{\mathbb{K}} y \text{ if and only if } y \in \textsc{Propagate}^*(\mathbb{K} \cup \{x\}).$$

From this order relation we extract a *strict* order relation:

$$x \succ_{\mathbb{K}} y \iff x \succcurlyeq_{\mathbb{K}} y \text{ and } x \not\preccurlyeq_{\mathbb{K}} y.$$

The pruning strategy informally described above amounts to consider only maximal elements for $\succ_{\mathbb{K}}$. We call this strategy "local pruning" because it only requires a local exploration of the DAG around the current node $\mathbb{K}$.

Global Pruning. A somewhat surprising consequence of the fact that Propagate* is *monotonic* brings in a interesting result, enabling us to further discard some bad guesses in a very powerful way.

Lemma 8.1. *Let $V \subsetneq \mathbb{X}$ be an* insufficient *set of variables, and let $\mathbb{G} \subseteq \mathbb{X}$ be a sufficient set of variables. Then:*

$$\mathbb{G} \cap (\mathbb{X} - \textsc{Propagate}^*(V)) \neq \emptyset$$

Proof. Let us reason by contradiction and assume that $\mathbb{G} \cap (\mathbb{X} - \textsc{Propagate}^*(V)) = \emptyset$. Then, because $\mathbb{G}$ is a subset of $\mathbb{X}$, then $\mathbb{G} \subseteq \textsc{Propagate}^*(V)$. By monotonicity and idempotence of Propagate* we find: $\mathbb{X} = \textsc{Propagate}^*(\mathbb{G}) = \textsc{Propagate}^*(V) \neq \mathbb{X}$. □

If $\mathbb{G}$ denotes a sufficient set of minimal size, then Lemma 8.1 gives us *a priori* knowledge on $\mathbb{G}$, and it enables to choose the first guess of the search procedure in $\mathbb{X} - \textsc{Propagate}^*(V)$ without risking to throw the best solution away.

It is possible to exploit lemma 8.1 even further for more pruning. Assume that in the exploration process we currently know the variables in $\mathbb{K}$, and that we have guessed the variables in $\mathbb{G}$, so that $\mathbb{K} = \textsc{Propagate}^*(\mathbb{G})$. Let $\mathbb{B}$ be a sufficient set of minimal size such that $\mathbb{G} \subseteq \mathbb{B}$, *i.e.*, the best we may hope to find from the current state. Lemma 8.1 tells us that $\mathbb{B} \cap (\mathbb{X} - \textsc{Propagate}^*(V)) \neq \emptyset$. This reveals us some variables in $\mathbb{B}$, and could be used to direct the exploration towards $\mathbb{B}$. However, if V is badly chosen then it may very well be that all the interesting variables we learn to be in $\mathbb{B}$ are *already known*, in which case we would not learn anything.

However, choosing V to be a superset of $\mathbb{K}$ ensures that $\mathbb{K} \cap (\mathbb{X} - \textsc{Propagate}^*(V)) = \emptyset$, and thus removes the previous problem. We may safely choose our next guess in $\mathbb{X} - \textsc{Propagate}^*(V)$ when $\mathbb{K} \subseteq V$. The problem remains to (efficiently) find *insufficient* sets of variables V such that $\textsc{Propagate}^*(V)$ is as big as possible. At each step of the DAG exploration, there is a tradeoff between spending more time pruning the graph and spending more time exploring it. We have found a simple greedy heuristic to be quite successful to build a good set V. It is shown in Algorithm 8.2, along with other the pruning strategies we have implemented.

Algorithm 8.2 Pruning Strategies for Algorithm 8.1.

```
1: function GreedyGlobalPruning(V)
2:     Find variable x ∈ 𝕏 − V such that |Propagate*(V ∪ {x})| is minimal
3:     if Propagate*(V ∪ {x}) = 𝕏 then return V
4:     return GreedyGlobalPruning(V ∪ {x})
5: end function
6: function LocalPruning(𝕂, V)
7:     Bad ← ∅
8:     for all x ∈ V do
9:         if x ∉ Bad then
10:            for all y ∈ Propagate*(𝕂 ∪ {x}) do
11:                if y ≠ x then Bad ← Bad ∪ {y}
12:            end for
13:        end if
14:    end for
15:    return V − Bad
16: end function
17: function FilterGuesses(𝕂)
18:    Candidates ← GreedyGlobalPruning(𝕂)
19:    for all x ∈ Candidates do
20:        if there exist y ∈ 𝔾 such that x <_𝕏 y then Candidates ← Candidates − {x}
21:    end for
22:    return LocalPruning(Candidates)
23: end function
```

Linearly Occurring Variables. Since the equations $\mathbb{E}$ describing a good cryptographic primitive cannot be completely linear, then some variables appear both linearly, and under the S-box. However, some variables may appear only linearly (this is for instance the case of the last round key in the AES). If a variable x occurs only linearly, then it can be eliminated from all the equations except one by taking linear combinations of the other equations. Taking apart the single equation containing x, we obtain a new system of equations $\mathbb{E}'$ with one less equation and one less variable. The search procedure can safely be run on $\mathbb{E}'$.

8.2.4 Computing and Testing Solutions

If $x_i \in \textsc{Propagate}(\mathbb{K})$, then there exist a vector α_i such that $[\mathbb{E} \cdot \alpha_i] = [x_i]$ (resp. $[S(x_i)]$), where the square brackets again denotes the equivalence class in the quotient by $\mathcal{V}(\mathbb{K})$. The vector α_i can be computed using straightforward linear algebra given x_i and $\mathbb{K}$, as mentioned in §8.2.1. Once a sufficient set $\mathbb{G}$ has been found, and all the vectors α_i have been computed, we consider the subspace P spanned by $\mathbb{E} \cdot \alpha_i$ for all vectors α_i corresponding to "propagated" variables in $\mathbb{X} - \mathbb{G}$. All the equations belonging to this subspace P are satisfied by definition once the variables in $\mathbb{X} - \mathbb{G}$ are "determined" from those (in $\mathbb{G}$) whose values have been guessed.

It is therefore interesting to consider a supplementary C of P in $\mathcal{V}(\mathbb{E})$: it describes equations that are (linearly) independent from those used for the "determine" step of the attack. To check whether a given choice of values for the guessed variables is correct, it suffices to a) determine the values of the other variables and b) check whether the equations in C hold. The complexity of the resulting procedure is roughly $256^{|\mathbb{G}|}$ encryptions.

8.2.5 Implementation Details

Writing a proof-of-concept implementation of Algorithm 8.1 is not very complicated, but writing an *efficient* version thereof is a bit more challenging. The only non-trivial part in the implementation of the search procedure is the Propagate function. To make it efficient, we exploited the sparsity of the equations $\mathbb{E}$. Recall from §8.2.1 that Propagate tries to solve the equation in α:

$$[\mathbb{E} \cdot \alpha] = [x] \tag{8.1}$$

The problem boils down to solving systems of linear equations. This is much more efficient if the equations come in some kind of triangular form. We therefore first echelonize the equations $\mathbb{E}$, and then use a sparse triangular solver with sparse right-hand-side, *i.e.*, a sparse linear algebra subroutine that solves $A \cdot x = y$

when A is triangular and sparse, and y is also sparse. The interest of this procedure is that it may perform sensibly less than n operations when A and y are sparse enough (see [Dav06] for more details).

The quotient operation in fact *removes rows* from the matrices and the vectors it is applied to. So, when a new variable x becomes "known", we have to remove the rows x and $S(x)$ from the matrix representing the equations. If one of these rows was pivotal, then removing it may leave the matrix in a non-echelonized state. This can be fixed through a simple column permutations in some cases. In some other cases, a new column has to be recomputed, using a variant of the sparse triangular solver. All in all, removing rows and re-echelonizing the matrix represent a negligible fraction of the running time, because the matrix representing $\mathbb{E}$ is stored in a special sparse data-structure: non-zero entries are stored column-wise and row-wise in doubly-linked lists. This allows to efficiently remove rows and columns. Removed entries are kept in memory, and can be efficiently restored when backtracking. An array stores the pivot column for each (pivotal) row. A useful optimization follows the from the observation that equation (8.1) only has a solution if x (resp. $S(x)$) is a pivotal row in the matrix.

The code has been written in the OCaml language, and weights about 5000 lines, more than 1500 of them devoted to the linear algebra. Debugging the sparse linear algebra subroutines was a bit challenging because of the unusual data-structure holding the matrix. Parallelizing the DAG exploration is not difficult, and we developed a distributed version of Algorithm 8.1 using a customized version of the MapReduce framework [DG10a] built on top of Leroy's OcamlMPI library (and building on ideas by Filliâtre and Kalyanasundaram [FK11]). We used it to run our program on two types of platform:

- Roughly 100 Intel cores of various speeds (between 2 and 3 Ghz) in parallel using all the desktop computers of the lab during the night.
- 400 MIPS-like cores in a server containing 8 Tilera TilePRO64 CPUs with 50 available cores each (unfortunately, the OCaml compiler cannot generate native MIPS assembly, and hence generated bytecode. Interpreting the bytecode causes a tenfold performance penalty).

On the second platform, exploring the graph for one full round takes about a minute. For 1.5 rounds, it takes 18 minutes (and finds an attack with 7 guessed bytes). For 2 full rounds, it takes 67 minutes. For 2.5 rounds, it takes 3 weeks. We did not have the patience to wait a few weeks for the exhaustive search to terminate on 3 rounds (a solution faster than exhaustive search was found, but we have no guarantee that it is the fastest attack of this class of attacks). At the very least, we hope that we demonstrated that it is possible to parallelize the search process at will.

8.2.6 Limitations

The main limitation of this approach is that it completely fails to take into account the differential properties of the S-box. For instance, it cannot exploit the fact that when the input and output differences of the S-box are fixed and non-zero, then at most 4 possible input values are possible. Therefore, this approach alone does not bring useful results when more than one plaintext is available. However, it can be used as a sub-component in a more complex technique. We now move on to describe such a generalization that allows to find more powerful attacks.

8.3 An Improved Tool for Meet-In-The-Middle Attacks

The equations describing the AES enjoy an interesting and important property. Let us consider a partition of the set of variables, $\mathbb{X} = \mathbb{X}_1 \cup \mathbb{X}_2$. Then any equation $f \in \mathbb{E}$ can be written $f = f_1 + f_2$, with $f_1 \in \mathcal{V}(\mathbb{X}_1)$ and $f_2 \in \mathcal{V}(\mathbb{X}_2)$. In some sense, these equations are *separable*. We will see that this allows a "divide-and-conquer" meet-in-the-middle approach.

8.3.1 Solving Subsystems Recursively

The simple algebraic structure of the equations allows us to efficiently extract from a system $\mathbb{E}$ a *subsystem* containing only certain variables (say $\mathbb{X}_1$), by simply computing the vector space intersection $\mathbb{E} \cap \mathcal{V}(\mathbb{X}_1)$. In the sequel we will denote it by $\mathbb{E}(\mathbb{X}_1)$. We note that a solution of $\mathbb{E}$ is also a solution of $\mathbb{E}(\mathbb{X}_1)$, for any $\mathbb{X}_1 \subsetneq \mathbb{X}$, but that the converse is not true in general.

Now let us be given a partition $\mathbb{X} = \mathbb{X}_1 \cup \mathbb{X}_2$ and two *black-box solvers* $\mathcal{A}_1$ and $\mathcal{A}_2$ that find all the solutions of $\mathbb{E}(\mathbb{X}_1)$ and $\mathbb{E}(\mathbb{X}_2)$, respectively. We then seek to use the two sub-solvers $\mathcal{A}_1$ and $\mathcal{A}_2$ to find the solutions $\mathcal{S}(\mathbb{E})$ of the full problem. An obvious way is to compute the solutions $\mathcal{S}_1$ of $\mathbb{E}(\mathbb{X}_1)$ and $\mathcal{S}_2$ of $\mathbb{E}(\mathbb{X}_2)$, and to test all the solutions in the Cartesian product $\mathcal{S}_1 \times \mathcal{S}_2$. This would require checking $|\mathcal{S}_1| \cdot |\mathcal{S}_2|$ candidates against the equations.

However, it is possible to do better. First, we observe that the vectors in $\mathcal{S}_1 \times \mathcal{S}_2$ automatically satisfy the equations in $\mathbb{E}(\mathbb{X}_1) + \mathbb{E}(\mathbb{X}_2)$. Therefore we first compute a supplementary of $\mathbb{E}(\mathbb{X}_1) + \mathbb{E}(\mathbb{X}_2)$ inside $\mathbb{E}$ (let us denote it by $\mathcal{M}$). The solutions of $\mathbb{E}$ are in fact the elements of $\mathcal{S}_1 \times \mathcal{S}_2$ satisfying the equations of $\mathcal{M}$.

This already makes less constraints to check. Second, sieving the elements satisfying these constraints can be done in roughly $|\mathcal{S}_1| + |\mathcal{S}_2|$ operations, using variable separation and a table. Let $(f_i)_{1\leq i\leq m}$ be a basis of $\mathcal{M}$, and $f_i = g_i + h_i$ with $g_i \in \mathcal{V}(\mathbb{X}_1)$ and $h_i \in \mathcal{V}(\mathbb{X}_2)$. If the values of all the variables in $\mathbb{X}_1$ (resp. $\mathbb{X}_2$) are available, then the g_i's (resp. h_i) may be evaluated. We denote by G (resp. H) the function that evaluates all the g_i (resp. h_i) on its input. If $\ell = |\mathbb{X}_1|$, then:

$$G : (x_1, \ldots, x_\ell) \mapsto \Big(g_1(x_1, \ldots, x_\ell), \ldots, g_m(x_1, \ldots, x_\ell)\Big)$$

We build two tables:

$$\begin{array}{rcl} L_1 & \longleftarrow & \{(G(x_1), x_1) \mid x_1 \text{ solution of } \mathbb{E}(\mathbb{X}_1)\} \\ L_2 & \longleftarrow & \{(H(x_2), x_2) \mid x_2 \text{ solution of } \mathbb{E}(\mathbb{X}_2)\} \end{array}$$

Then, the solutions of $\mathbb{E}$ are the pairs (x, y) for which there exist a z such that $(z, x) \in L_1$ and $(z, y) \in L_2$. They can be identified efficiently by various methods (sorting the tables, using a hash index, etc.). We have just combined $\mathcal{A}_1$ and $\mathcal{A}_2$ to form a new solver, $\mathcal{A} = \mathcal{A}_1 \bowtie \mathcal{A}_2$, that enumerates the solutions of $\mathbb{E}$.

Complexity of the Combination. Given a partition $\mathbb{X} = \mathbb{X}_1 \cup \mathbb{X}_2$, and two sub-solvers $\mathcal{A}_1$ and $\mathcal{A}_2$, respectively, computing $\mathcal{S}(\mathbb{E}(\mathbb{X}_1))$ and $\mathcal{S}(\mathbb{E}(\mathbb{X}_2))$, the complexity and the properties of $\mathcal{A}_1 \bowtie \mathcal{A}_2$ are easy to determine. Let us denote by $T(\mathcal{A})$ the running time of $\mathcal{A}$, by $M(\mathcal{A})$ its memory consumption, by $V(\mathcal{A})$ the set of variables occurring in the corresponding equations, and by $\mathcal{S}(\mathcal{A})$ the set of solutions of the corresponding equations. The number of operations performed by the combination is the sum of the number of operations produced by the sub-solvers, plus the number of solutions (the time required to scan the tables, namely $|\mathcal{S}_1| + |\mathcal{S}_2|$, is in the worst case of the same order as the running time of the two sub-solvers). However, we use the following approximation

$$T(\mathcal{A}_1 \bowtie \mathcal{A}_2) = \max\big(T(\mathcal{A}_1), T(\mathcal{A}_2), |\mathcal{S}(\mathbb{E}(V(\mathcal{A}_1) \cup V(\mathcal{A}_2)))|\big)$$

It is possible to store only the smallest table, and to enumerate the content of the other "on the fly", while looking for a collision. This reduces the memory complexity to the maximum of the memory complexity of the sub-solvers, and the size of the smaller table. This yields:

$$M(\mathcal{A}_1 \bowtie \mathcal{A}_2) = \max\Big\{M(\mathcal{A}_1), M(\mathcal{A}_2), \min\Big(\big|\mathcal{S}(\mathcal{A}_1)\big|, \big|\mathcal{S}(\mathcal{A}_2)\big|\Big)\Big\}$$

Heuristic Assumption On the Number of Solutions. Evaluating the complexity of a given (possibly recursive) combination requires evaluating the number of solutions of various sub-systems. This is a difficult problem in general, and in order to be able to quickly evaluate the properties of a combination, we use the following *heuristic assumption*:

$$\log_{256}\Big|\mathcal{S}(\mathbb{E}(X))\Big| \approx |X| - \dim \mathbb{E}(X)$$

This heuristic assumption introduces a risk of failure, or of wrong estimation of the complexity. To protect ourselves against this risk, we have tried, when possible, to run an implementation of the solvers and check whether this assumption holds. A difficulty that we encountered in practice stems from the following "differential" system:

$$\begin{array}{rcl} x + y & = & \Delta_i \\ S(x) + S(y) & = & \Delta_o \end{array}$$

If S is the S-box of the AES, then this system has one solution on average (over the random choice of the differences), and the hypothesis holds. However, in degenerate situations, for instance when $\Delta_i = \Delta_o = 0$, then the system has 2^8 solutions... Surprisingly, an S-box with very bad differential properties would make life more difficult for our tool. This follows from the fact that in a good S-box, there are very few pairs of input/output values that generate a given input/output difference, and this makes our assumption more likely to hold in "differential" situations. On the other hand, an S-box with bad differential properties would have many impossible differentials, and this could also be exploited by the tool (for example by assuming that an input and output difference can only coexist with small probability).

In any case, this assumption makes it very easy to evaluate the performance of the combination of two sub-solvers: it boils down to computing a vector-space intersection.

8.3.2 Automatic Search for Recursive Combinations of Solvers

Given a system of equations, we would like to build an efficient solver by breaking the problem down to smaller and smaller subsystems, recursively generating efficient sub-solver for the sub-problems and combining them back.

Note that $\mathbb{E}(\{x\})$ (the intersection of the vector space $\mathbb{E}$ with $\langle x, S(x), 1\rangle$ cannot be further broken down because $\{x\}$ cannot be partitioned anymore. It is a "base case" of the decomposition, and it can be dealt with in two ways:

- Either $\mathbb{E}(\{x\}) = \emptyset$, so that we cannot easily determine how the variable x is constrained by the equations. In that case, the "solutions" of $\mathbb{E}(\{x\})$ are in fact the whole field $\mathbb{F}_{2^8}$.
- Or $\mathbb{E}(\{x\}) \neq \emptyset$, so that we know an equation involving only x and $S(x)$. In that case, the set of all solutions of $\mathbb{E}(\{x\})$ is very likely to have very few elements, which can be precomputed and returned as a list.

The case where a single variable remains is a base case of the decomposition, and we will say that it is dealt with by "base solvers", implementing one of the above strategies. Combining base solvers in various ways yields *solving trees* of various shapes. In such a tree, the leaves are the "base solvers" associated to variables of $\mathbb{X}$, and the nodes are combinations of solvers. A tree describes a valid solver for $\mathbb{E}$ as long as all the variables of $\mathbb{E}$ appear as the leaves of the tree.

Note that the guess-and-determine attacks discussed in the previous section form a particular case of this more general framework. They can be described by a recursive combination where $\mathbb{X}_2$ always contains a single variable. However, it turns out that allowing more general tree shape results in better attacks.

Comparing Solvers. It is always possible to construct several solving trees that solve the same problem in different ways, and sometimes more or less efficiently. We therefore want to be able to *compare* solvers in a meaningful way. We want $\mathcal{A}_1 \succeq \mathcal{A}_2$ if $\mathcal{A}_1$ is overally more interesting (works faster, finds solution of a bigger system). We also want the order relation to be compatible with the combination operation (*i.e.*, $\mathcal{A}_1 \succeq \mathcal{A}_2$ implies $\mathcal{A}_1 \bowtie \mathcal{A}_3 \succeq \mathcal{A}_2 \bowtie \mathcal{A}_3$). We thus define:

$$\mathcal{A}_1 \succeq \mathcal{A}_2 \iff \begin{cases} T(\mathcal{A}_1) & \leq & T(\mathcal{A}_2) \\ V(\mathcal{A}_1) & \supseteq & V(\mathcal{A}_2) \\ |S(\mathcal{A}_1)| & \leq & |S(\mathcal{A}_2)| \end{cases}$$

Just like any other partial order, this order relation induces an equivalence relation:

$$\mathcal{A}_1 \equiv \mathcal{A}_2 \text{ if and only if } \mathcal{A}_1 \succeq \mathcal{A}_2 \text{ and } \mathcal{A}_2 \succeq \mathcal{A}_1.$$

This equivalence relation also carries an interesting meaning: if $\mathcal{A}_1 \equiv \mathcal{A}_2$ then $\mathcal{A}_1$ and $\mathcal{A}_2$ offer essentially the same functionality with the same performance. The equivalence relation is also compatible with the combination operation. We observe that given a set of variables $\mathbb{X}_1$, there can be only one maximal solver (up to equivalence) for $\mathbb{E}(\mathbb{X}_1)$. Thus, our objective is now clearly identified: find a maximal (*i.e.*, the best) solver for $\mathbb{E}$ (up to equivalence).

Exhaustive Search for the Best Recursive Solver. The procedure ExhaustiveSearch in Algorithm 8.3 computes the set of all maximal solvers for all sub-systems of a given system of equations $\mathbb{E}$ (up to equivalence). In particular, it will construct a maximal solver for $\mathbb{E}$ itself. The algorithm is reminiscent of (and inspired by) the Buchberger algorithm for Gröbner bases [Buc65]. More generally Algorithm 8.1 is a saturation procedure, and this also makes it similar to many automated deduction procedures (such a Resolution-based theorem provers or the Knuth-Bendix completion algorithm). At each step, the algorithm maintains a list G of solvers for subsystems of the original system $\mathbb{E}$. It also maintains a list $\mathcal{P}$ of pairs of solver that remain to be processed. When a new solver is found, all the solvers that are worse are removed from G (and all pairs containing it are removed as well). Then, new pairs containing the new solver are scheduled for processing.

The complexity of this algorithm seems difficult to evaluate. It depends on the equations, and on the order in which the combinations are performed. In any case, the size of its ouput is upper bounded by $2^{|\mathbb{X}|}$ (because it will return only one maximal solver for each subset of $\mathbb{X}$). The parameter T_{up} allows the user to enforce an upper-bound on the time complexity of the generated solvers (by discarding the ones that are too slow). For small values of T_{up}, this may for instance allow to prove the non-existence of recursive solvers with complexity lower than a threshold. The running time of the exhaustive search also gets smaller with lower values of T_{up}.

In practice, what dominates the execution of this algorithm is the computation of the dimension of the combination C, and the bookkeeping required to update G and $\mathcal{P}$. A nice improvement is to use the Propagate* function from §8.2.2: each time a new solver C is constructed, we could check whether $V(C)$

is stable by PROPAGATE*. If not, we could combine it with the base solvers in PROPAGATE*$(V(C)) - V(C)$, thus improving it without increasing its running time.

Algorithm 8.3 Exhaustive Search for a good recursive solver

```
1: function UPDATE-QUEUE(G, P, A)
2:     if A' ⊭ A for all A' ∈ G then
3:         G ← {A' ∈ G | A ⊭ A'} ∪ {A}
4:         P ← {(A_i, A_j) ∈ P : A_i ⋠ A and A_j ⋠ A} ∪ {(A, A') : A ≠ A' ∈ G}
5:     end if
6:     return (G, P)
7: end function
8: function EXHAUSTIVESEARCH(E, T_up)
9:     G ← {BaseSolver(x) : x ∈ X}
10:    P ← {(G_i, G_j) : 1 ≤ i < j ≤ |G|}
11:    while P ≠ ∅ do
12:        Pick (A_i, A_j) ∈ P and remove it from P
13:        C ← G_i ⋈ G_j
14:        if T(C) ≤ T_up then (G, P) ← UPDATE-QUEUE(G, P, C)
15:    end while
16:    return G
17: end function
```

Randomized Search. The complexity of the exhaustive search is inherently exponential, and exploring the whole space might not be feasible. In that case, a non-exhaustive randomized search might find good results, without offering the guarantee that they are the best possible. The procedure RANDOMIZEDSEARCH on Algorithm 8.4 shows a possible randomized search that we have found to give good results. The idea is again quite simple: at each step, we choose a random set of variables Y, we build a solver for $\mathbb{E}(Y)$, and if it is not subsumed by any previously known solver, we include it in the current solver list, and we try to combine it with all the solvers we know. It would make sense to choose Y with some care, for instance using the pruning strategies discussed in §8.2.3.

There are many possible other ways to perform such a randomize search: Choose the size of the random subsets of $\mathbb{X}$ according to some distribution, periodically restart the procedure, periodically flush "bad" solvers from G, run the exhaustive search for a while, fill G, then switch to randomized search, etc. This presently seems to be more of an art than a science.

Algorithm 8.4 Randomized Search for a good recursive solver

```
1: function RANDOMIZEDSEARCH(E, T_up)
2:     G ← ∅
3:     P ← ∅
4:     loop
5:         Y ← random subset of X, of size T_up
6:         (x_{i_1}, x_{i_2}, ..., x_{i_ℓ}) ← PROPAGATE*(Y)
7:         B ← BaseSolver(x_{i_1}) ⋈ BaseSolver(x_{i_2}) ⋈ ... ⋈ BaseSolver(x_{i_ℓ})
8:         (G, P) ← UPDATE-QUEUE(G, P, B)
9:         while P ≠ ∅ do
10:            Pick (A_i, A_j) ∈ P and remove it from P
11:            C ← A_i ⋈ A_j
12:            if T(C) > T_up then drop C
13:            if V(C) = X then return C
14:            (G, P) ← UPDATE-QUEUE(G, P, C)
15:        end while
16:    end loop
17: end function
```

8.3.3 Usage

Algorithms 8.3 and 8.4 have been developed and implemented in C by Patrick Derbez. The running time is dominated by the computation of the time-complexity of a combination of solvers, which involves computing

the dimension of a vector-space intersection. Various tricks can also be used to speed this operation up (using a sparse representation, precomputing partially echelonized forms, not computing an intersection but a sum, etc. The program is 10'000 lines long, the majority of which is dedicated to linear algebra subroutines.

When an interesting solver for $\mathbb{E}$ is found by the search procedure, it is not particularly complicated to recursively generate a C++ implementation thereof (*i.e.*, a function that takes as input the "known" variables, and returns the solutions of the system of equations), or a text file that describes which variables to enumerate, which tables to join, in a nearly human-readable language. The generated C++ files typically have $\geq$ 10k lines, with thousands of more-or-less dummy functions. They seem to be good torture tests for current compilers...

We emphasize again that this method is strictly more general than that presented in the previous section, because any attack that could be discovered by the preliminary tool can also be found by Algorithms 8.3 and 8.4. The next chapter show multiple examples of attacks found by both tools.

Chapter 9

A Collection of Low Data Complexity Attacks on AES-Derivatives

We present a collection of low-data complexity attacks on round-reduced versions of the AES, as well as attacks on several other cryptographic algorithms derived from the AES. These attacks have been found in the course of a joint work with Derbez, Dunkelman, Keller and Rijmen, during which the automated tools of the previous chapter have been developed. Some of these results led to the submission of a journal paper [BDD+10], while some others were presented in [BDF11].

During the course of 2009 and 2010, Dunkelman and Keller announced in several occasions that they were investigating low-data complexity attacks against the AES, and announced interesting results. We developed the automated tools described in the previous chapter hoping to catch up on their results. This effort has been gratifying, as the tools could often improve on the manually-found attacks. When it is the case, it is interesting to compare both the manually-found and the automatically-found attacks. Our results on round-reduced versions of the AES are summarized in Table 9.1.

Because our tools are somewhat generic, they are not restricted to the AES as a block cipher, and we used them to find new attacks on the message authentication code Pelican-MAC [DR05b], and to the stream cipher LEX [Bir05]. The tools found the fastest known attacks on these two constructions, again a gratifying result. This demonstrates in a concrete way that low-data complexity attacks can be leveraged into actual attacks on full versions of some primitives. Our results on AES-based primitives are summarized in Table 9.2.

This chapter is organized as follows: several useful observations on the AES are presented in §9.1. Then, attacks on one, two, three and four AES rounds are given in §9.2, §9.3, §9.4, and §9.5 respectively. A new attack on 6 AES rounds in the known-plaintext model is given in §9.6. The best known attack on Pelican-MAC is given in §9.7, and the best known attack against LEX is given in §9.8.

9.1 Observations on the Structure of AES

In this section we present well-known observations on the structure of AES, that we use in our attacks. We first consider the propagation of differences through `SubBytes`, which is the only non-linear operation in AES.

Property 9.1 (the `SubBytes` property)**.** *Consider pairs $(\alpha \neq 0, \beta)$ of input/output differences for a single S-box in the `SubBytes` operation. For 129/256 of such pairs, the differential transition is impossible, i.e., there is no pair (x, y) such that $x \oplus y = \alpha$ and $S(x) \oplus S(y) = \beta$. For 126/256 of the pairs (α, β), there exist two ordered pairs (x, y) such that $x \oplus y = \alpha$ and $S(x) \oplus S(y) = \beta$, and for the remaining 1/256 of the pairs (α, β) there exist four ordered pairs (x, y) that satisfy the input/output differences. Moreover, the pairs (x, y) of actual input values corresponding to a given difference pattern (α, β) can be found instantly from the difference distribution table of the S-box. We recall that the time required to construct the table is 2^{16} evaluations of the S-box, and the memory required to store the table is about 2^{17} bytes.*

The second observation uses the linearity of the `MixColumns` operation, and follows from the structure of the matrix used in `MixColumns`:

Property 9.2 (the `MixColumns` property)**.** *Consider a pair (a, b) of 4-byte vectors, such that $a = MC(b)$, i.e., the input and the output of a `MixColumns` operation applied to one column. Denote $a = (a_0, a_1, a_2, a_3)$ and $b = (b_0, b_1, b_2, b_3)$ where a_i and b_j are elements of $\mathbb{F}_{2^8}$. The knowledge of* any *four out of the eight bytes $(a_0, a_1, a_2, a_3, b_0, b_1, b_2, b_3)$ is sufficient to* uniquely *determine the value of the remaining four bytes.*

The third observation is concerned with the key schedule of AES, and exploits the fact that most of the operations in the key schedule algorithm are linear. It allows the adversary to get relations between bytes of non-consecutive subkeys (*e.g.*, k_r, k_{r+3} and k_{r+4}), while "skipping" the intermediate subkeys. The observation extends previous observations of the same nature made in [FKL+00, DK10a].

Attacks on round reduced version of the AES-128					
	Complexity				
#Rounds	Data	Time	Memory	Description	Found by
1	1 KP	2^{48}	1	[DK10b]	human beings
1	1 KP	2^{40}	1	§9.2.2	preliminary tool
1	1 KP	2^{32}	2^{24}	§9.2.2	improved tool
1	1 KP	2^{32}	2^{16}		improved tool
1.5	1 KP	2^{56}	1		preliminary tool
1.5	2 KP	2^{24}	2^{16}		improved tool
2	1 KP	2^{80}	1	§9.3.4	human beings,preliminary tool
2	1 KP	2^{64}	2^{48}	§9.3.5	improved tool
2	2 KP	2^{48}	1	§9.3.1	human beings
2	2 KP	2^{32}	2^{24}	§9.3.1	improved tool
2	3 KP	2^{32}	1	§9.3.2	human beings
2	2 CP	2^{28}	1	§9.3.3	human beings
2	2 CP	2^{8}	2^{8}	§9.3.3	improved tool
2.5	1 KP	2^{96}	1		preliminary tool
2.5	1 KP	2^{88}	2^{88}		improved tool
2.5	2 KP	2^{80}	2^{80}		improved tool
2.5	2 CP	2^{24}	2^{16}		improved tool
3	1 KP	2^{120}	1	§9.4.4	preliminary tool
3	1 KP	2^{96}	2^{96}		improved tool
3	9 KP	2^{40}	2^{35}	§9.4.3	human beings, improved tool
3	2 CP	2^{32}	1	[BDD+10]	human beings
3	2 CP	2^{16}	2^{8}	§9.3.3	improved tool
4	1 KP	2^{120}	2^{120}		improved tool
4	2 CP	2^{104}	1	[BDD+10]	human beings
4	2 CP	2^{80}	2^{80}		improved tool
4	5 CP	2^{64}	2^{68}	§9.5.2	human beings
4	4 CP	2^{32}	2^{24}	§9.5.3	improved tool
4	10 CP	2^{32}	2^{24}	§9.5.1	human beings
4.5	1 KP	2^{120}	2^{120}		improved tool

KP — Known plaintext, CP — Chosen plaintext,
Time complexity is measured in approximate encryption units.
Memory complexity is measured approximately

Table 9.1: Summary of our Proposed Attacks on AES-128

Property 9.3 (the key-schedule properties)**.** *Consider a series of consecutive subkeys $k_r, k_{r+1}, \ldots$, and denote $k_r = (a, b, c, d)$ and:*

$$\begin{aligned} u &= \mathsf{RotBytes}(\mathsf{SubBytes}(k_r[12..15])) \oplus RCON[r+1] \\ v &= \mathsf{RotBytes}(\mathsf{SubBytes}(k_{r+1}[12..15])) \oplus RCON[r+2] \\ w &= \mathsf{RotBytes}(\mathsf{SubBytes}(k_{r+2}[12..15])) \oplus RCON[r+3] \\ x &= \mathsf{RotBytes}(\mathsf{SubBytes}(k_{r+3}[12..15])) \oplus RCON[r+4] \end{aligned}$$

Then, the subkeys $k_{r+1}, k_{r+2}, \ldots$ can be represented as linear combinations of (a, b, c, d) (the columns of k_r) and the 32-bit words u, v, w, x, as shown in the following table:

Round	$k[0..3]$	$k[4..7]$	$k[8..11]$	$k[12..16]$
r	a	b	c	d
$r+1$	$a \oplus u$	$a \oplus b \oplus u$	$a \oplus b \oplus c \oplus u$	$a \oplus b \oplus c \oplus d \oplus u$
$r+2$	$a \oplus u \oplus v$	$b \oplus v$	$a \oplus c \oplus u \oplus v$	$b \oplus d \oplus v$
$r+3$	$a \oplus u \oplus v \oplus w$	$a \oplus b \oplus u \oplus w$	$b \oplus c \oplus v \oplus w$	$c \oplus d \oplus w$
$r+4$	$a \oplus u \oplus v \oplus w \oplus x$	$b \oplus v \oplus x$	$c \oplus w \oplus x$	$d \oplus x$

Attacks on Primitives based on AES					
Primitive	Complexity			Where	Found by
	Data	Time	Memory		
Pelican-MAC	$2^{85.5}$ queries	$2^{85.5}$	$2^{85.5}$	[YWJ+09]	human beings
Pelican-MAC	2^{64} queries	2^{64}	2^{64}	§9.7	improved tool
Pelican-MAC	2^{64} queries	2^{64}	2^{64}	§[DKS11]	human beings
Alpha-MAC	2^{65} queries	2^{64}	2^{64}	[YWJ+09]	human beings
LEX	$2^{36.3}$ bytes	2^{112}	2^{36}	[DK08]	human beings
LEX	2^{40} bytes	2^{100}	2^{64}	[DK10a]	human beings
LEX	$2^{36.3}$ bytes	2^{96}	2^{80}	§9.8.1	improved tool
LEX	2^{48} bytes	2^{88}	2^{48}	§9.8.2	improved tool
LEX	2^{48} bytes	2^{80}	2^{48}		improved tool
modified LEX	360 bytes	2^{40}	2^{43}	§9.8.3	human beings
AES-128	1 fault	2^{32}	2^{32}	[PQ03]	human beings
AES-128	1 fault	2^{24}	2^{16}	§9.4.2	improved tool

Time complexity is measured in approximate encryption units.
Memory complexity is measured approximately

Table 9.2: Summary of our Proposed Attacks on Primitives based on AES

As a result, we have the following useful relations between subkeys

i) $k_{r+2}[0..3] \oplus k_{r+2}[8..11] = k_r[8..11]$,
ii) $k_{r+2}[4..7] \oplus k_{r+2}[12..15] = k_r[12..15]$,
iii) $k_{r+2}[4..7] \oplus v = k_r[4..7]$,
iv) $k_{r+4}[12..15] \oplus x = k_r[12..15]$,
v) $k_{r+3}[12..15] = k_r[8..11] \oplus k_r[12..15] \oplus w$.

9.2 Attacks on One-Round AES

We start our analysis with the simplest case, an adversary who seeks to break one full round of AES (a sequence of `AddRoundKey`, `SubBytes`, `ShiftRows`, `MixColumns`, and `AddRoundKey` operations).

9.2.1 Two Known Plaintexts

We first describe a simple but suboptimal attack. It starts by applying $SR^{-1} \circ MC^{-1}$ to the ciphertext difference, to obtain the output differences of all the S-boxes. Since the input differences of the S-boxes are equal to the plaintext difference in the respective bytes, the adversary can consider each S-box independently, go over the 2^8 possible pairs of inputs whose difference equals the plaintext difference, and find the pairs suggesting the "correct" output difference. In each S-box, the expected number of suggested pairs is two, and each such pair gives a suggestion of one byte in the subkey k_0.[1] Thus, the adversary gets 2^{16} suggestions for the entire subkey k_0, which can be checked by trial encryption.

This attack, whose time complexity is 2^{16} encryptions, can be further improved using the relation between the subkeys k_0 and k_1. If the adversary checks the S-boxes in bytes $0, 5, 10$, and 15, she can use the $2^4 = 16$ suggestions of output values of these S-boxes to get 16 suggestions for the column $k_1[0..3]$, along with bytes $0, 5, 10, 15$ of k_0. Similarly, checking bytes $3, 4, 9, 14$ yields 16 suggestions for the column $k_1[4..7]$, along with bytes $3, 4, 9, 14$ of k_0. Combining the suggestions, the adversary obtains 256 suggestions for two columns of k_1 and eight bytes of k_0. At this stage, the adversary can use the relation $k_1[4] = k_0[4] \oplus k_1[0]$, which holds by the AES key schedule, as a consistency check. Only a single suggestion is expected to remain. The value of the remaining 8 bytes of k_0 can be obtained similarly by examining the other eight S-boxes. This improvement reduces the time complexity of the attack to 2^{12} S-box applications.

9.2.2 One Known Plaintext

If the data available to the adversary is only a single plaintext, then the attack must use the relation between the two subkeys k_0 and k_1. If the subkeys were independent, then the information available to the

1. We note that this step can be performed only if the differences in all S-boxes are non-zero. However, since in the known plaintext attack model it is common to assume that the plaintexts are chosen at random, it is expected that two known plaintexts have non-zero difference in all the 16 bytes with probability $(255/256)^{16} = 0.939$.

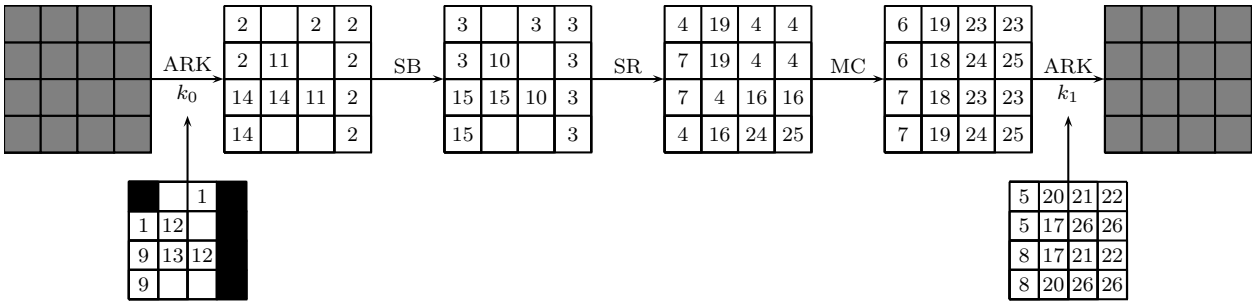


Figure 9.1: A 2^{40} time attack on one round AES given one known plaintext. Bytes marked by black are guessed, bytes marked by gray are known.

adversary would not be sufficient to retrieve the key uniquely. Since any relation between the plaintext and the ciphertext involves the `MixColumns` operation, it seems likely that any such attack should require the guess of a full column, and thus have complexity of at least 2^{32} encryptions. [2]

In this setup, the best attack found manually is a guess-and-determine attack by Dunkelman and Keller [DK10b] that has a running time equivalent to that of 2^{48} encryptions. The preliminary tool of §8.2 found a faster attack with time complexity of 2^{40} encryptions and a negligible memory requirement, while the tool of §8.3 found an even faster attack with time complexity of 2^{32} encryptions, and memory requirement of 2^{25} bytes. Thus, it seems that we are closing on to the optimum for this variant.

9.2.2.1 The Preliminary Tool's Attack

The first attack, depicted in Figure 9.1, is based on Property 9.4 below. The adversary guesses five bytes of the subkey k_0 — the column $k_0[12..15]$ as well as an additional byte, $k_0[0]$. The steps in the key derivation are shown in the figure, where the number in each cell denotes the step in which its value is retrieved. Steps 1 and 13 are based on Property 9.4, steps $5, 9, 17, 21$, and 22 exploit the key schedule, steps $7, 19, 24$, and 25 are based on Property 9.2, and the rest of the steps are performed using the application of known operations on known values.

Discovering Property 9.4 below is the key step in finding the attack. Given this observation, (human) cryptanalysts would have no problem completing the attack. Yet finding the Property was difficult, and was only "achieved" by the preliminary tool (we mere humans had a hard time figuring out what was going on). The Property shows that the knowledge of a single column in one of the subkeys, along with the input and output values of the round, allows the retrieval of the value of a few "unexpected" additional subkey bytes.

Property 9.4. *The knowledge of P, C and the column $k_0[12..15]$ allows to retrieve two additional bytes of k_0, namely $k_0[1]$ and $k_0[8]$.*

The main idea behind this observation is that the linearity of the `MixColumns` operation interacts with the almost linear key schedule of the AES. This allows to apply the `MixColumns` operation to *the XOR of two columns*, and to move backwards one round in the key-schedule where the same column-XOR appears.

Proof. The derivation of the two additional bytes is performed as follows. Consider the 32-bit value $w[8..11] \oplus w[12..15]$. By the key schedule, we have:

$$\begin{aligned} w[8..11] \oplus w[12..15] &= \Big(C[8..11] \oplus C[12..15]\Big) \oplus \Big(k_1[8..11] \oplus k_1[12..15]\Big) \\ &= \Big(C[8..11] \oplus C[12..15]\Big) \oplus k_0[12..15]. \end{aligned} \tag{9.1}$$

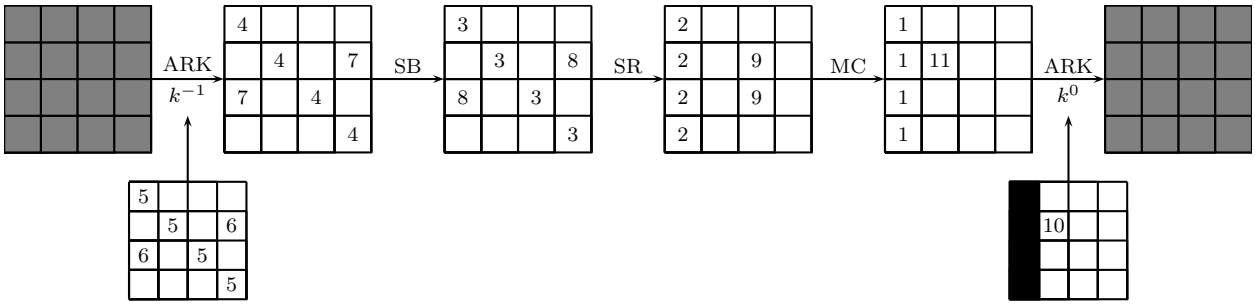
Following the linearity of `MixColumns`, we obtain that:

$$\begin{aligned} z[8..11] \oplus z[12..15] &= MC^{-1}(w[8..11]) \oplus MC^{-1}(w[12..15]) \\ &= MC^{-1}\Big(w[8..11] \oplus w[12..15]\Big). \end{aligned} \tag{9.2}$$

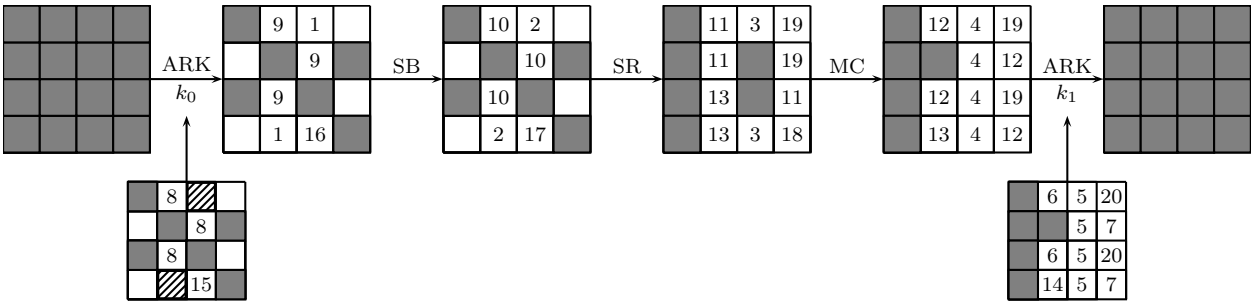
It is possible to compute $z[12]$ using the knowledge of $k_0[12]$ as:

$$z[12] = S(P[12] \oplus k_0[12]). \tag{9.3}$$

2. We note that the problem of attacking one round AES without the MixColumns operation with a single known plaintext is studied in [DK10b]. It is shown that 2^{16} encryptions are sufficient to retrieve the key.



(a) First half of the attack



(b) Second half of the attack

Figure 9.2: An attack on one round AES given one known plaintext with time complexity of 2^{32} and memory complexity of 2^{24}. Bytes marked by gray are known (from previous steps of analysis). Bytes marked with tilted lines have at most 12 possible values by Property 9.5.

Then, the value of $z[8]$ can be retrieved by combining Equations (9.1), (9.2) and (9.3). From $z[8]$ it is possible to compute the subkey byte $k_0[8]$ using:

$$k_0[8] = P[8] \oplus x[8] = P[8] \oplus S^{-1}(z[8]).$$

It is possible to use a similar procedure to obtain the value $z[13]$, and retrieve the subkey byte $k_0[1]$ as:

$$k_0[1] = P[1] \oplus x[1] = P[1] \oplus S^{-1}(z[13]).$$

□

We note that if the adversary knows also the value $k_0[2]$, a similar strategy allows her to retrieve the value $k_0[6]$. This derivation is used in the attack described just above. We also note that similar statements hold if the adversary knows $k_0[4..7]$ or $k_0[8..11]$.

9.2.2.2 The Improved Tool's Attack

The second attack, depicted in Figure 9.2, is based on Property 9.5 below. In the first phase of the attack, the adversary guesses the column $k_1[0..3]$, and retrieves the value of seven additional subkey bytes. This phase is shown in Figure 9.2a, where steps 6 and 10 are based on key schedule arguments, and the rest of the steps use the application of known operations on known values. The second phase of the attack, depicted in Figure 9.2b, starts with retrieving the possible values of bytes $k_0[7]$ and $k_0[8]$ using Property 9.5. Steps $6, 7, 8$, and 15 use the key schedule, steps 13 and 19 use Property 9.2, and the rest of the steps follow the application of AES' operations to known values.

Again, the key step of the attack is the discovery of Property 9.5. It is even more sophisticated than the previous one and was "found" by the improved tool of §8.3. We went though the output produced by the tool to write a readable description of the solver's steps.

Property 9.5. *The knowledge of P, C, and the column $k_1[0..3]$ allows one to retrieve bytes $k_0[7]$ and $k_0[8]$ by a table look-up to a precomputed table of size 2^{24}. For each value of $k_1[0..3]$, there are at most 12 values of $(k_0[7], k_0[8])$, and on average a single value. The time complexity required to generate the table is 2^{32} operations.*

The proof of this property is slightly more complex than that of the previous property, and is based on obtaining two nonlinear 8-bit relations involving two key bytes (given that other bytes of the key and the state are known). In such a case, we expect on average one solution to these equations (and as our test shows, in reality the maximal number of solutions is 12). Moreover, as there are 2^{32} possible systems, one can precompute the acceptable solutions and store them.

Proof. First, we note that the knowledge of the column $k_1[0..3]$ along with the plaintext and the ciphertext allows us to retrieve one additional byte of k_1 and six bytes of k_0, as shown in Figure 9.2a.

We denote $a = y[8]$ and $b = y[7]$, and express several other bytes in terms of a, b, and known bytes (from the plaintext, the ciphertext and $k_1[0..3]$). Our goal is to obtain a system of two equations in a and b, and to solve it using a precomputed table. This system is constructed in 5 steps.

1. First, note that the third column of k_1 can be expressed as:

$$\begin{aligned} k_1[8..11] &= C[8..11] \oplus w[8..11] \\ &= C[8..11] \oplus MC(z[8..11]) \\ &= C[8..11] \oplus MC\Big(a, S(P[13] \oplus k_0[13]), S(P[2] \oplus k_0[2]), b\Big). \end{aligned} \tag{9.4}$$

   In this expression, bytes 2 and 13 of k_0 can be deduced from the 4 guessed bytes in k_1 after the first phase of the attack, as shown in Figure 9.2a.

2. Let us now turn our attention towards the second column of k_1. Note that $k_1[5]$ can be computed following the procedure shown in Figure 9.2a, and the remaining three bytes of $k_1[4..7]$ can be expressed as:

$$\begin{cases} k_1[4] &= k_1[8] \oplus S^{-1}(a) \oplus P[8], \\ k_1[6] &= k_1[10] \oplus k_0[10], \\ k_1[7] &= k_1[3] \oplus S^{-1}(b) \oplus P[7]. \end{cases} \tag{9.5}$$

   Again, $k_0[10]$ (appearing in the expression of $k_1[6]$) can be derived from the guessed bytes.

3. Next, we will turn our attention away from the key-schedule, to the second column of z, and in particular to $z[4]$ and $z[5]$. We first express it in function of the plaintext. It follows from the definition of the encryption and schedule algorithm that:

$$\begin{cases} z[4] &= S\Big(P[4] \oplus k_0[4]\Big) = S\Big(P[4] \oplus k_1[0] \oplus k_1[4]\Big) \\ z[5] &= S(P[9] \oplus k_0[9]) = S(P[9] \oplus k_1[9] \oplus k_1[5]), \end{cases} \tag{9.6}$$

   It must be noted that all key bytes occurring in these expression can be readily derived from a, b and the four guessed key bytes.

4. On the other hand, the whole column $z[4..7]$ can be expressed as a function of the ciphertext:

$$z[4..7] = MC^{-1}(w[4..7]) = MC^{-1}\Big(C[4..7] \oplus k_1[4..7]\Big). \tag{9.7}$$

5. Identifying $z[4]$ and $z[5]$ in (9.6) and (9.7), and exploiting (9.4) and (9.5) results in a system of two equations in the unknowns a, b, and the known plaintext, ciphertext, and bytes that can be derived from $k_1[0..3]$. These equations can be written in the form:

$$\begin{cases} \Delta_1 &= f_1(a,b) \oplus S(f_2(a,b) + \Delta_3), \\ \Delta_2 &= f_3(a,b) \oplus S(f_4(a,b) + \Delta_4), \end{cases} \tag{9.8}$$

   where f_1, f_2, f_3, and f_4 are fixed known functions, and $\Delta_1, \Delta_2, \Delta_3$, and Δ_4 are one-byte parameters depending on the plaintext, the ciphertext, and the guessed subkey bytes. The actual expressions are not given, as they are a bit lengthy, but they can be found in [Der10].

Equation (9.8) allows to perform the following two-phase procedure:

Offline Phase: for each of the 2^{32} possible values of $(a, b, \Delta_3\Delta_4)$, evaluate the right-hand in (9.8), and find the values of Δ_1 and Δ_2. Store the pair (a, b) in a data-structure (typically an array of linked lists, or a hash table) indexed by $(\Delta_1, \Delta_2, \Delta_3, \Delta_4)$. It turns out that the maximal number of solutions for given values of $(\Delta_1, \Delta_2, \Delta_3, \Delta_4)$ is 12, and the average number is 1. Hence, we can construct a table of the solutions in 2^{32} time and 2^{32} memory.

Online phase: once the values of P, C are known, and for each guess of $k_1[4..7]$, compute the value $(\Delta_1, \Delta_2, \Delta_3\Delta_4)$, and obtain from the precomputed table the corresponding values of (a, b). Then deduce the subkey bytes $k_0[7]$ and $k_0[8]$ using the equations:

$$\begin{aligned} k_0[8] =& P[8] \oplus x[8] = P[8] \oplus S^{-1}(a), \\ k_0[7] =& P[7] \oplus x[7] = P[7] \oplus S^{-1}(b). \end{aligned} \tag{9.9}$$

Reducing the Memory Complexity: We can reduce the 2^{32} memory required for storing the solutions to only 2^{24}. This is done by fixing Δ_1, such that we can deal only with the equations relevant to this specific Δ_1 in each step of the attack. The specific expression of Δ_1 makes it easy to enumerate the 2^{24} possible first columns of k_1 leading to this specific value of Δ_1. The outcome of such an approach is the ability to reduce the number of possible systems that we need to consider in a given time to 2^{24}, which reduces the memory complexity without affecting the time complexity. Going into the details would require giving the actual expressions of the Δ's and the f's, which the interested reader will find in [Der10]. □

9.3 Attacks on Two-Round AES

In this section we consider attacks on two rounds of AES, denoted by rounds 1 and 2. First we present attacks on two *full* rounds with two known plaintexts. We then study the interesting case of two *chosen* plaintext. In both settings, the tools vastly outperformed human cryptanalysts. We then look at the case of a single known plaintext. We conclude by presenting an improved attack with two known plaintexts that can be applied if the `MixColumns` operation in round 1 is omitted (*i.e.*, if we are facing 1.5 rounds). This attack is used as a procedure in our attack on 6-round AES presented in §9.6.

9.3.1 Two Known Plaintexts

The Manually-Found Attack. We first describe an attack found manually by Dunkelman and Keller. This attack with two known plaintexts, depicted in Figure 9.3, is based on Property 9.1. As in the one-round attack with two known plaintexts, we observe that the ciphertext difference allows us to retrieve the intermediate difference after the `SubBytes` operation of round 2. This observation is used in both phases of the attack. We also "swap" the order of the `MixColumns` and the `AddRoundKey` operations of the second round. This can be done since both operations are linear, as long as the subkey k_2 is replaced by the equivalent subkey $u_2 = MC^{-1}(k_2)$.

In the first phase of the attack, the adversary guesses bytes $0, 5, 10, 15$ of k_0, which allows her to retrieve the intermediate difference in $x_2[0..3]$ (*i.e.*, just before the `SubBytes` operation of round 2). Then, Property 9.1 can be applied to the four S-boxes in that column, yielding their actual input/output values in both encryptions. This in turn allows us to obtain $k_1[0..3]$ (as the values before the `AddRoundKey` with k_1 are known). At this stage, the adversary tries to deduce and compute as many additional bytes as she can.

In the second phase of the attack, the adversary guesses two additional subkey bytes ($k_0[7]$ and $k_0[8]$) which are sufficient to retrieve the intermediate difference in $x_1[8..11]$. Then, Property 9.1 can be applied to the four S-boxes in bytes 8–11 of round 2.

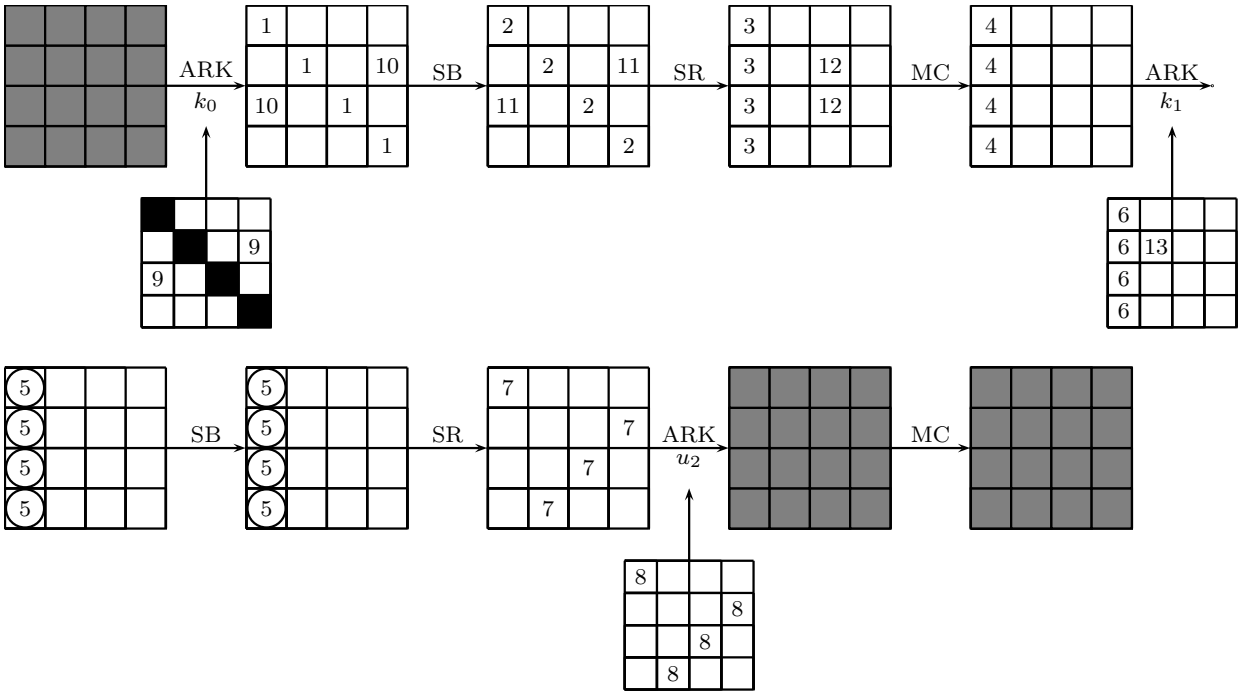
We note that the while the first phase of the attack allows us to obtain several bytes in u_2, the knowledge of these bytes cannot be combined directly with the knowledge of bytes in k_1 and k_0, since u_2 does not satisfy the equations of the key schedule algorithm. Hence, in the second phase of the attack, we obtain bytes in both u_2 and k_2 in parallel, and apply Property 9.2 to the relation between k_2 and u_2, since they are the input and output of a MixColumns operation.

In Phase 1 of the attack, depicted in the top half of Figure 9.3, step 5 is based on Property 9.1, steps 9 and 13 exploit the key schedule, and the rest of the steps are performed using encryption/decryption. In the second phase, depicted in the bottom half of the figure, step 5 is based on Property 9.1, step 12 uses Property 9.2 applied to the relation between k_2 and u_2, steps $9, 10, 11$, and 13 exploit the key schedule, and the rest of the steps are performed using encryption/decryption.

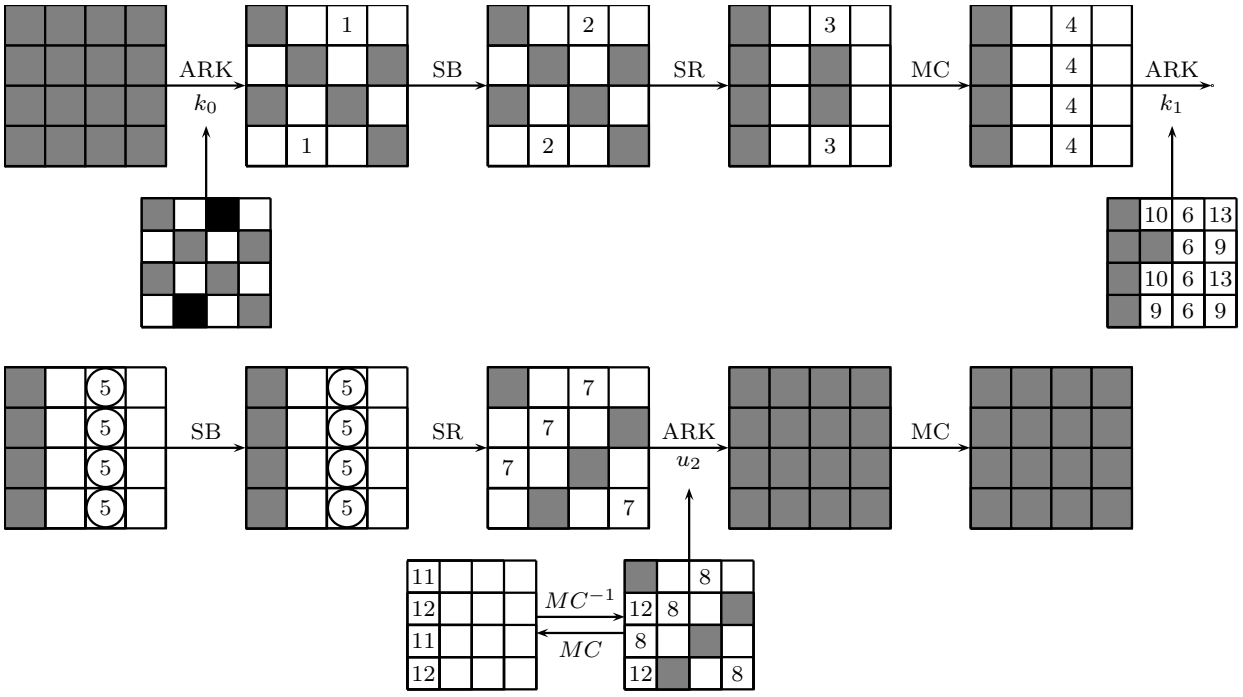
The time complexity of the attack is determined by the fact that 6 subkey bytes are guessed, and for each guess a few simple analysis steps are performed. Hence, the time complexity of the attack is 2^{48}.

The Automatically-Found Attack. The improved tool of §8.3 has found an attack with time and memory complexity 2^{32} in this setting, vastly outperforming human cryptanalysts. The attack is a meet-in-the-middle whose main ingredient is the possibility to isolate a set of about 2^{32} candidates for both $k_1[0..3]$ and $k_1[12..15]$ with only 2^{32} operations. These 8 bytes are then sufficient to recover the full key instantly.

First, we assume that $x_1[12..15]$ is known, and we try to derive the value of some other bytes. We can easily obtain the differences in $x_1[12..15]$. Then, by linearity of the `MixColumns` operation, we obtain the



(a) First half of the attack: The difference in the bytes marked is guessed. The bytes marked by 5 are found using the known input and output differences.



(b) Second Half: The difference in the bytes marked in black is guessed. The bytes marked by 5 are found using the known input and output differences. The bytes marked by 12 are found using the relation between the keys k_2 and u_2.

Figure 9.3: The attack with two known plaintexts on two round AES

differences in $w_0[12..15]$. Using Property 9.1, we also obtain the values and the differences in byte 1, 6, 11 and 12 of x_0 (and thus of k_0). Note that the values of $w_0[12..15]$ and $k_1[12..15]$ are revealed in the process. Let us denote by A the set of bytes that can be obtained from $x_1[12..15]$.

Similarly, if the value of $x_1[0..3]$ is known, then the values (and differences) in byte 0,2, 5,10, 13 and 15 of x_0 and k_0, as well as $w_0[0..3]$ and $k_1[0..3]$ could be recovered. Let us denote these bytes by B.

Even though the bytes in $A \cup B$ can take 2^{64} values, this can efficiently be narrowed down to 2^{32}. There exist (at least) 4 *linear* relationships between bytes of A and those of B:

$$\begin{aligned} f_1(A) &= g_1(B) \\ f_2(A) &= g_2(B) \\ f_3(A) &= g_3(B) \\ f_4(A) &= g_4(B) \end{aligned}$$

Thanks to these relations, a tuple of values from A is associated to a single tuple of values of B on average: for each one of the 2^{32} tuples of values in A, evaluate the f_i's and store the result in a hash table. Then for each one of the of the 2^{32} tuples of values in B, evaluate the g_i's, and loop-up the corresponding value(s) in A.

Two of these linear relations can be obtained very simply: given $k_1[0..3]$ and $k_1[12..15]$, we deduce $k_2[0..3]$. From there, it is also possible to compute bytes 0, 5, 10 and 15 from x_1 by partial decryption. Amongst these, $x_1[15]$ occurs in A while $x_1[0]$ occurs in B. This already gives two linear equations connecting A and B.

Two other constraints can be obtained in a more sophisticated way. First, we notice that given the key bytes in A and B, it is possible to retrieve the full k_2 except byte 4, 8 and 12 by just exploiting the key-schedule and Property 9.3. Focusing on the last two columns of w_1, we find that 3 bytes are known in each column in w_1 and two bytes are known in each column of z_1. Thanks to Property 9.2, this gives a linear relation for between the known bytes of each column.

9.3.2 A Three Known Plaintext Variant

We note that if the adversary is given *three* known plaintexts, then a simpler attack can be applied, with the same complexity, namely 2^{32} encryptions. The adversary applies the first phase of the manually-found attack twice (for the pairs (P_1, P_2) and (P_1, P_3)), and uses the values of $k_1[0..3]$ retrieved in that phase for a consistency check. Since for the correct guess of bytes $0, 5, 10, 15$ of k_0, both pairs suggest the same value of the four bytes of k_1, and for an incorrect guess, the two pairs suggest the same value only with probability 2^{-32}, this allows us to discard most of the wrong guesses. Then, the adversary performs the second phase of the attack only for the remaining guesses, and thus the time complexity of the attack is dominated by the first step, whose complexity is 2^{32} encryptions.

9.3.3 A Two Chosen Plaintext Variant

If the adversary is given two *chosen* plaintexts, then the time complexity can be reduced. We first describe an attack found manually by Dunkelman and Keller, with a complexity of 2^{28} encryptions. We will next describe an attack found by the improved tool, with complexity 2^8 (!).

The Manually-Found Attack. In order to improve on the known-plaintext scenario, the adversary asks for the encryption of two plaintexts which differ only in four bytes composing one column. Figure 9.4 shows the difference pattern. In this case, at the end of round 1, there are exactly 127 possible differences in each column. For each such difference, the adversary can apply Property 9.1 to the four S-boxes of the column, and obtain one suggestion on average for the actual values after the `SubBytes` operation of round 2. Combining the values obtained from all four columns, the adversary gets about 2^{28} suggestions for the entire state after the `SubBytes` operation of round 2, and each such suggestion yields a suggestion of the subkey k_2. Thus, the time complexity of the attack is 2^{28} encryptions.

The Automatically-Found Attack. The adversary also asks for the encryption of two plaintexts which differ only in four bytes composing one column. The attack relies on Property 9.6 below, which cleverly uses the linearity in the key-schedule of the AES.

Property 9.6. *For all $i \geq 1$ we have the following equations:*

i) $z_{i-1}[4..7] \oplus z_i[0..3] \oplus z_i[4..7] = MC^{-1}\Big(x_i[4..7] \oplus x_{i+1}[0..3] \oplus x_{i+1}[4..7]\Big)$

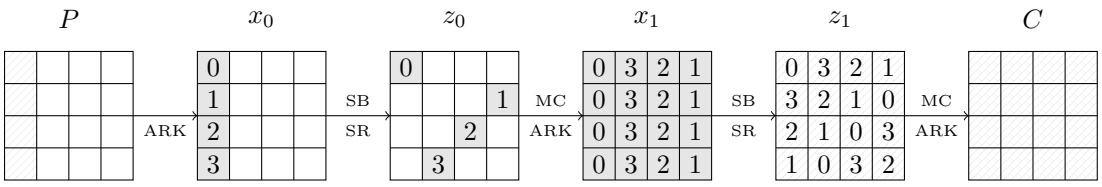


Figure 9.4: Two chosen plaintexts attack on two AES rounds. Gray bytes indicate the presence of a difference, and hatched bytes indicate the presence of a known difference. If byte i is known in x_0, then the actual values of all the bytes with the same number can be found.

ii) $z_{i-1}[8..11] \oplus z_i[4..7] \oplus z_i[8..11] = MC^{-1}\Big(x_i[8..11] \oplus x_{i+1}[4..7] \oplus x_{i+1}[8..11]\Big)$

iii) $z_{i-1}[12..15] \oplus z_i[8..11] \oplus z_i[12..15] = MC^{-1}\Big(x_i[12..15] \oplus x_{i+1}[8..11] \oplus x_{i+1}[12..15]\Big)$

Proof. Here again the theme is to exploit the interaction between the linearity of `MixColumns` and the linear operations in the key-schedule. We only prove the first equation (the proofs of the other two is quite similar). Expressing y in terms of w gives:

$$z_{i-1}[4..7] = MC^{-1}\left(w_{i-1}[4..7]\right)$$

We can relate w_{i-1} to x_i thanks to the `AddRoundKey` operation:

$$z_{i-1}[4..7] = MC^{-1}\left(k_i[4..7] \oplus x_i[4..7]\right)$$

And there, we can exploit the linearity of the key-schedule:

$$z_{i-1}[4..7] = MC^{-1}\left(k_{i+1}[0..3] \oplus k_{i+1}[4..7] \oplus x_i[4..7]\right)$$

The subkeys can then be expressed back in terms of w and x:

$$z_{i-1}[4..7] = MC^{-1}\left(w_{i+1}[0..3] \oplus x_{i+1}[0..3] \oplus w_{i+1}[4..7] \oplus x_{i+1}[4..7] \oplus x_i[4..7]\right)$$

And then, the linearity of `MixColumns` can be exploited as well:

$$z_{i-1}[4..7] = z_i[0..3] \oplus z_i[4..7] \oplus MC^{-1}\left(x_i[4..7] \oplus x_{i+1}[0..3] \oplus x_{i+1}[4..7]\right).$$

□

Assume that $x_0[0]$ is known: it is possible to deduce therefrom the value (and the difference) in $z_0[0]$, and finally the difference in $x_1[0..3]$ (by Property 9.2). Because the difference in $y_1[0..3]$ can be deduced from the ciphertexts, it follows that the actual values in $x_1[0..3]$ can be deduced thanks to Property 9.1. This also reveals bytes 0,7,10 and 13 of z_1 (observe Figure 9.4). It follows that if $x_0[0..3]$ were known, then the key could easily be deduced. The attack works by constructing a set of possibles values of $x_0[0..3]$ of expected size 256 in which the actual solution is guaranteed to be found. This process has a complexity of the order of 256 encryptions, and therefore dominates the complexity of the attack. A pseudo-code of the attack is shown in Algorithm 9.1. The attack works in 3 stages, each one using property 9.6 in a different way.

1. We first show that once $x_0[1]$ is known, then $x_0[0]$ can be determined using Property 9.6, item *iii)*. The equation is:

   $$z_0[12..15] \oplus z_1[8..11] \oplus z_1[12..15] = MC^{-1}\Big(x_1[12..15] \oplus C[8..11] \oplus C[12..15]\Big),$$

   We enumerate the possible values of $x_0[1]$ and compute all the bytes marked "1" in Figure 9.4. At this stage, the right-hand side the equation is fully known. In the left-hand side, $z_0[13]$ and $z_1[9]$ are known, and therefore $z_1[13]$ can be deduced by projecting the (vector) equation on the second component. The actual values and the differences can then be deduced in $x_1[1]$, which reveals the difference in $z_0[0]$ (by Property 9.2). The actual values in $x_0[0]$ can then be deduced by Property 9.1. We expect on average one possible value of $x_0[0]$ per value of $x_0[1]$.
2. We then seek to extend this procedure to $x_0[2]$ and $x_0[3]$. To this end, we still use Property 9.6, equation *ii)*:

   $$z_1[4..7] \oplus z_1[8..11] = z_0[8..11] \oplus MC^{-1}\Big(x_1[8..11] \oplus C[4..7] \oplus C[8..11]\Big), \qquad (\clubsuit)$$

Algorithm 9.1 Pseudo-code of the attack on 2 rounds using 2 chosen plaintexts.

```
1:  function 2R-2CP-Attack(P, C)
2:      for all x_0[2] ∈ F_{2^8} do                                  ▷ Build T_2
3:          compute z_0[10] and x_1[8..11]
4:          let u = x_1[8..11] ⊕ C[4..7] ⊕ C[8..11] in
5:          let i = z_0[10] ⊕ (0d, 09, 0e, 0b) · u in
6:          T_2[i] ← T_2[i] ∪ {x_0[2]}
7:      end for
8:      for all x_0[3] ∈ F_{2^8} do                                  ▷ Build T_3
9:          compute z_0[7] and x_1[4..7]
10:         let u = x_1[4..7] ⊕ C[0..3] ⊕ C[4..7] in
11:         let i = z_0[7] ⊕ (0b, 0d, 09, 0e) · u in
12:         T_3[i] ← T_3[i] ∪ {x_0[3]}
13:     end for
14:     for all x_0[1] ∈ F_{2^8} do                                  ▷ Retrieve the key
15:         Compute z_0[13], x_1[12..15], z_1[3], z_1[6], z_1[9], z_1[12]
16:         Compute z_1[13]                                          ▷ Using property 9.6
17:         Compute x_1[1], the difference in z_0[1], and x_0[0]
18:         Compute z_0[0], x_1[0..3], z_1[0], z_1[7] and z_1[10]
19:         Read possible value(s) of x_0[2] in T_2[z_1[6] ⊕ z_1[10]]
20:         Read possible value(s) of x_0[3] in T_3[z_1[3] ⊕ z_1[7]]
21:         Compute k_2 and check for correctness
22:     end for
23: end function
```

The third coordinate of the right-hand side can be entirely deduced from $x_0[2]$. We can therefore build a table yielding $x_0[2]$ from the third coordinate of the right-hand side of (♣), as shown in Algorithm 9.1, lines 2–7.

We perform the same operations with $x_0[3]$, using Property 9.6, equation i):

$$z_1[0..3] \oplus z_1[4..7] = z_0[4..7] \oplus MC^{-1}\Big(x_1[4..7] \oplus C[0..3] \oplus C[4..7]\Big), \qquad (\heartsuit)$$

Here, the fourth coordinate of the right-hand side can be entirely deduced from $x_0[3]$. We therefore build a table yielding $x_0[3]$ from the third coordinate of the right-hand side of (♡) (as shown in Algorithm 9.1, lines 8–13).

3. Once the two tables T_2 and T_3 have been built, we are ready to derive $x_0[2]$ and $x_0[3]$. For this purpose, we enumerate the values of $x_0[1]$, derive $x_0[0]$ as explained above. The third component of equation (♣) and the fourth component of (♡) can be computed, and thanks to T_2 and T_3 the corresponding values of $x_0[2]$ and $x_0[3]$ can be retrieved in constant time, resulting in an average of 256 suggestion for the first column of x_0. From there, k_2 can be deduced, and the key-schedule can be inverted to retrieve k_0.

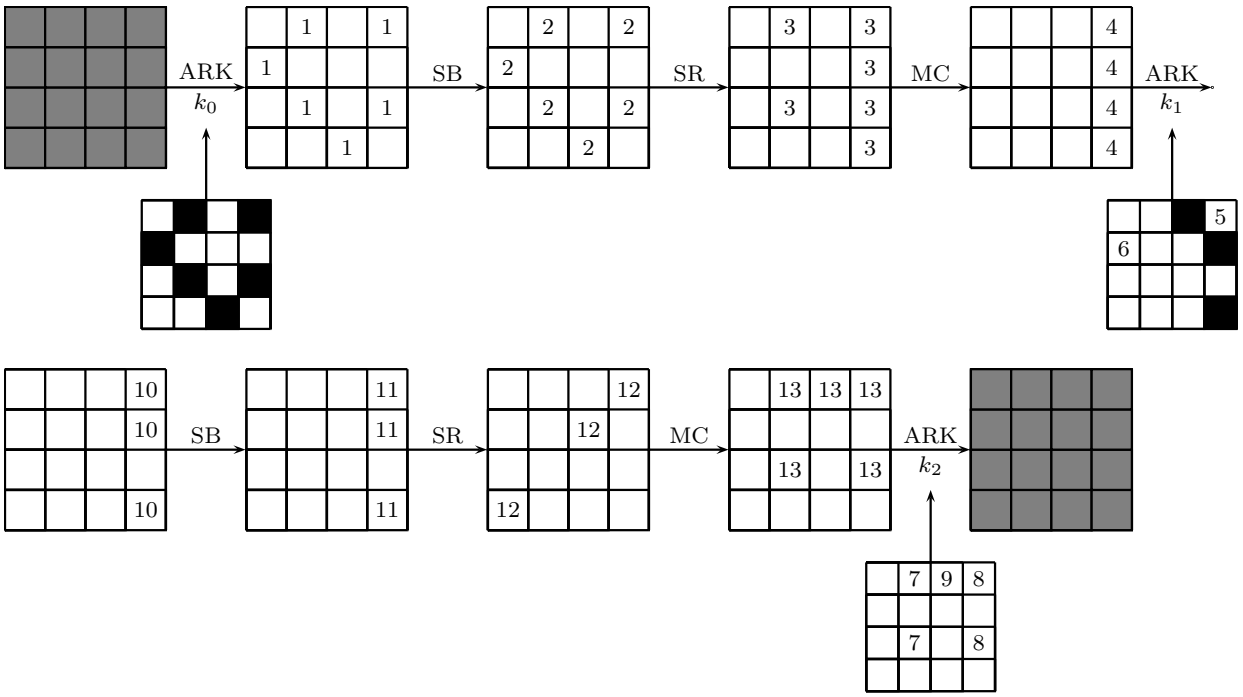
9.3.4 One Known Plaintext

Both Dunkelman and Keller, and the preliminary tool of §8.2 independently found a 1 known-plaintext attack against two full rounds. One possible version of the attack, depicted in Figure 9.5, is based mainly on Property 9.3 (the "jumps" in the key-schedule) and on many simpler key schedule considerations.

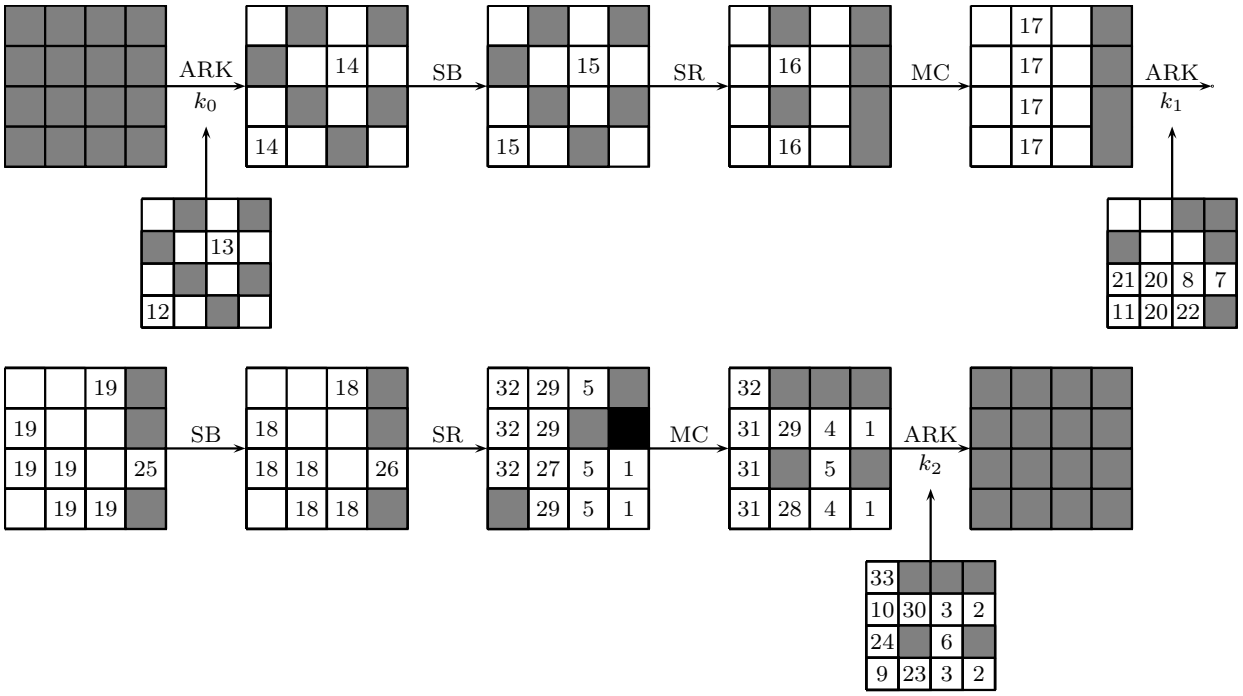
In the first phase of the attack, the adversary guesses nine subkey bytes (marked in black in the upper part of the figure). Step 7 uses Property 9.3(3), step 8 uses Property 9.3(2), steps 5, 6, and 9 use the key schedule, and the remaining steps are computed using the AES algorithm.

In the second phase of the attack, the adversary guesses one state byte (marked in black in the lower part of the figure). Steps 9 and 13 are based on Property 9.3(1), steps 1, 5, 29, and 32 use Property 9.2, steps 3,7,8,10–12, and 21–24 use the key schedule, and the remaining steps are performed by applying AES' operations on known values.

The time complexity of the attack is determined by the amount of bytes which are guessed. Namely, as the adversary guesses 10 bytes, the time complexity of the attack is 2^{80} encryptions. Because the automated tools are quite flexible, we could check without any effort that the SQUARE [DKR97] block cipher was a bit less strong than its successor (the AES): in the same setup, the preliminary tool finds an attack with 9 guessed bytes, *i.e.*, a time complexity of 2^{72} encryptions.



(a) First half of the attack: the value of the bytes marked in black is guessed. The bytes marked by 7 and 8 are found using Property 9.3.



(b) The bytes marked is black are guessed. The bytes marked by 9 and 13 are found using Property 9.3(1).

Figure 9.5: The attack with one known plaintext on two round AES.

9.3.5 A Time-Memory Tradeoff

The time complexity can be reduced at the expense of enlarging the memory complexity, using non-linear equations and a precomputed table as in Property 9.5. This improved attack was found by the improved tool of §8.3. In order to achieve this reduction, the adversary performs the following precomputation: Let bytes 4 and 14 of k_0 be denoted by b and c.[3] It is possible to represent all the bytes found during the attack procedures in terms of b, c, the plaintext, the ciphertext, and the other 8 key bytes which are guessed in the original attack procedure. At the end of the deduction procedure, after a suggestion for the full subkey k_2 (in terms of b and c) is obtained, the adversary decrypts the ciphertext through the last round and obtains a suggestion for bytes 4 and 5 of k_1. These bytes can be used as a consistency check, as they can be retrieved independently by the key schedule algorithm, using the suggestion of k_2. This consistency check supplies two non-linear equations in b and c, and it turns out that the equations are of the following form:

$$\begin{aligned} a_5 &= f_0(b, c, a_0, a_1, a_2, a_3, a_4), \\ a_7 &= f_1(b, c, a_0, a_1, a_2, a_4, a_6), \end{aligned} \tag{9.10}$$

where f_0 and f_1 are fixed known functions, and $a_0, a_1, \ldots, a_7$ are one-byte parameters depending on the plaintext, the ciphertext, and the eight additional subkey bytes guessed in the original attack. Since the values of $a_0, a_1, \ldots, a_7$ are very cumbersome, we do not present them in this paper, and refer the reader to [Der10].

Hence, it is possible to compute in advance the values of (b, c) corresponding to each value of $(a_0, a_1, \ldots, a_7)$, and store them in a table. In the online phase of the attack, the adversary guesses only 8 subkey bytes (instead of 10), computes the values of $(a_0, a_1, \ldots, a_7)$, and uses the table in order to retrieve b and c. The rest of the attack is similar to the original attack.

The time complexity of the resulting attack is reduced to 2^{64}, but on the other hand, the attack requires 2^{65} bytes of memory.

The memory requirement can be further reduced to 2^{49} bytes by observing that the knowledge of a_1, a_4, and the six subkey bytes $k_0[6], k_0[11], k_0[12], k_1[8], k_1[13], k_1[15]$ allows one to deduce the value of the two remaining subkey bytes guessed in the modified attack. Using this observation, the attack procedure can be slightly changed as follows: The adversary starts with guessing the values of a_1 and a_4, and prepares the table for the given value of a_1, a_4. In the online phase of the attack, the adversary guesses the six subkey bytes $k_0[6], k_0[11], k_0[12], k_1[8], k_1[13], k_1[15]$, deduces the value of the two additional required subkey bytes, and performs the original attack. This change reduces the memory complexity to 2^{49} bytes (since the table is constructed according to 6 byte parameters instead of 8), while the time complexity remains unchanged at 2^{64}.

9.3.6 Improved Attack When the Second MixColumns is Omitted

In §9.6 we present a differential attack on 6-round AES which uses as a subroutine a 2-round attack on AES. In the attack scenario, the two rounds attacked in the subroutine are the last two rounds of AES, *i.e.*, a full round and a round without the `MixColumns` operation. In this section we present an improved variant of the attack with two known plaintexts presented above that applies in this scenario. We note that this attack gives another evidence to the claim made in [DK10b] that the omission of the the last `MixColumns` operation in AES reduces the security of the cipher.

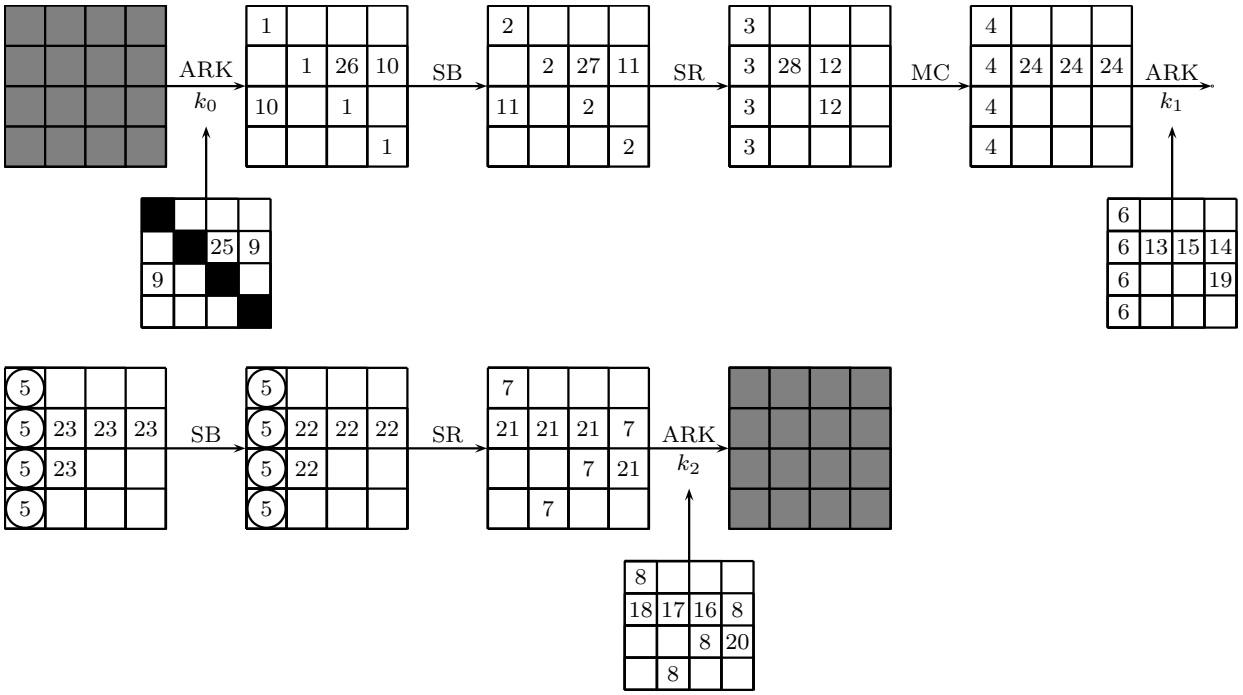
The attack, presented in Figure 9.6, consists of two phases. In the first phase, the first 13 steps are identical to the first 13 steps of the attack on two full rounds presented in §9.3.1 above. Steps 14–20 and 25 exploit the key schedule, and the rest of the steps apply AES' operations to known values.

The second phase uses Property 9.3 and simpler key schedule observations. Step 1 uses Property 9.3(1,2), step 20 uses Property 9.3(1), step 9 uses Property 2, steps 2–4,11,12,16–19, and 25 use the AES key schedule, and the remaining steps are performed using partial encryption or decryption.

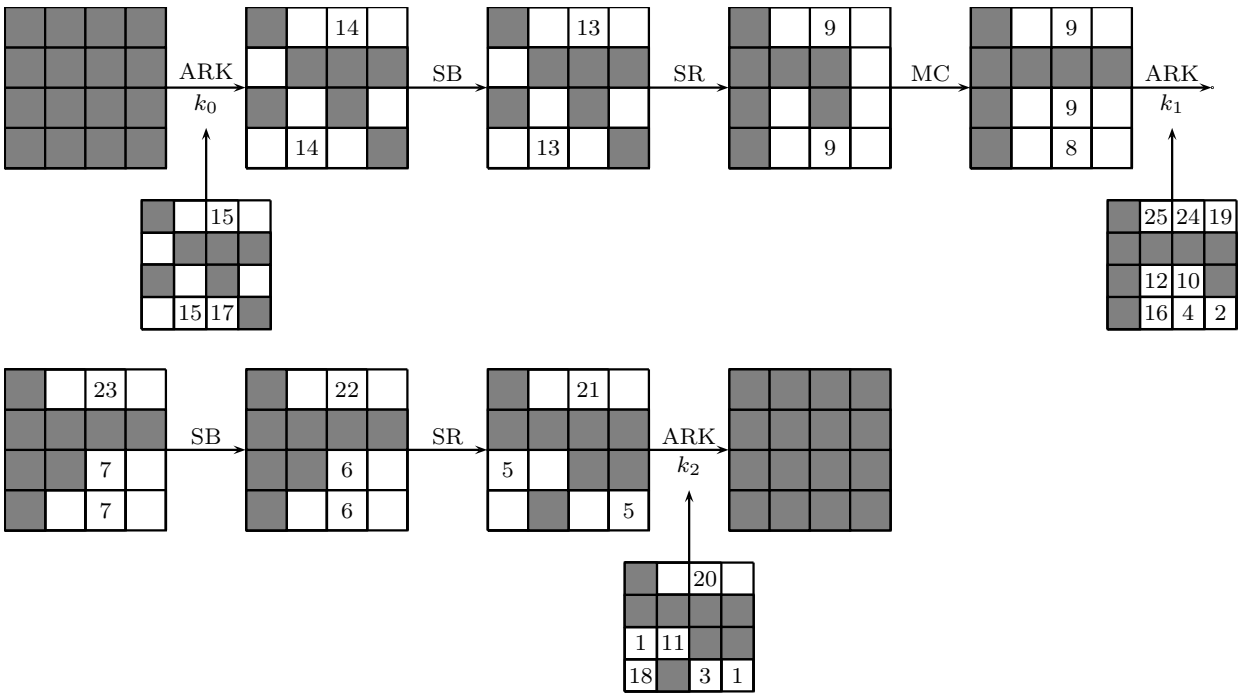
9.4 Attacks on Three-Round AES

In this section we consider attacks on three rounds of AES, denoted by rounds 1–3. First we present a simple attack with two *chosen* plaintexts, then we present a bit more complex meet-in-the-middle attack with 9 *known* plaintexts, and finally we present a very time-consuming attack with a *single known* plaintext.

3. We note that in this subsection we use notations which are somewhat different from the notations in the rest of the paper, in order to be consistent with the reference [Der10], in which this improvement is described in detail.



(a) First half of the attack.



(b) Second Half.

Figure 9.6: The attack on two rounds of AES without the second `MixColumns` using two known plaintexts. Bytes marked in black are guessed, and bytes marked in gray are known at this phase of the attack.

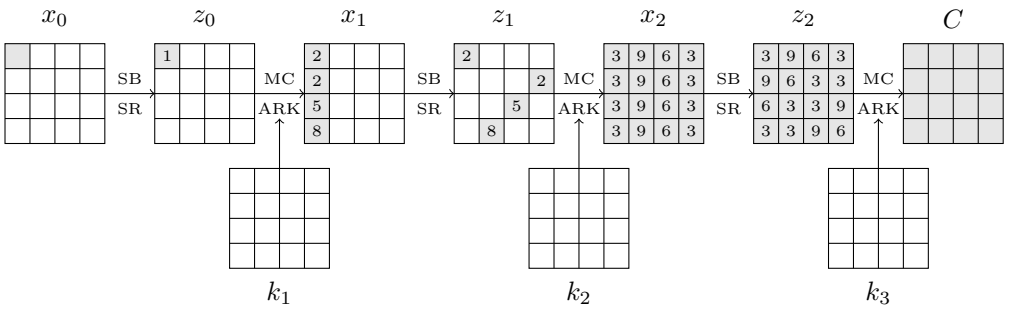


Figure 9.7: Fault attack against the AES. Gray square indicates the presence of a difference. The number indicates the step of the attack in which the value of each byte is discovered.

9.4.1 Two Chosen Plaintexts

The 2-round attack of §9.3.3 can easily be leveraged into a 3-round attack of complexity 2^{16}, thus improving on an attack found manually by Dunkelman and Keller with complexity 2^{32}, and described in [BDD+10].

In this improved attack, the adversary asks for the encryption of two plaintexts which differ only in the first byte. By guessing $k_0[0]$, the adversary obtains the differences in $x_1[0..3]$. This is sufficient to apply the attack of §9.3.3 to rounds 2 and 3. The complexity of the process is therefore 2^{16} encryptions.

9.4.2 Improvement to the Piret-Quisquater Fault Attack

In the Piret-Quisquater fault attack, against the full AES, an unknown difference is introduced in byte 0 of the internal state x_7. The adversary observes the output difference, and recovers the secret key in time 2^{32} [PQ03]. We show an improved procedure (found by the improved tool of §8.3) that recovers the key after the equivalent of 2^{24} encryptions.

The attack considers the last three rounds (rounds 8, 9 and 10), but to be consistent with the other three-round attack, we number the attacked rounds 1, 2 and 3. In this setting, *the plaintext is unknown*, and the only information is that there is a non-zero difference δ in $x_0[0]$. For the sake of simplicity, we describe the attack assuming that the final `MixColumns` operation has *not* been removed. The attack can be replayed without it, but some details become significantly messier.

One possible way to view this attack would be to guess the "fault" difference δ, to guess the actual value of $x_0[0]$, to derive the difference in $x_1[0..3]$, and to apply the two-round attack of §9.3.3 to rounds 2-3. However, it is possible to give a more direct yet pleasantly simple description of the key-recovery.

1. Guess the difference in $z_0[0]$
2. Guess the actual value of $x_1[0]$ and $x_1[1]$
3. Compute the difference in $x_2[0..3]$ and $x_2[12..15]$, then the actual values thanks to Property 9.1.
4. Use Property 9.6, with $i = 2$ and $j = 3$ (second component of the vector equation) to filter the guesses. Only 2^{16} out of 2^{24} should pass the test.
5. Guess the actual value of $x_1[2]$
6. Compute the difference in $x_2[8..11]$, then the actual values.
7. Use Property 9.6 with $i = 2$ and $j = 2$ (third component of the vector equation) to filter the guesses of step 5. Only 2^{16} should pass.
8. Guess the actual value of $x_1[3]$
9. Compute the difference in $x_2[4..7]$, then the actual values.
10. Use Property 9.6 with $i = 2$ and $j = 1$ (fourth component of the vector equation) to filter the guesses of step 8. Only 2^{16} should pass.
11. At this point we should have 2^{16} candidates for the actual values and the differences in $x_1[0..3]$. From those, x_2 can be reconstructed entirely, as well as k_3. It remains to simply test all the candidates.

9.4.3 Nine Known Plaintexts

An attack with 9 known plaintexts has been found manually by Dunkelman and Keller. It uses a combination between the differential approach and the standard meet-in-the-middle approach. The adversary

guesses subkey material in k_0 and k_3, and obtains a consistency check on the intermediate difference after the `ShiftRows` operation of round 2.

Concretely, denote the intermediate values in byte 0 after the `ShiftRows` operation of round 2 by $X_1, X_2, \ldots, X_9$. In the first phase of the attack, the adversary guesses bytes $0, 7, 10, 13$ of the equivalent subkey u_3 and partially decrypts the ciphertexts through the last round (obtaining the actual values in $x_3[0..3]$). Then, using the linearity of the `MixColumns` operation, the adversary computes the differences $X_1 \oplus X_2, X_1 \oplus X_3, \ldots, X_1 \oplus X_9$, and stores their concatenation (a 64-bit vector) in a hash table. In the second phase of the attack, the adversary guesses bytes $0, 5, 10, 15$ of k_0 and byte $k_1[0]$ and by partial encryption of the plaintexts, obtains the values of $X_1 \oplus X_2, X_1 \oplus X_3, \ldots, X_1 \oplus X_9$, and checks whether their concatenation appears in the hash table. This consistency check is a 64-bit filtering, and thus only $2^{72} \cdot 2^{-64} = 2^8$ key suggestions are expected to remain. By repeating the procedure with the three other columns, the adversary obtains about 2^{32} suggestions for the full subkey k_0 (along with many other subkey bytes), which can be checked by exhaustive key search. The time complexity of the attack is about 2^{40} encryptions, and the memory requirement is 2^{35} bytes of memory.

9.4.4 One Known Plaintext

The preliminary tool of §8.2 found a guess-and-determine attack with a single known plaintext, depicted in Figure 9.8. The attack consists of two phases.

In the first phase (shown in the top part of the figure) the adversary guesses 15 subkey bytes, and uses key schedule considerations deduce numerous additional subkey bytes in the four subkeys k_0, k_1, k_2, and k_3. Step 4 of the deduction uses Property 9.3(1), and the other steps use the key schedule algorithm directly.

The second phase (shown in the bottom part of the figure) is the meet-of-the-middle part of the attack. Using the known subkey bytes, the adversary partially encrypts the plaintext and decrypts the ciphertext and obtains sufficient information in order to apply Property 9.2 to the `MixColumns` operations of rounds 2 and 3. Steps 13 and 17 of this part use Property 9.2, and the other steps use AES' operations and the knowledge obtained in previous steps.

Since the adversary guesses 15 key bytes, the time complexity of the attack is 2^{120} encryptions. As in the single-plaintext attacks on one-round and two-round AES, the adversary can reduce the time complexity at the expense of enlarging the memory requirement, using non-linear equations and a precomputed table. The time complexity of the resulting attack is 2^{104} encryptions, and the memory requirement is 2^{49} bytes. Since the technique is similar to the improvement of the 2-round attack presented in §9.3.4, and the obtained equations are quite cumbersome, we do not present the improvement here and refer the reader to [Der10].

9.5 Attacks on Four-Round AES

In this section we consider attacks on 4-round AES. The improved tool of §8.3 found a single known-plaintext attack of complexity approximately 2^{120} operations. It is unclear that this "attack" can be implemented faster than exhaustive search, and our (limited) efforts to describe it in a human-readable way have failed so far.

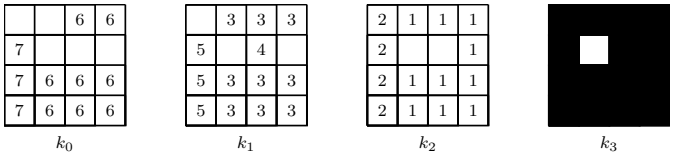
No other *known plaintext* attacks could be found manually, so we turned our attention to *chosen-plaintexts* attacks. The well-known "square" attack on 4 rounds requires 256 chosen plaintexts and the equivalent of 2^{14} encryptions. Dunkelman and Keller manually found attacks with 10,5 or 2 chosen plaintexts with respective time complexities 2^{40}, 2^{64} and 2^{104}. The improved tool of §8.3 automatically found a practical attack using four plaintext differing only in one byte, of complexity about 2^{32}.

We note that these attacks can be transformed into known plaintext attacks using the standard birthday-based transformations, but these usually result in a high data complexity. We do not present the attacks on 2 and 4 chosen plaintext as they are quite complex. The attacks using 5 and 10 chosen plaintexts are used in §9.8.3 below to give attacks on a modified version of the LEX stream cipher.

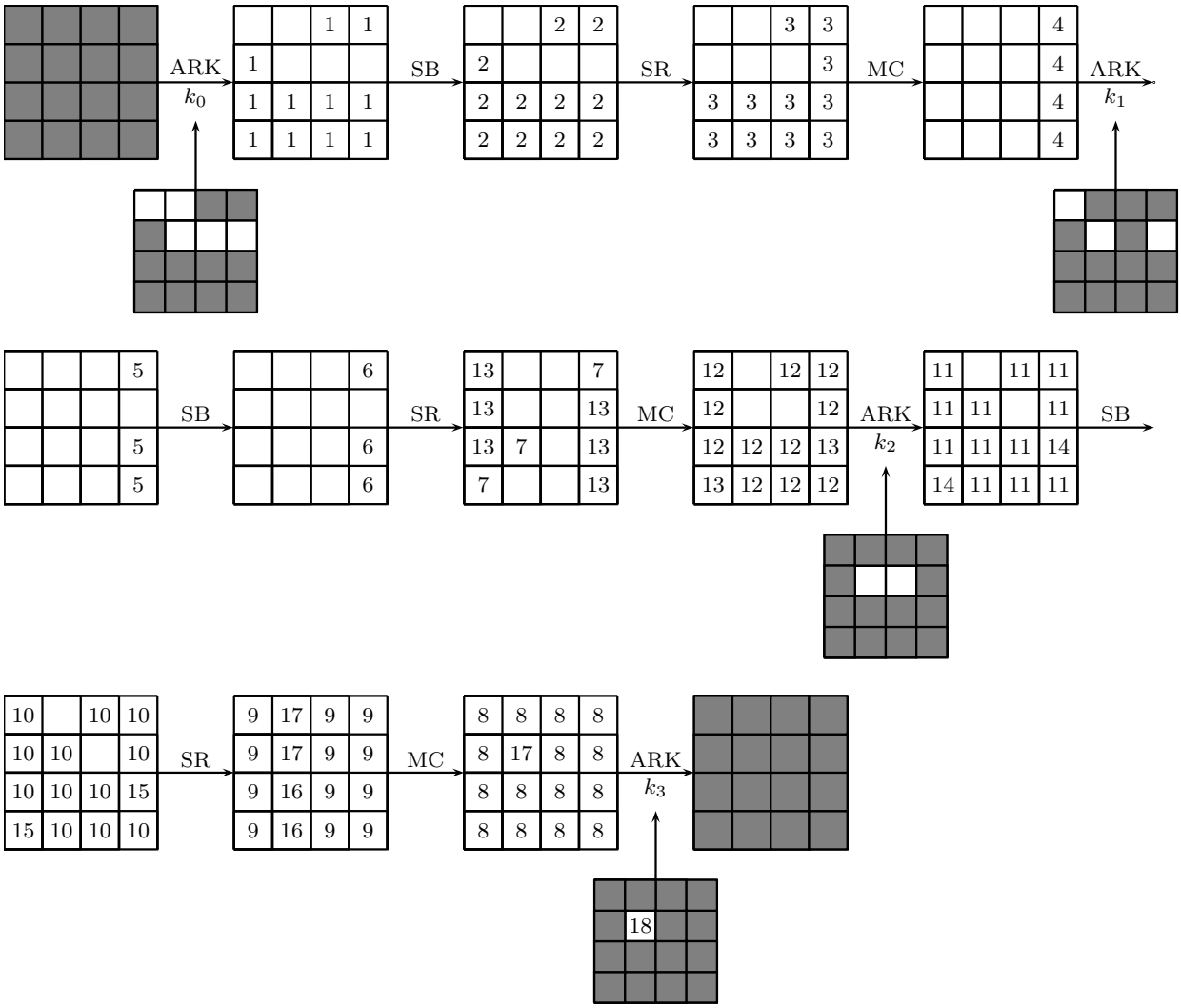
9.5.1 Ten Chosen Plaintexts

The attack with 10 chosen plaintexts is similar to the 3-round attack with 9 known plaintexts presented in §9.4.3. The adversary asks for the encryption of ten plaintexts which differ only in bytes 0,5,10,15. Then she guesses subkey material in the subkeys k_0, k_1 and the equivalent subkeys u_3 and u_4, and obtains a consistency check on some intermediate difference after the `MixColumns` operation of round 2.

Let us denote the intermediate values in byte 0 after the `MixColumns` operation of round 2 by $X_1, X_2, \ldots, X_{10}$. In the first phase of the attack, the adversary guesses bytes $0, 7, 10, 13$ of the equivalent subkey u_4 and byte 0 of the equivalent subkey u_3 and partially decrypts the ciphertexts through the last two rounds obtaining the actual values in the byte $x_3[0]$. (Note that reversing the order of the `MixColumns` and `AddRoundKey`



(a) The first half of the attack only uses the key-schedule. All bytes but one are guessed in k_3 (marked in black), and the diagram shows in which order other subkey bytes can be deduced using the key relations.



(b) Second half of the attack: deduction of the remaining subkey bytes. Known bytes are marked in gray.

Figure 9.8: The attack on three rounds of AES using one known plaintext

operations in the two last rounds allows her to obtain this intermediate value by guessing only 40 subkey bits). Then, using the linearity of the `AddRoundKey` operation, the adversary computes the differences $X_1 \oplus X_2, X_1 \oplus X_3, \ldots, X_1 \oplus X_{10}$, and stores their concatenation (a 72-bit vector) in a hash table.

In the second phase of the attack, the adversary guesses bytes $0, 5, 10, 15$ of k_0 and the byte $k_1[0]$. By the structure of the chosen plaintexts, this allows her to compute the differences between pairs of intermediate values $w_2[0..3]$ (since the actual values in byte 0 before the `MixColumns` operation of round 2 are known by partial encryption, and the difference in bytes $1, 2, 3$ is zero). Thus, the adversary obtains the values of $X_1 \oplus X_2, X_1 \oplus X_3, \ldots, X_1 \oplus X_{10}$, and checks whether their concatenation appears in the hash table. This consistency check is a 72-bit filtering, and thus only $2^{80} \cdot 2^{-72} = 2^8$ key suggestions are expected to remain. By repeating the procedure with the three other columns (from the ciphertext side), the adversary obtains about 2^{32} suggestions for the full equivalent subkey u_4 (along with many other subkey bytes), which can be checked by exhaustive key search. The time complexity of the attack is about 2^{40} encryptions, and the memory requirement is about 2^{43} bytes of memory.

9.5.2 Five Chosen Plaintexts

If only five chosen plaintexts are available to the adversary, she can perform a variant of the attack described above, at the expense of enlarging the time and memory complexities. The plaintexts are chosen as before, but more key material is guessed: from the ciphertext side, the adversary guesses bytes $0, 7, 10, 13$ of u_4 and bytes $0, 1, 2, 3$ of u_3, and from the plaintext side, the adversary guesses bytes $0, 5, 10, 15$ of k_0 and bytes $0, 1, 2, 3$ of k_1. This allows her to get a consistency check on the intermediate difference at the end of round 2 in bytes $0, 5, 10, 15$ (instead of only byte 0), and thus, the four pairs which can be extracted from the data supply a 128-bit filtering and only the correct key suggestion is expected to remain. Finally, the adversary repeats the attack procedure with three other columns from the ciphertext side, and obtains a single suggestion (or a few suggestions) for the full equivalent subkey u_4. The time complexity of the attack is about 2^{64} encryptions, and the memory requirement is about 2^{68} bytes.

9.5.3 Four Chosen Plaintexts

The improved tool of section 8.3 found a practical attack requiring only 4 chosen plaintexts and the time equivalent of about 2^{32} encryptions. The four plaintext only differ in byte 0 of the plaintext (but they *must* be pairwise different). We use the notation $x_i^{(j)}$ to denote the j-th message.

In a first phase, we construct 16 hash tables $\mathcal{T}_0, \ldots, \mathcal{T}_{15}$, which are subsequently used in the remaining steps of the attack. The table $\mathcal{T}_\ell$ is constructed according to the following steps:

1. First, enumerate all the possible values of $x_0^{(0)}[0]$. Because the differences in x_0 are known, then $x_0^{(i)}[0]$ can be deduced for $i = 1, 2, 3$. This in turn allows to determine the differences in $y_0[0]$, and also in $x_1[0..3]$.
2. Define $c_2 = \lfloor \ell/4 \rfloor$ and $r_1 = \sigma(c_2)$, where σ denotes the permutation (0321).
3. Next, enumerate $x_1^{(0)}[r_1]$. Because the differences in this byte are known, then the values in $x_1^{(i)}[r_1]$ can be deduced for $i = 1, 2, 3$. This allows to find the differences in $y_1[r_1]$, and then in $x_2[4c_2..4c_2 + 3]$.
4. Finally, enumerate the values of $x_2^{(0)}[\ell]$. Again, recover $x_2^{(i)}[\ell]$ for $i = 1, 2, 3$, and thus recover the differences in $y_2[\ell]$.
5. Store the association

$$\left(y_2^{(0)}[\ell] \oplus y_2^{(1)}[\ell], y_2^{(0)}[\ell] \oplus y_2^{(2)}[\ell], y_2^{(0)}[\ell] \oplus y_2^{(3)}[\ell]\right) \mapsto \left(x_0^{(0)}[0], x_1^{(0)}[r_1]\right)$$

   in the hash table $\mathcal{T}_\ell$.

The hash tables are now used in the following way: enumerate the values of $x_3^{(0)}[0..3]$, compute the differences in byte 0, 5, 10 and 15 of y_2, and use the differences to look-up in $\mathcal{T}_0, \mathcal{T}_5, \mathcal{T}_{10}$ and $\mathcal{T}_{15}$. Only keep values of $x_3[0..3]$ that suggest the same value of $x_0^{(0)}[0]$ (there should be about 2^8 of them). We implemented the attack, and we could indeed verify in practice that this procedure isolates a set of about $2^{8.5}$ candidates for the first column of x_3. It can then be repeated for the other three columns, and we are left with about $2^{34.5}$ candidates for the full x_3, each one of which suggest a full key (partial encryption reveals w_3, which in turns reveal k_4 and the key-schedule can be inverted back to k_0).

This could be refined a little bit by only considering the quadruplets of columns that suggest the same values of $x_1[0..3]^{(0)}$ (and there should very likely be very few of them). This would avoid testing 2^{32} keys.

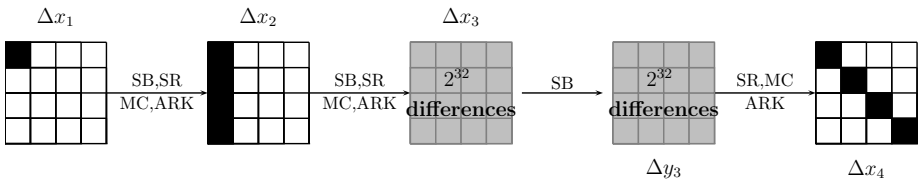


Figure 9.9: The 3-Round Truncated Differential Used in the 6-round Differential Attack.

9.6 Differential Attack on 6-Round AES

The design of AES follows the *wide trail* design strategy, which assures that the probability of differentials is extremely low, even for only four rounds of the cipher. For example, it was proved in [PSC+02, PSLL03], that any 4-round differential of AES has probability of at most 2^{-110}. Hence, it is widely believed that no regular differential attack can be mounted on more than 5 rounds of AES. Furthermore, the best currently known differential attack on AES-128 is on only four rounds, and all known attacks on 5 and more rounds use "more sophisticated" techniques like impossible differentials, boomerangs, or Squares.

In this section we show that the low data complexity attack on 2-round AES presented in §9.3.6 can be leveraged to a differential attack on 6-round AES. Although the data complexity of the resulting attack is high, the data complexity of its *known plaintext* variant is still smaller than the data complexity of the best known attack on 6-round AES in the known plaintext model. While our attack certainly does not threaten the security of AES, it shows that its security with respect to conventional differential attacks is lower than expected before.

As in most published attacks on reduced-round variants of AES, we assume that the `MixColumns` operation in the last round is omitted, like in the full AES. We were not able to extend the attack to the case where the last `MixColumns` operation is not omitted. This gives another evidence to the claim made in [DK10b] that the omission of the last round `MixColumns` affects the security of AES.

Our 6-round attack is based on the following 3-round truncated differential: The input difference in all bytes except for byte 0 is zero, and the output difference in all bytes except for bytes $0, 5, 10, 15$ is zero. We depict the differential in Figure 9.9. A pair satisfying the input and output requirements of the differential in rounds 2–4 is called a right pair.

Consider a right pair (P, P'). By the structure of AES, the intermediate difference at the input of round 3 is zero in all bytes except for $0, 1, 2, 3$. Thus, there are at most 2^{32} possible differences in the input of the `SubBytes` operation of round 4. On the other hand, since the difference at the output of round 4 is zero in all bytes except for $0, 5, 10, 15$, there are only 2^{32} possible differences after the `SubBytes` operation of round 4. Note that by Property 9.1, the input and output differences of a `SubBytes` operation yield a single suggestion (on average) for the actual values. Therefore, if (P, P') is a right pair, then there are only 2^{64} possibilities of the corresponding actual values after the `SubBytes` of round 4 (or equivalently, for the actual values after the `MixColumns` operation of round 4).

This observation allows us to mount the following known plaintext attack:

1. Ask for the encryption of $2^{108.5}$ plaintexts P_i under the unknown key, and denote the corresponding ciphertexts by C_i.
2. Insert (P_i, C_i) into a hash table indexed according to bytes 1–4,6–9,11–14 of P_i and bytes 1–6,8,9,11,12,14,15 of C_i, and consider only the colliding pairs in the hash table (which are the only pairs which may be right). The number of remaining pairs is $2^{216} \cdot 2^{-192} = 2^{24}$.
3. For each of the remaining pairs, assume that it is a right pair, and for each of the 2^{64} possible actual values after the `MixColumns` operation of round 4, apply the attack presented in §9.3.6 on rounds 5–6. Note that the 2-round attack requires that the difference in the four bytes $x_2[0..3]$ in the attacked variant is non-zero, and this condition is indeed satisfied in our attack (for the state x_6 which corresponds to x_2 in the two-round attack).

Since the time complexity of the 2-round attack presented in §9.3.6 is 2^{32} encryptions, the overall complexity of the attack is $2^{24} \cdot 2^{64} \cdot 2^{32} = 2^{120}$ encryptions. The data complexity of the attack is $2^{108.5}$ known plaintexts, which is smaller than the data complexities of the previously known attacks in the known plaintext model (see, *e.g.*, [CKK+01]).

9.7 A Forgery Attack Against Pelican-MAC

Pelican-MAC [DR05b] is a Message Authentication Code designed by Daemen and Rijmen in 2005. It is an instance of the more general ALRED construction by the same authors, which is reminiscent of CBC-

MAC but aims at greater speed [DR05a]. MACs derived from the ALRED construction enjoy some level of provable security: it is shown that breaking an ALRED MAC requires breaking the underlying block cipher or generating an inner collision, which presumably requires at least $2^{n/2}$ queries if the block cipher is n-bit wide. Pelican-MAC works as follows:

1. The internal state (an AES state) is initialized to $x_0 = \mathsf{AES}_K(0)$.
2. The message is split in 16-byte chunks, and each chunk is processed in two steps: it is XORed to the internal state, and 4 keyless AES rounds are applied (the `AddRoundKey` operation is skipped).
3. Finally, the full AES is applied with the key K to the internal state, which is then truncated and returned as the tag.

In this construction, recovering the internal state x_0 is sufficient to perform nearly-universal forgeries: first the adversary asks the MAC of an arbitrary message. Given her knowledge of x_0, she can compute the internal state x_{last} just before the full AES is applied and the tag T is returned. Then, given an arbitrary message M, she computes the internal state x_M after M has been fully processed. Then, she knows that $\mathsf{Pelican\text{-}MAC}_K(M \,\|\, x_M \oplus x_{last}) = T$, without querying the MAC (the extra message block sets the internal state to x_{last}, which is known to result in the tag T).

The best published attacks against Alpha-MAC (another ALRED construction) and Pelican-MAC has been recently found by Zheng Yuan, Wei Wang, Keting Jia, Guangwu Xu, Xiaoyun Wang [YWJ+09] and aim at recovering the initial secret internal state. For Alpha-MAC, after having found an internal state collision (this requires 2^{65} queries), the internal state is recovered with a guess-and-determine attack that makes about 2^{64} simple operations. For Pelican-MAC, an impossible differential attack recovers the internal state with data and time complexity $2^{85.5}$.

The general idea of our attack on Pelican-MAC is to find a single collision in the internal state, found by injecting message blocks following a fixed truncated differential characteristic. Then, the state recovery problem has been encoded in equations and given to the improved tool of §8.3. It must be noted that an attack with the same global complexity has been independently found time by Dunkelman, Keller and Shamir [DKS11], using impossible differential techniques. The "state-recovery" phase presented here is faster though. However, these authors also give attacks against 5-round and 6-round Pelican-MAC.

Our Attack. We now present our attack against Pelican-MAC, with time and data complexity 2^{64}. We pick an arbitrary message block M_1 and query the MAC with 2^{64} random two-block messages $M_1 \parallel M_2$, and store the (message,tag) pair in a table. Then, we query the MAC on $(M_1 \oplus \Delta) \,\|\, M_2'$, where Δ is zero everywhere except on the first byte, and M_2' is random. When the tags collide, we check whether there is also a collision in the internal state by checking if:

$$\mathsf{MAC}_K(M_1 \parallel M_2 \parallel M_3) = \mathsf{MAC}_K\Big((M_1 \oplus \Delta) \,\|\, M_2' \,\|\, M_3\Big)$$

for several random message blocks M_3. If all the resulting tags collide, then we known that an internal collision occurred after the first two blocks with overwhelming probability, and we have:

$$\mathsf{AES}_4(x_0 \oplus M_1) \oplus M_2 = \mathsf{AES}_4(x_0 \oplus M_1 \oplus \Delta) \oplus M_2'$$

In other terms, the input difference Δ goes to the output difference $M_2 \oplus M_2'$ though 4 keyless AES rounds. The most likely differential characteristic is the one shown in Figure 9.10, even though there could be accidental difference cancellations with small probability.

We then write down the state-recovery problem as a system of equations: two unknown states with a known one-byte difference yields two unknown states with a known (full) difference. The improved tool of §8.3 quickly found [4] an attack that runs in time and space about 2^{32}, and which is summarized by Figure 9.10. Property 9.2 tells us that if α, β, γ and δ denote the differences in z_1, then the differences in x_2 are:

$$\begin{pmatrix} 02\alpha & \beta & \gamma & 03\delta \\ \alpha & \beta & 03\gamma & 02\delta \\ \alpha & 03\beta & 02\gamma & \delta \\ 03\alpha & 02\beta & \gamma & \delta \end{pmatrix}$$

The state-recovery proceeds as follows:

1-a. Guess the values in $x_3[0..3]$ and obtain the differences (thanks to the output difference).

1-b. Partially decrypt to get suggestions for α, β, γ and δ (using Property 9.2).

4. it also found an attack with a smaller memory consumption 2^{24}, but the improved attack is much more complicated to describe

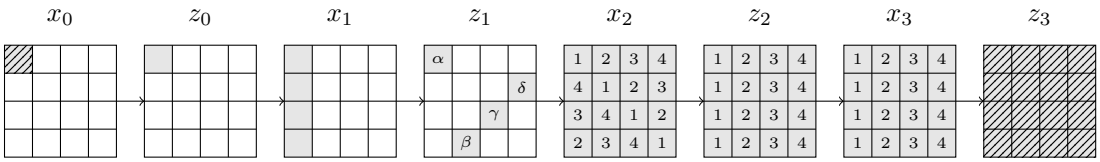


Figure 9.10: Truncated Differential characteristic used in the attack against Pelican-MAC. Gray squares denote the presence of a difference. Hatched squares denote a known difference.

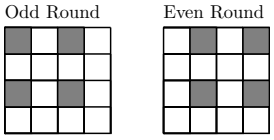


Figure 9.11: State Bytes which Compose the Output in Odd and Even Rounds of LEX. The gray bytes are the leaked bytes.

1-c. Store bytes 0–3 of x_3 in a hash table $\mathcal{T}_0$ indexed by $(\alpha, \beta, \gamma, \delta)$

2. Repeat the process with the second column of x_3. Store bytes 4–7 of x_3 in a table $\mathcal{T}_1$ indexed by $(\alpha, \beta, \gamma, \delta)$.
3. Repeat the process with the third and fourth column of x_3. Build tables $\mathcal{T}_2$ and $\mathcal{T}_3$
4. Enumerate $(\alpha, \beta, \gamma, \delta)$. Look-up $\mathcal{T}_0, \mathcal{T}_1, \mathcal{T}_2$ and $\mathcal{T}_3$ and retrieve the parts of x_3 corresponding to $(\alpha, \beta, \gamma, \delta)$, if present.
5. if $(\alpha, \beta, \gamma, \delta)$ occurs in the 4 tables, then we get a complete suggestion for x_3. Decrypt 3 rounds and recover x_0. Check if the input difference is right.

Alpha-MAC. Obviously, we cannot overally improve on the attack of [YWJ+09], since finding the internal state collision dominates the running time of their attack. However, it is noteworthy that the tool found a state-recovery procedure that requires only 2^{32} elementary operations and memory, when the first input message difference contains only one active byte. This is much more efficient than its counterpart in [YWJ+09].

9.8 A Key-Recovery Attack Against LEX

LEX is a stream cipher presented by Biryukov as an example of the *leak extraction* methodology of stream cipher design [Bir05, Bir06a]. In this methodology, a block cipher is used in the OFB mode of operation, where after each *round* of the cipher, some part of the intermediate encryption value is output as part of the key stream. LEX itself uses the AES as the block cipher.

In the initialization step of LEX, the publicly known IV is encrypted by AES under the secret key K to obtain $S = AES_K(IV)$. Actually, LEX uses a tweaked version of AES where the `AddRoundKey` before the first round is omitted, and the `MixColumns` operation of the last round is present. Then, S is repeatedly encrypted in the OFB mode of operation under K, where during the execution of each encryption, 32 bits of the internal state are leaked in each round. These state bits compose the key stream of LEX. The state bytes used in the key stream are shown in Figure 9.11. After 500 encryptions, another IV is chosen, and the process is repeated. After 2^{32} different IVs, the secret key is replaced. It follows that with a given key LEX can only generate $2^{46.3}$ bytes of keystream.

9.8.1 Prior Art

LEX was submitted to the eSTREAM competition (see [Bir05, Bir06b]). Due to its high speed (2.5 times faster than the AES in counter mode), fast key initialization phase (a single AES encryption), and expected security (based on the security of AES), LEX was considered a very promising candidate and selected to the third (and final) phase of evaluation. However, it was not selected to the final portfolio of eSTREAM due to an attack with data complexity of $2^{36.3}$ bytes of key stream and time complexity of 2^{112} encryptions found by Dunkelman and Keller a few weeks before the end of the eSTREAM competition [DK08]. These authors subsequently improved their own result, and the best published attack on LEX requires about 2^{40} bytes of keystream and the time equivalent of 2^{100} AES encryptions [DK10a].

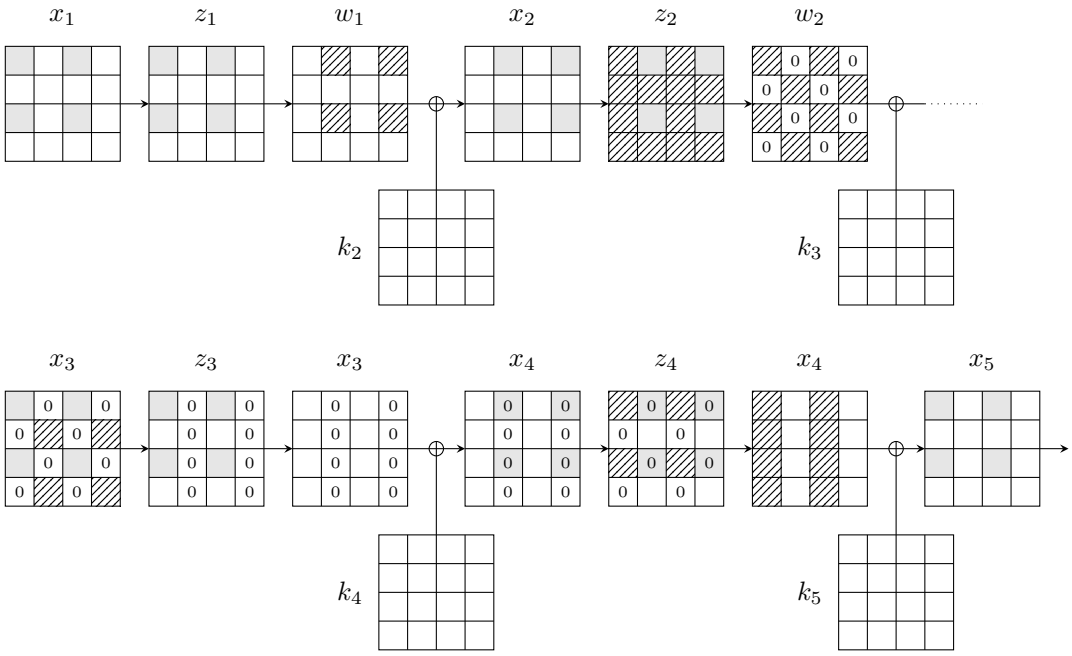


Figure 9.12: Gray squares are leaked to form the key-stream. The differences are null in squares with a 0. The differences in the hatched squares can be deduced from the leaked bytes and the existence of zero differences.

Their attack is illustrated by Figure 9.12. The key idea is to find a pair of internal states, potentially obtained with different IVs, and after different numbers of encryptions, that partially collide after 4 rounds. More precisely, the objective is to find a pair of state yielding the same bytes in $x_4[4..7]$ and $x_4[12..15]$. Because this is a collision on 64 bits, the birthday paradox guarantees that 2^{32} distinct internal states are necessary. In fact, the attack is not restricted to "start" at the first round of an AES encryption cycle, but can be applied (with minor variations) to rounds $1, \ldots, 8$. Thus, only $2^{64}/8 = 2^{61}$ pairs of encryptions are necessary for the collision to occur. This number of pairs can be obtained from 2^{31} distinct encryptions, and thus from $2^{32} \cdot 10 \cdot 4 = 2^{36.3}$ keystream bytes.

One of the problems is that the collision needed for the attack cannot be fully detected just by observing the keystream: it can be detected on bytes 4,6,12 and 14, but we have no way of detecting whether bytes 5,7,13 and 15 collide or not. The only solution is to assume that the full collision occurred and to run the next steps of the attack. In case of failure, we know *a posteriori* that the full collision did not occur. Thus, the remaining steps of the attacks have to be carried out on average 2^{32} times in order for a full collision to occur.

In the first attack of Dunkelman and Keller (given in [DK08]), the collision is exploited by a guess-and-determine attack that guesses 10 bytes. Their second attack (given in [DK10a]) uses an improved key-ranking procedure that filters the guesses and discards unlikely candidates.

Revisiting the Existing Attacks. The key-recovery problem can be encoded as a system of equations and given to the tools. The improved tool of §8.3 found that Dunkelman and Keller first attack was sub-optimal, as the guess-and-determine part of the attack could be dealt with in 2^{64} elementary operations (versus 2^{80} previously). This yields an attack with time complexity about 2^{96} and data complexity $2^{36.3}$, marginally improving on their second attack.

9.8.2 A New Attack

It turns out that the tool can be used to mount a different, more efficient attack. This new attack proceeds in 3 phases. The first phase is similar to the existing attacks. However, instead of looking for a *pair* of states colliding on bytes 4-7 and 12-15 in x_4, we look for *3-way collisions* on these bytes (*i.e.*, a triplet of states all having the same values in these bytes). The advantage of working with 3 messages instead of just two is that observation 9.1 generalizes nicely to this case: if 4 differences $\alpha, \beta, \gamma, \delta$ are randomly chosen in $\mathbb{F}_{2^8}$, then the probability that $S(x \oplus \alpha) \oplus S(x) = \gamma$ and $S(x \oplus \beta) \oplus S(x) = \delta$ is $2^{-9.5}$. Thus, in most cases, no single value of x satisfies these constraints.

Phase 1: Finding the 3-Collision. Finding the 3-collision requires $2^{128}/8 = 2^{125}$ triplets of encryptions, which can be obtained from $2^{42.5}$ distinct encryptions. This makes $2^{47.8}$ bytes of key-stream, about three times the maximally allowed quantity for a given key. This means that in the normal setting where LEX is restricted to produce $2^{46.3}$ bytes of key stream (80 terabytes), then out attack will only succeed with probability $\approx 1/32$. Indeed, under the normal restrictions, only 500×2^{32} encryptions are allowed, leading to $2^{120.3}$ triplets. Because each triplet leads to a 3-collision with probability 2^{-125}, it follows that the probability that the 3-collision exists is about $1/32$. Our attack thus targets on average one key over 32.

The problem of detecting the 3-collision is even more acute than previously, because it can only be partially observed. The strategy is again to repeat the last two phases of the attack on the expected 2^{64} triplets matching on the observable 32 bits. The subsequent steps require about 2^{24} simple operations, yielding a total time complexity of 2^{88}.

Phase 2: Exploiting the 3-Collision. First of all, by exploiting the zero-difference bytes and the known key-stream bytes, it is possible to reconstruct the differences between the 3 concurrent processes in vast portions of the internal state. Figure 9.12 shows the situation.

- The differences in bytes 0, 2, 8 and 10 of w_4 are given by the leakage in x_5. Also, the differences are known to be zero in bytes 1, 3, 9 and 11 of z_4. Thus, thanks to observation 9.2n the differences can be found in bytes 0-3 and 8-11 of both z_4 and w_4.
- It is also known that the differences are zero in bytes 4-7 and 12-15 of both z_3 and w_3, and these zero differences propagate to x_3 and w_2. Accordingly, using Property 9.2 in z_2 and w_2 yields the missing differences in x_3, w_2 and z_2.

The second phase of the attack obtains the value of bytes 0-3 and 8-11 in x_2, as well as bytes 5,7,13 and 15 in x_3 and bytes 0,2,8 and 10 in x_4. This requires 2^{18} simple operations, and is illustrated by Figure 9.13. In fact, four independent processes could be run in parallel:

1-a. Guess bytes 7 and 13 of x_3 (these are the dotted squares). This enables to find the actual values in the 3 concurrent states in bytes 8–11 of z_3 and w_3, because the differences in x_3 are known. This also yields the differences in bytes 8-11 of x_4.

1-b. In both x_4 and y_4, the differences are now known in bytes 8 and 10. Only a fraction $2^{-9.5}$ of the differences are consistent in each byte. Thus, we expect to sieve *all* the wrong guesses in the previous step, and to be left with *only the right value*. In addition, the actual values in bytes 8 and 10 of x_4 are revealed.

2-a. Guess bytes 5 and 15 of x_3 (cross-hatched squares). This yields the differences in bytes 0–3 of x_4.

2-b. Using the same sieving technique allows us to filter just the right value for the two guesses, and to get bytes 0 and 2 in $_4$.

3-a. Guess bytes 1 and 3 in x_2 (cross-hatched squares). This yields the corresponding differences in w_1. Then, the differences in bytes 0–3 of w_1 and x_2 can be found thanks to Property 9.2.

3-b. The differences are known in bytes 0 and 2 in both x_2 and y_2. Therefore, the sieving technique yields the only feasible value for bytes 0–3 of x_2.

4. Guess bytes 9 and 11 in x_2 (dotted squares). Use the same difference propagation and sieving to recover the only value of bytes 8–11 in x_2.

Phase 3: a Guess-and-determine Finish. The third phase of the attack is a standard guess-and-determine procedure that guesses 2 bytes in order to completely recover k_3, and thus the master key. It requires 2^{16} simple operations, and is summarized by Figure 9.14. The actual values are known (from the previous phase) in gray squares. Hatched squares denotes known differences. The bytes are numbered in the order in which they can be computed. Circled bytes numbered 11 are guessed. In fact, some key bytes can be determined from the result of the second phase without guessing anything.

Step 1,5,10,13 and 17 result from the knowledge of both w_i and x_{i+1}. Step 2,6,7,14 and 18 exploit the key-schedule equations, and bytes obtained in previous steps. Steps 3,8 and 15 are just partial encryptions/decryptions. Step 4,9,12 and 16 use Property 9.2.

9.8.3 An Attack on a Variant of LEX

In order to study the *leak extraction* design methodology, we consider a modified variant of LEX, in which instead of extracting 32 bits of the state after every AES round, the cipher outputs the entire state after every four rounds. In this case, the core cipher is reduced to 4-round AES (without the initial key whitening), but due to the stream cipher environment, the adversary is restricted to $2^{46.3}$ *known* plaintext bytes (as after that amount of key material, the stream cipher is rekeyed). It turns out that while 4-round AES is considered very weak due to the Square attack which can break it with only 2^8 chosen plaintexts,

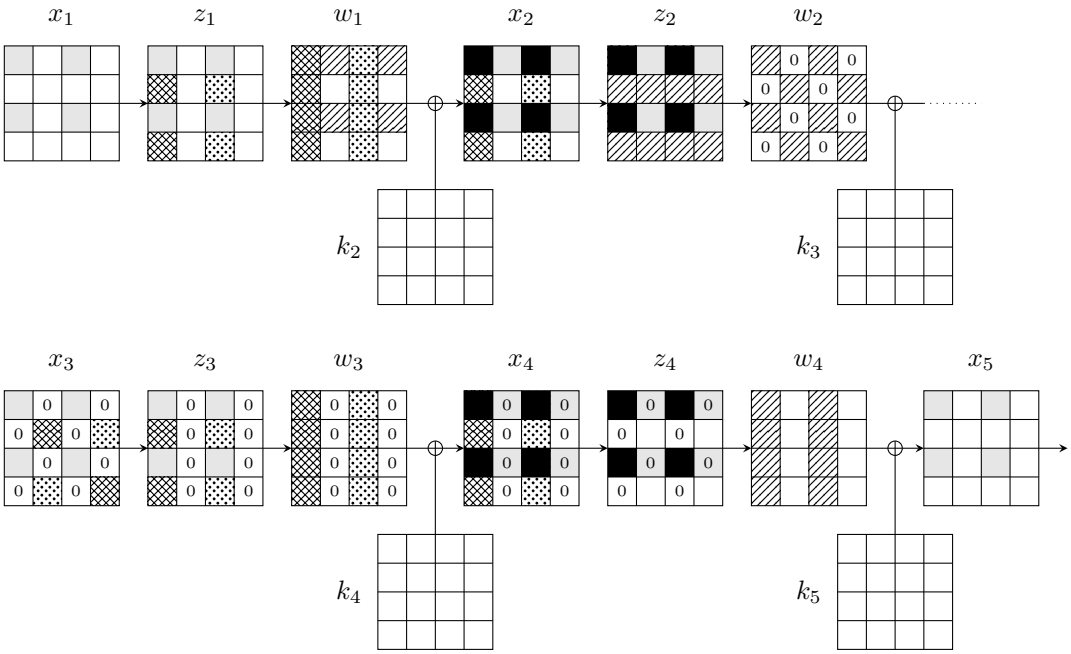


Figure 9.13: LEX: second stage of the attack.

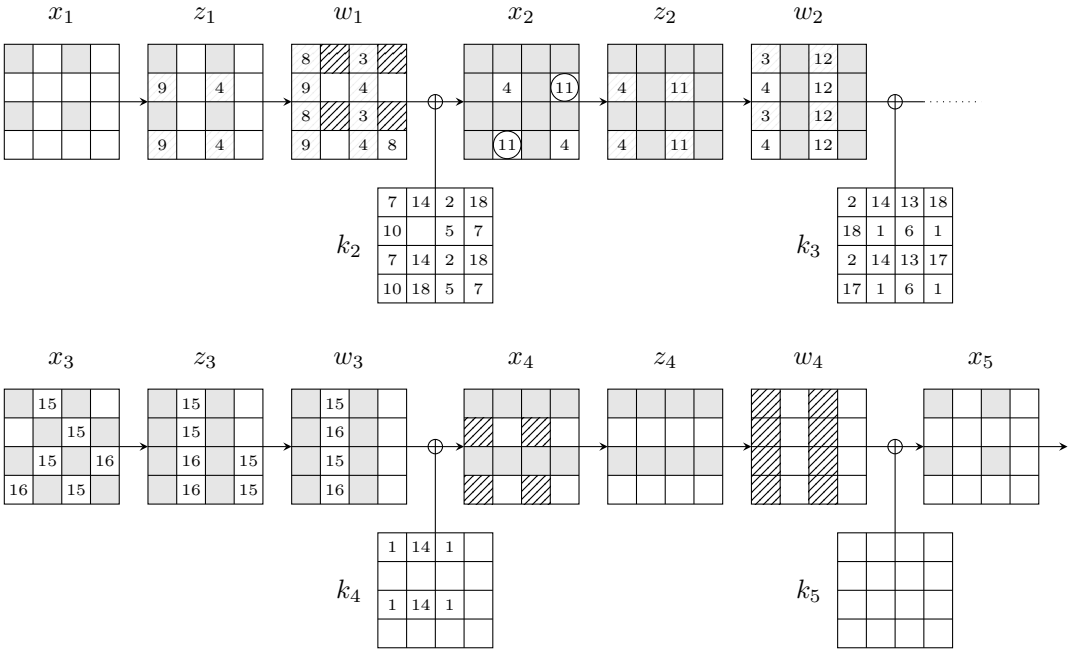


Figure 9.14: LEX: third phase of the attack.

all the previously known attacks on AES have very high data complexity when transformed to the known plaintext model, and thus cannot be applied in our scenario. Thus, a priori, it is not clear whether this variant of LEX is less secure than the original LEX.

However, using the attacks on reduced-round AES presented in the previous sections, we can show that this variant is practically insecure, and that even a stronger variant in which the full state is output every 5 rounds is still less secure than the original LEX.

Note that four-round AES without the initial key whitening step is equivalent to three full AES rounds (since the adversary can encrypt all the plaintexts through the first round without knowledge of the key). Thus, the attack on 3-round AES with nine known plaintexts, described in §9.4.3, applies directly to this variant and allows to break it with only 360 bytes of keystream (obtained from 9 AES invocations), time complexity of 2^{40} encryptions, and 2^{43} bytes of memory.

Similarly, if the full state is output every 5 rounds, then the underlying cipher is equivalent to four full AES rounds. In this case, our low complexity attacks perform only in the chosen plaintext model, and thus we use transformation to the known plaintext model, based on collecting a sufficient number of known plaintexts, until enough pairs with the required input difference are encountered.

We consider the attack with five chosen plaintexts presented in §9.5.2. A set of $2^{49.5}$ plaintexts picked at random is expected to contain $2^{98} \cdot 2^{-96} = 4$ pairs with zero difference in the 12 input bytes required in the chosen plaintext attack, and thus the attack can be applied with data complexity of $2^{54.5}$ keystream bytes, time complexity of 2^{64} encryptions and memory requirement of 2^{68} bytes.

It is possible to reduce the data complexity of this attack to $2^{33.5}$ known plaintexts at the expense of enlarging the time complexity, by slightly changing the underlying chosen plaintext attack. Instead of considering only pairs of plaintexts with zero difference in 12 bytes of the state, the adversary uses pairs with zero difference in only eight bytes: $2, 3, 4, 7, 8, 9, 13$, and 14. The adversary guesses bytes $0, 1, 5, 6, 10, 11, 12, 15$ of k_0 and bytes $0, 15$ of k_1, and obtains the intermediate difference in $w_2[0..3]$ (*i.e.*, at the output of round 2). On the other hand, the guess of bytes $0, 7, 10, 13$ of u_4 and byte 0 of u_3 is sufficient to obtain the intermediate difference in byte 0 at the end of round 2, which can be used as a consistency check. The rest of the attack is similar to the chosen plaintext attack on 4-round AES with ten chosen plaintexts presented in §9.5. Since the filtering is on 8 bits, 15 pairs with the required input difference are sufficient to discard most of the wrong key guesses. Thus, the data complexity is $2^{33.5}$ known plaintexts, or $2^{38.5}$ bytes of keystream. The memory requirement is 2^{44} bytes, and the time complexity is 2^{80} encryptions.

9.9 Implementations

We have implemented and verified attacks (or parts thereof) in practice. This brief section mentions some of the technique we used and the result we obtained. The source code of some of these attack is available at:

`http://www.di.ens.fr/~bouillaguet/implementation.html`

Several attacks are meets-in-the-middle that require hash tables containing 2^{32} entries, each entry being 2 or 4-byte long. The main difficulty in implementing these attacks was memory management (how to represent and store the tables). Careful and "low-level" memory management, *e.g.*, using `mmap`, was necessary for the attack to be somewhat practical. The standard techniques for hash tables (storing buckets as linked lists) incurs an important space overhead in our case, because the pointers are 64-bit wide, and are impractical.

We also observed that the distribution of the number of entries in each bucket roughly follows a Poisson law of expectation 1, so that the maximum number of entries in a bucket can be represented by an 8-bit number. We thus use three arrays to store the hash table:

- An array A_c stores the size of each bucket in 8-bit entries (size = 4Gbyte)
- An array A_h stores the content of all the buckets (size=16Gbyte)
- An array A_i stores the location of each bucket in the previous array (size=16Gbyte)

The last array is useful to access the hash table in $\mathcal{O}(1)$ time, but it needs not be stored, which means that such a hash table can be stored in a 20Gbyte file. We then used a two-pass approach: first count the number of entries with the same key in the table and update A_c. Then computes the entries in A_i. Lastly, perform a second pass and stores the actual data in A_h. This way, the peak memory consumption is 36Gbyte.

1 AES Round / 1 Known Plaintext. The code of a purely guess-n-determine attack that enumerates all the possible value of 5 bytes has been generated by the preliminary tool of §8.2. We easily parallelized the code by inserting some OpenMP statements at the right positions. This way, the attack runs in about 18 hours using 8 Intel Xeon E5440 cores at 2.83GHz.

The code of a meet-in-the-middle attack has been generated by the improved tool, and runs in a few minutes.

2 AES Rounds / 2 Known Plaintext. The meet-in-the-middle part attack has been implemented manually in C. Using the above techniques, it uses 52Gbyte of RAM, and isolates a set of about 2^{32} candidates for the first and last column of x_1 in about two hours. We checked that the set of candidates actually contains the correct solution, and that the number of candidates was consistent with our estimates.

2 AES Rounds / 2 Chosen Plaintext. The automated tools generated an implementation of this attack, which allowed us to test it. The automatically-generated C file is 110Kbyte long. On average, there are $2^{8.65}$ candidates for $x_0[0..3]$, which is very close to our hypothesis.

Piret-Quisquater Fault Attack. We implemented this attack manually in C and validated it in practice. It terminates in a couple of seconds on a laptop and finds the right solution. In particular, we could check that the actual number of tested candidates was consistent with the expected number.

4 AES Rounds / 4 Chosen Plaintext. We implemented the meet-in-the-middle part of the attack manually in C++. Our implementation uses the above techniques for representing the hash tables, and each one of the 16 tables requires 112Mbyte. The attack therefore runs on a laptop and uses less than 1.8Gbyte of RAM. The total running time of the meet-in-the-middle phase is about 2 hours on a single core (the code is easily parallelized is easy using OpenMP, and actually runs in 14 minutes using eight Xeon E5520 cores at 2.27Ghz).

Pelican-MAC. We implemented the state-recovery part of the attack (the collision-finding would not be feasible in practice for us) and validated it experimentally. The program, written in C++ is 650 lines long. Building the 4 tables took little less than 3 hours on one core of the above machine. Scanning the tables looking and testing the candidates took half an hour. The number of tested candidates is consistent with the expected number (2^{32}). We used C++ templates to write a single version of the function that generates the table, indexed by the number of the table to generate ; this way, the compiler does a good job of customizing the function for each table, while we only had to write it once.

LEX. We used our automatic code-generator to generate an implementation of phases 2 and 3 of the attack. On average, some bytes are assigned $2^{20.3}$ times, which is higher than our assumption. But their number is very small and finally, the overall complexity is close to 2^{16} encryptions.

Conclusion

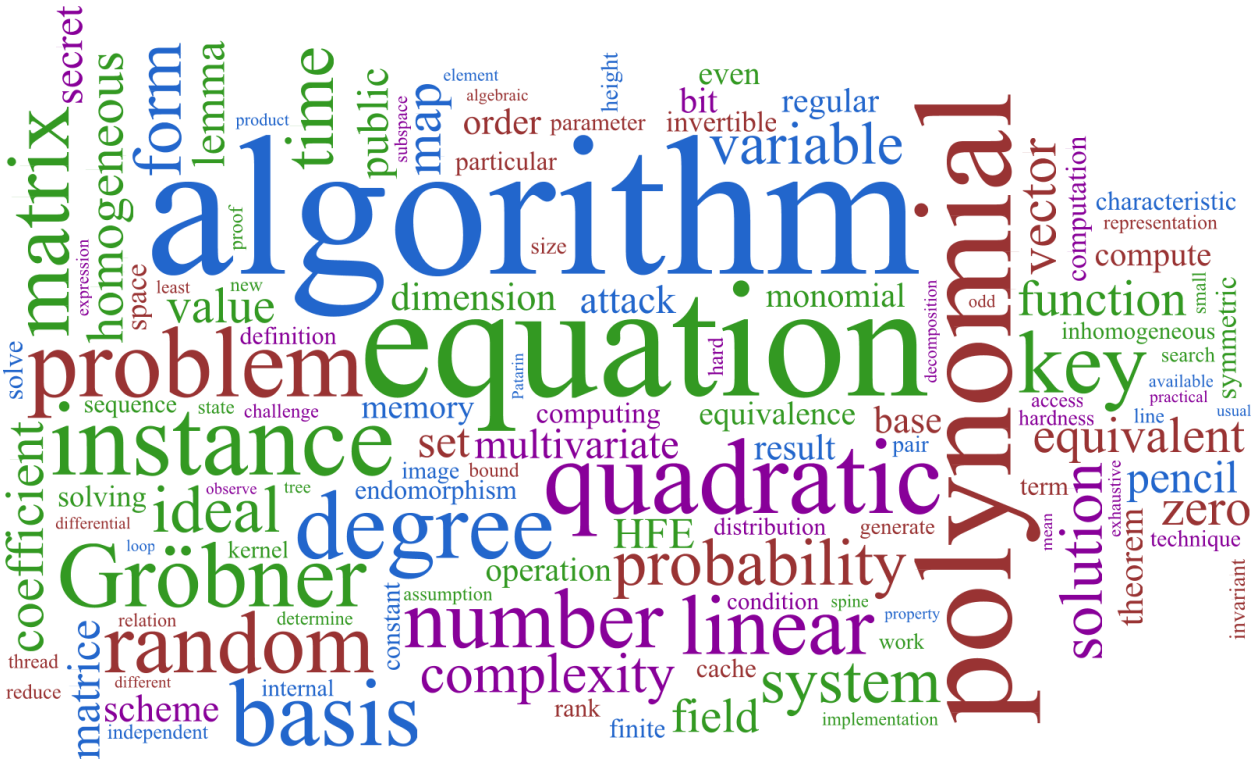
We have only had a limited experience with these tools so far, yet it is possible to draw a few preliminary observations.

Using the tools requires some knowledge of the primitive under scrutiny. For instance, the tools cannot are not designed to find good truncated differential paths. They can exploit such a path, for instance by finding a conforming pair efficiently, but the path has to be found by the user (or by a different tool). In this specific context, it is also up to the user to find a path that can be exploited by the tool. For instance, on two AES rounds, two truncated differential paths with probability one yield two very different results: if the 4 active byte are on the same column, the tool finds an attack of complexity about 2^8, whereas if the active bytes are on a diagonal, the best attack found by the tool has complexity 2^{32}.

The tools can be used to quickly verify high-level ideas or intuitions, while taking care of the low-level and nasty details". For instance, the idea "let us try to attack LEX with a 3-collision" could quickly be found to be effective, even though the concrete details of the attack took some time to be fully worked out.

In their present forms, *the tools are suited to situations where all the solutions of the given equations are wanted*. If there are much more variables than equations, the number of solutions will be overwhelming, and returning them all will be very expensive (and often unnecessary). A typical example is the case of collisions in hash functions (there are many, yet a single one is sufficient). A possible workaround would be to arbitrary fix some of the variables, but this requires human intervention, and it is not clear how to obtain good results this way. Another possibility would be to design a new set of tools tailored to find at least one solution to the given equations. This would likely require different strategies though (*i.e.*, no expensive precomputation). This seems to be an interesting topic for future work, since AES-based hash functions seem to be a natural target for automated techniques.

Analysis of Hardness Assumptions in Multivariate Cryptography



Introduction

Public-key cryptography crucially relies on *hard problems*. Recall that in this setting the public key contains enough information to encrypt or verify a signature in polynomial time, while the secret-key contains enough information to perform the converse operations (decrypting or generating signatures) efficiently. The two keys are usually, if not always, related by a hard problem: the public key completely defines an instance of a computational problem whose solutions form the set of corresponding secret keys. In the well-known RSA cryptosystem [RSA78] for instance, the public-key n is the product of two large secret prime numbers p and q. Retrieving the two primes from their product (*i.e.*, factorizing the public modulus) is probably the most classical hard problem in cryptography. Similarly, in the ElGamal encryption/signature scheme [Gam84], the public key is g^x, where g is a known element of some group, and x is a secret exponent. Retrieving x from g and g^x is known as the *discrete logarithm* problem, and is hard in some groups. The list of hard problems used in cryptography is ever growing and is already too long to be quoted extensively.

Security Proofs and Hardness Assumptions. The current trend in cryptographic design is to give mathematical proofs that any attacker able of breaking a cryptographic scheme is in fact capable of solving instances of a hard problem. This argument is usually made formal by exhibiting *reductions*, *i.e.*, programs supposed to solve the hard problem efficiently, and that for this purpose may use the attacker by sending it challenges to break, and exploit the corresponding answers. The security of the cryptographic scheme is therefore reduced to the *hardness* of the hard problem. The "security proofs" do not prove that the cryptographic scheme is unconditionally secure, but they prove that it is secure as long as solving instances of the hard problem requires too much computation to be possible in practice.

Of course, we must then trust the hard problem to be hard enough. The fact that they remain hard after decades of intense scrutiny is probably a good indicator that they will remain hard for the years to come. This seems somewhat unsatisfactory, but rigorous lower bounds on the complexity of solving computational problems are notoriously hard to obtain (just consider the **P** vs. **NP** issue). In fact, there is no *provably hard* problem known to date, in the sense that we do not know any function family that could be evaluated with a polynomial number of operations, but that provably could not be inverted with a polynomial number of operations.

The security of public key schemes therefore relies on the validity of *hardness assumptions*, whose formulation often looks similar to: "solving any instance of size n of problem P requires at least $f(n)$ operations", where f is a fast-growing, typically (sub)exponential function. Under such an assumption, choosing n big enough such that $f(n)$ operations is vastly beyond the current computational power of mankind guarantees that the cryptographic scheme at hand is secure. Of course, not all instances of the hard problem are necessarily hard, and weak instances must be avoided.

Hardness assumptions can be falsified by the discovery of new algorithms for solving the hard problem. This was for instance the case when the Number Field Sieve [LLMP93] was invented, and when the security of factorization-based schemes, such as RSA, had to be re-evaluated.

A universal Hardness Assumption. Consider any cryptographic primitive F taking as input a plaintext and a key, and producing something dependent on both (for instance F can be the RSA encryption, the AES, etc.). It is standard to assume that the description of F is known to the attacker, who is then able to synthesize a boolean circuit evaluating F. Since F is efficient, the circuit size is polynomial in the security parameter. In turn, from the boolean circuit, it is possible to derive a system of boolean equations in the input and output variables of F, as well as in some internal state variables. The size of this system is also polynomial in the security parameter. Given sufficiently input-output pairs so that the key is uniquely determined, retrieving the key amounts to solve the boolean equations in the key variables.

Cryptography would badly break down if solving large systems of boolean equations were feasible. It is therefore implicitly assumed that this is a very hard problem. Shannon pointed out in 1949 [Sha49] that lower-bounding the complexity of breaking a cryptographic scheme by that of solving a (complicated) system of boolean equations would be satisfying, because in any case there would not be any efficient cryptography if this were generically feasible.

From Boolean functions to Multivariate Polynomials. Solving systems of boolean equations is essentially equivalent to solving systems of polynomial multivariate equations over $\mathbb{F}_2$ (the field with two elements). Indeed, the multiplication in $\mathbb{F}_2$ is the AND operation between individual bits, and the addition is the XOR. Adding one then amounts to negate a boolean variable. Any system of boolean equations can be rewritten to contain only ANDs, XORs, and constants, with at most a polynomial blowup.

In turn, any system of multivariate polynomial equations can be transformed into a system of multivariate *quadratic* equations with the exact same solutions, by introducing new variables and new equations, with again an at most polynomial blowup. So, our slogan could be that "solving quadratic multivariate systems of equations over $\mathbb{F}_2$ is as hard as everything in cryptography" (but we know that this is a little exaggerated).

Note that this shows in passing that the problem of solving quadratic multivariate systems of equations over $\mathbb{F}_2$ is NP-complete, with an obvious reduction to SAT [GJ79]. This is also true over any finite field. The problem of solving Multivariate Quadratic equations (MQ for short) is thus fundamental, besides being mathematically natural and simple to understand.

In addition, no quantum algorithm is known to solve it faster than in the classical world, in contrast to some number-theoretic problems, such as factorization, to which Shor's polynomial-time algorithm [Sho97] would apply if quantum computer were available. It therefore makes sense to argue that cryptographic schemes relying on the hardness of MQ would survive in a world where efficient quantum computers exist.

Multivariate Cryptography. It was therefore natural to design cryptographic schemes whose security explicitly relies on the hardness of MQ. This resulted in "multivariate cryptography", a brand that encompasses all such cryptographic constructions. Multivariate polynomials have been used in cryptography as early as 1984 [OSS84a, OSS84b, FD85], mostly with the purpose of designing RSA variants with faster decryption. However, even before that, a group of Japanese researchers around Imai and Matsumoto designed the first public-key scheme explicitly based on the hardness of MQ. In fact, they built several, but only one (their "Scheme A") made it to the general crypto community, and was presented at Eurocrypt'88 [MI88] under the name C*.

Several years later, Patarin [Pat95] found a devastating attack against C*, allowing to decrypt and to forge signatures very efficiently, but not to recover the secret key from the public key. Soon after, he proposed a new multivariate trapdoor one-way function called Hidden Field Equations (HFE), on top of which reasonably efficient signature and encryption schemes could be built [Pat96b].

The decade starting with the introduction of HFE retrospectively looks like the *golden age* of multivariate cryptography. Many multivariate scheme have been proposed (we know at least 20 of them), including a plethora of bogus and vainly complicated proposal with a short lifespan. Three constructions stood out and received more attention than the others, because of their simplicity and their elegance:

- HFE [Pat96b] is a generalization of C*, retains its simplicity, but looks much more solid. Decryption is a bit more complicated and less efficient though. HFE has been the object of much scrutiny and cryptanalysis [KS99, FJ03, WP05b, GJS06, DGS06, DG10b], and we discuss its security at length in chapter 19. The most notable result on HFE is the practical break of an HFE challenge supposed to offer 80 bits of security by Faugère [FJ03]. Because of that, the cryptographic community vastly perceives HFE as broken, while the truth is apparently more complex.
- The Unbalanced Oil and Vinegar (UOV) signature scheme [KPG99] has been proposed in 1999 by Kipnis, Patarin and Goubin, as an improvement of the older "Oil and Vinegar" [Pat97] proposed in 1997 by Patarin. The original OV scheme turned out to be vulnerable to the "Rank-Attack" of Kipnis and Shamir [KS98], but UOV is immune to it, and remains unbroken to this day despite its simplicity.
- The SFLASH signature scheme [PCG01a] designed in 2001 by Courtois, Goubin and Patarin, is famous for having been considered by the NESSIE consortium to be suitable for signature in constrained environments. It is essentially the original C* with a crucial tweak that prevents Patarin's attack. SFLASH has been very badly broken in 2007 by Dubois *et al.* [DFS07, DFSS07], and all possible hopes of making it secure disappeared with a subsequently improved attack of Macario-Rat and Fouque [MR10].

The demise of SFLASH, coming after the practical break of the 80-bit HFE challenge, marks the end of the golden age of multivariate cryptography. It shattered the hopes, and trust, of the cryptographic community at large in multivariate cryptography, and at some point it was not unrealistic to say that multivariate cryptography was dead. In any case, it seems to be a less active field of research than what it was a decade ago.

Multivariate Cryptanalysis. The blossom of *multivariate cryptanalysis* is a consequence of that of multivariate cryptography. The cryptanalysis of multivariate schemes has a particular "flavor", compared to cryptanalysis of number-theoretic constructions, for instance.

Breaking the newly designed multivariate schemes required new and *ad hoc* cryptanalysis techniques and tools that it is impossible to exhaustively mention. Linear algebra and Gröbner basis [Buc65, CLO91] compu-

tations are the main tools of multivariate cryptanalysts. They even designed their own algorithms to tackle MQ and challenge its hardness assumption: first the *relinearization* technique [KS99] for overdetermined systems, then the *eXtended Linearization* (XL for short) [CKPS00], the *linear method* [KJ07], the *eXtended Sparse Linearization* (XSL) [CP02] and their numerous variants. XL was claimed to be subexponential, but the claim turned out to be bogus [Die04]. In the same vein, XSL was claimed to break the AES, but the claim was debunked in [CL05]. In addition, XL was eventually found [AFI+04] to be a rediscovery of an old Gröbner basis computation algorithm by Lazard [Laz83], which made it theoretically inferior to the most recent algorithms designed for this task, such as Faugère's $\mathbf{F_4}$ [Fau99] and $\mathbf{F_5}$ [Fau02]. The linear method is essentially an independent reformulation of "matrix-$\mathbf{F_5}$", a simplification of $\mathbf{F_5}$ described in [Bar04].

Some multivariate cryptanalysts also tried to use their new tools and techniques to attack symmetric primitives. The main ingredient of *algebraic attack* is to write down a system of multivariate quadratic equations describing the cryptographic primitive, and try to recover the secret key by solving it using generic solvers. Multivariate cryptanalysts first tried to attack real-life block ciphers, such as the AES [Cou04, CP02], without success, except maybe on very reduced versions of the DES [CB07]. Weak block ciphers have been broken this way though, such as Toyocrypt [Cou02] and Keelock [CBW08]. This kind of attack turned out to be a more serious thread against stream ciphers, as they could generically apply to whole families of designs [CM03, Cou03, COQ09]. In addition, algebraic techniques can be used as a sub-component of otherwise statistical attacks (for example to perform various kinds of "message modification"), as demonstrated for instance by [SKPI07, BF08, KMNP11].

Multivariate Trapdoor One-Way Functions. In multivariate public-key schemes, the public key is very often a family of quadratic polynomials $f_1, \ldots, f_m$ in n variables, with coefficients in a finite field $\mathbb{F}_q$. Encrypting a given vector $\mathbf{x} \in (\mathbb{F}_q)^n$ is done by simply evaluating the m polynomials over $\mathbf{x}$. An attacker, given the public key (the equations) and a ciphertext (the result of the evaluation of the polynomials), could try to recover the plaintext by solving the equations, *i.e.*, by solving an instance of MQ—which is supposedly intractable.

To make decryption possible, the polynomial equations must have a *special structure* enabling the legitimate user, aware of this special structure, to solve the equations in polynomial time. This is essentially achieved by taking an easily invertible system of polynomial equations, and performing a random linear change of coordinates, as well as a random linear combination of the equations. This could be seen as an obfuscation technique, because the public key is essentially an *obscured representation* of an easily-invertible system of quadratic equations. The implicit assumption is that knowing the two secret linear maps involved in the obfuscation process is the only way to access the easily-invertible core.

A seemingly insurmountable problem with this way of building multivariate trapdoor one-way function is that the public key is then *provably not* a uniformly distributed system of quadratic equations. As a consequence it is paradoxically impossible to prove that these schemes rely on the average-case hardness of MQ, because they only rely on the hardness of specific instances of the problem. This explains why a reasonably-sized HFE challenge could be broken: we do not know how to solve "generic" system of quadratic equations over $\mathbb{F}_2$ faster than exhaustive search, but HFE public keys turned out to be much easier to solve than "generic" systems.

Hardness assumptions in Multivariate Cryptography. An adversary could also attack multivariate schemes by trying to recover the secret linear change of coordinates from the public key. Multivariate schemes thus do not *only* rely on the hardness of MQ, but also on the hardness of another problem, that of retrieving the secret key from the public key. For C*, this problem has a relatively simple algebraic formulation in terms of equivalence of quadratic maps. Even after Patarin broke C*, the C* key-recovery problem looked quite difficult. This prompted Patarin to promote it as a hardness assumption of its own, somewhat unfortunately called the Isomorphism of Polynomials (IP) problem.

Besides MQ, IP is the most widely spread hardness assumption in multivariate cryptography, and a lot of work has been devoted to understand its exact hardness [PGC98b, GMS03, dVP03, Per05, FP06]. A third hard problem, MinRank, is sometimes used in multivariate cryptography [Cou01] and cryptanalysis [KS99]. While MQ and MinRank are NP-complete [BFS99], the status of IP however turned out to be much less clear.

Contributions. The third part of this dissertation is devoted to the study of hardness assumptions in multivariate cryptography. Chapter 10 describes our cryptanalytic toolbox. In chapter 11, we describe a new exhaustive-search algorithm for MQ with very good practical performances. Chapters 12 to 18 form a comprehensive study of the IP problem. The problem is defined in chapter 12, where its relevant subcases are determined and its cryptographic uses are also surveyed. Prior art in IP solving is reviewed in chapter 13. In chapter 14 we introduce the generic technique that we will use in the subsequent chapters to design new IP algorithms. We apply this generic technique to four subcases: to the Quadratic Forms Simultaneous

Equivalence problem (aka "Isomorphism of polynomial with One secret", or IP1S) in chapter 15, to the Cubic Form Equivalence problem (aka cubic IP1S) in chapter 16, to the Inhomogeneous Quadratic Maps Linear Equivalence problem (aka inhomogeneous IP) in chapter 17 and finally to the (homogeneous) Quadratic Maps Linear Equivalence problem (aka full IP) in chapter 18. We conclude by presenting an application of these new polynomial equivalence algorithms to the cryptanalysis of HFE in chapter 19.

Chapter 10

A Toolbox for Multivariate Cryptanalysis

This chapter collects the mathematic preliminaries and useful results that are used throught the last part of this dissertation. Most of these results are known or well-known, with the exception of some results on the properties of random matrices presented in §10.7 that are, to the best of our knowledge, original.

In this chapter, we expose the main ideas, techniques and notions used in multivariate cryptography and cryptanalysis. We recall very basic facts about finite fields and linear algebra. We next introduce our main object of study, namely multivariate polynomials, and we define useful derived notions. In particular we recall the usual theory of quadratic forms over odd-characteristic fields, and the less usual theorey of quadratic forms over a characteristic-two field. This leads us to consider the set of common roots of a family of multivariate polynomial, and the associated algebraic structures, the ideal they span. We then present Gröbner bases, and some of the mainstream algorithms used to compute them. We conclude by establishing some results on random matrices by counting the number of linear maps meeting various conditions over a given finite field.

10.1 Finite Fields and Vector Spaces

When $f_1, \ldots, f_k$ belong to a vector space, we denote by $\langle f_1, \ldots, f_k \rangle$ the subspace they span. We denote by $e_1, \ldots, e_n$ the canonical basis of a finite vector space of dimension n.

Let $\mathbb{K}$ be a finite field and $\mathbb{L}$ an extension of $\mathbb{K}$ of degree $n > 1$. Recall that $\mathbb{L}$ is essentially the quotient of $\mathbb{K}[X]$ by the principal ideal generated by $P(X)$, an irreducible polynomial of degree n over $\mathbb{K}[X]$. $\mathbb{L}$ is isomorphic to $\mathbb{K}^n$ via an application φ. For the sake of convenience, it can be specified that φ returns the only polynomial of each equivalence class of degree less than n. Hence, any application A defined over $\mathbb{L}$ can be seen as an application over $\mathbb{K}^n$ and conversely (just consider $\varphi^{-1} \circ A \circ \varphi$). Recall that any application over $\mathbb{L}$ is a polynomial of $\mathbb{L}[X]$. Lastly, given a fixed element $a \in \mathbb{L}$, the application $x \mapsto a \cdot x$ is linear, and thus can be represented over $\mathbb{K}^n$ by a matrix M_a. Lastly, we denote by $M_{i\bullet}$ the i-th row of the matrix M, and by $M_{\bullet j}$ the j-th column.

The Frobenius Map. The application $\mathrm{F} : X \mapsto X^q$ over $\mathbb{L}$ is called the *Frobenius map*. It is an automorphism of $\mathbb{L}$ that fixes any element of $\mathbb{K}$. As a consequence, F can also be seen as a matrix $\mathrm{F} \in \mathrm{GL}_n(\mathbb{K})$. A polynomial $P \in \mathbb{L}[X]$ commutes with F if and only if its coefficients are in $\mathbb{K}$.

Endomorphisms We denote by $\mathcal{M}_n(\mathbb{K})$ the set of all $n \times n$ matrices over the field $\mathbb{K}$. It is well-known that matrices represent linear maps, and that square $n \times n$ matrices in particular represent *endomorphisms* of $\mathbb{K}^n$. As usual, $\mathrm{GL}_n(\mathbb{K})$ denotes the set of all invertible $n \times n$ matrices over $\mathbb{K}$. The identity matrix of order n is denoted by I_n. Two matrices A, B are said to be *similar* (resp. *congruent*) if there exist $P \in \mathrm{GL}_n(\mathbb{K})$ such that $A = P^{-1} \times B \times P$ (resp. $A = {}^tP \times B \times P$). We recall the rank-nullity theorem: given an endomorphism of $\mathbb{K}^n$, then the dimension of its image plus the dimension of its kernel equals n.

Symmetric and Skew-Symmetric matrices. A matrix M is said to be *symmetric* if it is equal to its transpose (${}^tM = M$). A matrix M is *skew-symmetric* (also called *anti-symmetric*) if ${}^tM = -M$ and the diagonal coefficients of M are all zero. Symmetric and skew-symmetric matrices are usually very different objects. However, over fields of characteristic two, where $1 = -1$, the distinction is much less clear. The condition that the diagonal of a skew-symmetric matrix should be zero is not redundant in characteristic two, where skew-symmetric matrices are precisely the symmetric matrices with null diagonal. We will say that a matrix is *non-skew symmetric* if it is symmetric but not skew-symmetric (*i.e.*, if it has a non-zero element on the diagonal). The following theorems recall some elementary results on these objects.

Theorem 10.1 ([Alb38], theorem 1, 3, 4 and 6)**.** *Over any field,*

i) The rank of a skew-symmetric matrix is always even.

ii) Every matrix congruent to a skew-symmetric matrix is skew-symmetric.

iii) Every $n \times n$ skew-symmetric matrix of rank $2r$ is congruent to:

$$\begin{pmatrix} 0 & -I_r & 0 \\ I_r & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix}$$

iv) Two skew-symmetric matrices are congruent if and only if they have the same rank.

Theorem 10.2 ([Alb38], theorem 2 and 6)**.** *Over any field, we have:*

1. *Every matrix congruent to a non-skew symmetric matrix is non-skew symmetric.*
2. *Every non-skew symmetric matrix is congruent to a diagonal matrix.*

10.2 Multivariate Polynomials

The ring of all polynomials in n variables with coefficients over a finite field, conventionally denoted by $\mathbb{F}_q[x_1, \ldots, x_n]$ plays a central role in multivariate cryptography.

Definition 10.1. If $\mathbf{f} \in \mathbb{F}_q[x_1, \ldots, x_n]$, then the **homogeneous component** of degree d of $\mathbf{f}$, denoted by $\mathbf{f}^{(d)}$, is the sum of all monomials of degree d occuring in $\mathbf{f}$. A degree-d polynomial is said to be **homogeneous** if it only contains monomials of degree d.

A distinctive property of degree-d homogeneous polynomials is that if $\alpha \in \mathbb{K}$, then $f(\alpha \cdot \mathbf{x}) = \alpha^d \cdot f(\mathbf{x})$.

10.2.1 Quadratic Forms and Associated Bilinear Forms

Since multivariate *quadratic* polynomials play a special role in multivariate cryptography, we investigate some of their properties in more detail. A particularly important class of quadratic polynomials is formed by quadratic forms.

Definition 10.2. A **quadratic form** $f : \mathbb{K}^n \to \mathbb{K}$ is a homogeneous multivariate polynomial of degree two:

$$f = \sum_{i=1}^{n} \sum_{j=1}^{n} a_{ij} \cdot x_i x_j$$

It is customary to represent a quadratic form by the matrix $A = (a_{ij})_{1 \leq i,j \leq n}$. This representation is however redundant: it is not difficult to verify that if N is a skew-symmetric matrix, then A and $A + N$ both represent the same quadratic form form (in fact, all the possible representations of a given f can be written this way). It follows that a quadratic form can be *uniquely* represented by a (upper/lower) triangular matrix. In odd characteristic, a quadratic form can be uniquely represented by a symmetric matrix, and this more convenient representation is often prefered. It is also deeply connected, as we shall see below, to another standard object, the polar form.

Definition 10.3. If f is a quadratic form then $\phi(\mathbf{x}, \mathbf{y}) = f(\mathbf{x} + \mathbf{y}) - f(\mathbf{x}) - f(\mathbf{y})$ is a symmetric bilinear form called the **polar form** of f.

We are aware that the usual definition of the polar form is $\phi(\mathbf{x}, \mathbf{y}) = \frac{1}{2}(f(\mathbf{x} + \mathbf{y}) - f(\mathbf{x}) - f(\mathbf{y}))$. The advantage of the usual definition is that in odd characteristic $f(x) = \phi(\mathbf{x}, \mathbf{x})$. However, the inconvenient is that the polar form is undefined in characteristic two. Our definition is such that $f(x) = \frac{1}{2}\phi(\mathbf{x}, \mathbf{x})$ in odd characteristic. In even characteristic, we have $\phi(\mathbf{x}, \mathbf{x}) = 0$, and in fact ϕ is then a skew-symmetric bilinear form.

Bilinear forms are also conveniently represented by matrices: if $(e_i)_{1 \leq i \leq n}$ is a basis of $\mathbb{K}^n$, and if we consider the matrix A defined by $A = (\phi(e_i, e_j))_{1 \leq i,j \leq n}$, then $\phi(\mathbf{x}, \mathbf{y}) = {}^t\mathbf{x} \cdot A \cdot \mathbf{y}$. We denote by $\mathcal{P}(f)$ the matrix of the polar form of f in the canonical basis. The following lemma is straightforward, and links the polar form with the matrix representation of the corresponding quadratic form.

Lemma 10.3. *If a quadratic form $\mathbf{f}$ is represented by a matrix A, then $\mathcal{P}(f) = A + {}^tA$.*

Definition 10.4. Two quadratic forms $\mathbf{f}$ and $\mathbf{g}$ are said to be **equivalent** if there exist a matrix $S \in \mathrm{GL}_n(\mathbb{K})$ such that $f = g \circ S$.

The equivalence of quadratic forms has an immediate counterpart on their polar forms. It is easy to check that if S is a $n \times n$ matrix, and $g = f \circ S$, then $\mathcal{P}(g) = {}^tS \times \mathcal{P}(f) \times S$. Equivalent bilinear forms thus have congruent representations, and vice-versa.

We may now leverage our knowledge on symmetric matrices. Let $\mathbb{K}$ be a field of odd characteristic from now on. We therefore know that $2f(\mathbf{x}) = {}^t\mathbf{x} \cdot A \cdot \mathbf{x}$ for some symmetric matrix A. By theorem 10.2, $A = {}^tS \times D \times S$, where $S \in \mathrm{GL}_n(\mathbb{K})$ and $D = Diag(d_1, \ldots, d_n)$ is a diagonal matrix. Thus, f is equivalent, via S, to:

$$g(\mathbf{x}) = \sum_{i=1}^{n} d_i \cdot \mathbf{x}_i^2$$

We have just established the classical result: "any quadratic form is equivalent to a diagonal form". Note that if $d_i = 0$ in the expression of g, then f is equivalent to a quadratic form with (at least) one less variable.

Definition 10.5. The **rank of a quadratic form** f is the smallest integer r such that f is equivalent to a quadratic form in r variables. If the rank of f is strictly smaller than n, then f is said to be **degenerate** (otherwise it is **regular**).

It is straightforward that the rank of f is the number of non-zero d_i's in the expression of g. It follows that the rank of f is also the rank of its polar form, and of its symmetric matrix representation (but not of *any* matrix representation).

In characteristic two, this beautiful and practical theory unfortunately completely breaks down. The polar form is still congruent to a diagonal matrix, but this no longer implies that the corresponding quadratic form is equivalent to a diagonal form. In addition, diagonal forms are particularly uninteresting. First of all, since the Frobenius map is linear, the diagonal quadratic forms are in fact *linear*. In addition, they are all equivalent.

Lemma 10.4. *Any two non-zero diagonal quadratic forms are always equivalent over* $\mathbb{F}_{2^k}$.

Proof. Let $f(\mathbf{x}) = \sum_{i=1}^{n} a_i \cdot \mathbf{x}_i^2$ be a diagonal quadratic form. We assume w.l.o.g. that $a_{11} = 1$, and we define:

$$S = \begin{pmatrix} 1 & \sqrt{a_2} & \sqrt{a_3} & \cdots & \sqrt{a_n} \\ 0 & 1 & 0 & \cdots & 0 \\ 0 & 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & 1 \end{pmatrix}$$

Note that the square root is always well-defined and unique, since the Frobenius map is a bijection. Now, we set $g = f \circ S$. Then g is clearly equivalent to f, and we find:

$$\begin{aligned} g(\mathbf{y}) &= g\left(\mathbf{y}_1 + \sum_{i=2}^{n} \sqrt{a_i} \cdot \mathbf{y}_i, \mathbf{y}_2, \ldots, \mathbf{y}_n\right) \\ &= \left(\mathbf{y}_1 + \sum_{i=2}^{n} \sqrt{a_i} \cdot \mathbf{y}_i\right)^2 + \sum_{i=2}^{n} a_i \cdot \mathbf{y}_i^2 \\ &= \mathbf{y}_1^2 \end{aligned}$$

Since this applies to any f, the result is established. □

Lastly, theorem 10.1 implies that a non-diagonal quadratic form cannot be equivalent to a diagonal quadratic form. This follows from the fact that in characteristic two, the polar form is skew-symmetric, and thus cannot be congruent to a non-skew symmetric matrix. So, in characteristic two, diagonal forms are certainly not the nice canonical form they are in odd characteristic.

Another problem is that while the function that maps a quadratic form to its polar form is bijective in odd characteristic, this is no longer the case in characteristic two (the "diagonal terms" are lost in the process).

10.2.2 Quadratic Maps

In most cryptographic contexts, it is difficult to do much with a single polynomial. However, quadratic *functions* (*i.e.*, vectors of quadratic polynomials) are more interesting objects.

Definition 10.6. A **quadratic map** is a vector of n polynomials in n variables with coefficients over $\mathbb{K}$. The map itself is seen as a function from $\mathbb{K}^n$ to $\mathbb{K}^n$, where each coordinate is a quadratic expression of the input coordinates.

While a quadratic map is naturally a vector of quadratic polynomials, it can also be seen as a single quadratic polynomial whose coefficients are vectors in $\mathbb{K}^n$:

$$\begin{pmatrix} 5x^2 + 7xy - 3y^2 \\ x^2 - 3xy + 2y^2 \end{pmatrix} \simeq \begin{pmatrix} 5 \\ 1 \end{pmatrix} x^2 + \begin{pmatrix} 7 \\ -3 \end{pmatrix} xy + \begin{pmatrix} -3 \\ 2 \end{pmatrix} y^2$$

Most, if not all, usefull notions for quadratic polynomials translate nicely to quadratic maps, except maybe the pleasant representation by a single matrix with coefficients in $\mathbb{K}$.

10.2.3 Differentials and Derivatives

For instance, the polar form is easily adapted to quadratic maps. In the existing cryptographic litterature, this object has been called the *differential*, for reasons we explain below.

Definition 10.7. The **(discrete) differential** of a quadratic map $\mathbf{a}$ is the symmetric bilinear map:

$$\mathrm{D}\mathbf{a} : (\mathbf{x}, \mathbf{y}) \mapsto \mathbf{a}(\mathbf{x} + \mathbf{y}) - \mathbf{a}(\mathbf{x}) - \mathbf{a}(\mathbf{y}) + \mathbf{a}(0)$$

Given a vector $\mathbf{x} \in \mathbb{K}^n$, the **differential of a in x**, denoted by $\mathrm{D}_{\mathbf{x}}\mathbf{a}$, is the endomorphism of $\mathbb{K}^n$ that maps $\mathbf{y}$ to $\mathrm{D}\mathbf{a}(\mathbf{x}, \mathbf{y})$.

The name ("differential") comes from the analogy with the notion of the same name in calculus: if f a is (real) function from $\mathbb{R}^n$ to $\mathbb{R}^m$, then we say that f is *differentiable* at $p \in \mathbb{R}^n$ if there is a linear map $df_p : \mathbb{R}^n \to \mathbb{R}^m$, called the *differential*, such that for any $\varepsilon > 0$, there is a neighbourhood $N(p)$ of p such that for $x \in N(p)$:

$$|f(x) - f(p) - df_p(x - p)| < \varepsilon \cdot |x - p|$$

So, in calculus, the differential is a linear map associated to a point p, and expressing how a multivariate function varies around that point. Our discrete differential has similar properties. As the name suggests, this is reminiscent of the notion of derivation. However, the usual derivatives from "calculus I" make little sense over finite fields. We again have a related, *ad hoc* discrete notion.

Definition 10.8. The **(discrete) derivative** $\frac{\partial f}{\partial \mathbf{x}}$ of a quadratic map $\mathbf{a}$ with respect to a vector $\mathbf{x} \in \mathbb{K}^n$ is:

$$\frac{\partial \mathbf{a}}{\partial \mathbf{x}} : \mathbf{y} \mapsto \mathbf{a}(\mathbf{x} + \mathbf{y}) - \mathbf{a}(\mathbf{y}).$$

The (discrete) *derivative* $\frac{\partial \mathbf{a}}{\partial i}$ of $\mathbf{a}$ *with respect to its i-th variable* is $\frac{\partial \mathbf{a}}{\partial e_i}$.

Just like the usual derivative, if $\mathbf{a}$ is a polynomial map of total degree d, then $\frac{\partial \mathbf{a}}{\partial \mathbf{x}}$ is a polynomial map of degree $d - 1$. In particular, if $\mathbf{a}$ is a quadratic map, then $\frac{\partial \mathbf{a}}{\partial \mathbf{x}}$ is an affine function whose homogeneous linear component is $\mathrm{D}_{\mathbf{x}}\mathbf{a}$. It is also useful to note that:

$$\mathbf{f}(\mathbf{x} + e_i) = \mathbf{f}(\mathbf{x}) + \frac{\partial \mathbf{f}}{\partial i}(\mathbf{x}) \tag{10.1}$$

It is also important to notice that if $\mathbb{K}$ is a field of characteristic two, then for any $\mathbf{x} \in \mathbb{K}^n$, $\mathrm{D}_{\mathbf{x}}\mathbf{a}$ is singular, since it vanishes on $\mathbf{x}$.

10.2.4 Ideals and Varieties

Multivariate cryptography is built on the hardness of solving simultaneous polynomial equations over finite fields. While this can always be done in finite (exponential) time by exhaustive search, the inherently algebraic formulation of the problem leads to inherently algebraic algorithms. The theory of multivariate polynomial systems is fascinating, deep and elegant. In this section we briefly examine some selected aspects of the problem. The interested reader should consult [CLO91] for a more detailed exposition. While our objective is only to discuss the complexity of finding the solutions of a system, we will be dragged quite far.

Consider a system $\mathcal{S}$ of *linear* equations over $\mathbb{K}$, and the set $X \subset \mathbb{K}^n$ of all its solutions. Take any linear combinations of the equations: it also vanishes on X. A linear combination of the original equation is a new equation that "follows" from the original equations. The set of all the linear equations that can be derived from $\mathcal{S}$ is precisely the set of all possible linear combinations of the equations of $\mathcal{S}$, and it is a *vector space*. What is nice is that any other linear system generating this vector space has exactly the same solutions as $\mathcal{S}$.

With polynomial systems, the situation is more complex. Consider now a system $\mathcal{S}'$ of multivariate polynomials in $\mathbb{K}[x_1, \ldots, x_n]$, and let $X' \subset \mathbb{K}^n$ be the set of its solutions. Take any *polynomial* combination

of the original equations (*i.e.*, a sum of products of an initial equation and an arbitrary polynomial): we obtain a new polynomial that also vanish on X', *i.e.*, a new equation that "follows" from the original equations. The set of all polynomial equations that can be derived from $\mathcal{S}'$ is an **ideal** of $\mathbb{K}[x_1, \ldots, x_n]$. Recall that an ideal of $\mathbb{K}[x_1, \ldots, x_n]$ is a subset $\mathcal{I} \subset \mathbb{K}[x_1, \ldots, x_n]$, such that if $f, g \in \mathcal{I}$ then $f + g \in \mathcal{I}$, and if $f \in \mathcal{I}, g \in \mathbb{K}[x_1, \ldots, x_n]$, then $fg \in \mathcal{I}$. Any other system of polynomials generating the same ideal will have the same solutions, but the converse is not true in general (the polynomial systems (x, y) and (x^2, y^2) have the same solutions, namely $\{(0, 0)\}$, but they generate different ideals). More precisely, the sequence of polynomials $f_1, \ldots, f_m$ span the ideal [1]

$$\langle f_1, \ldots, f_m \rangle = \left\{ \sum_{i=1}^{m} P_i \cdot f_i, \text{ with } P_1, \ldots, P_m \in \mathbb{K}[x_1, \ldots, x_n] \right\}$$

The **affine variety** associated to an ideal $\mathcal{I}$ is the subset of $(\mathbb{F}_q)^n$ over which all the polynomials in the ideal vanish:

$$\mathbf{V}(\mathcal{I}) = \{\mathbf{x} \in \mathbb{K}^n \mid \forall P \in \mathcal{I}, P(\mathbf{x}) = 0\}$$

It is also possible to associate an ideal to a variety:

$$\mathbf{I}(V) = \{P \in \mathbb{K}[x_1, \ldots, x_n] \mid \forall \mathbf{x} \in V, P(\mathbf{x}) = 0\}$$

The ideal $\mathbf{I}(V)$ is therefore by definition the largest ideal whose associated variety is V. For instance, continuing on the previous example, $\mathbf{V}(\{(0, 0)\}) = \{x, y\}$.

Solving a system of polynomial equations essentially amounts to determine the variety of the ideal generated by the polynomials. Solving systems of linear equations is (usually) achieved by computing an equivalent system with a special shape, from which the solutions can be easily deduced, for instance a triangular system. The global strategy to solve polynomial equations is similar, and the key step is to compute a "nice" equivalent system.

10.2.5 Homogeneous Ideals

Definition 10.9. A **homogeneous ideal** of $\mathbb{K}[x_1, \ldots, x_n]$ is an ideal which is generated only by homogeneous polynomials (however it does not only *contains* homogeneous polynomials).

Homogeneous ideals are very convenient to express certain results, and overall their theory is easier than that of general ideals. Given an ideal $\mathcal{I}$, we may "homogenize" it using an extra dummy homogenization variable.

Definition 10.10. Let $f \in \mathbb{K}[x_1, \ldots, x_n]$ be a degree-d polynomial. The **homogenized** version of f, denoted by f^h, is the homogeneous polynomial of degree d of $\mathbb{K}[x_1, \ldots, x_n, h]$ defined by

$$f^h(x_1, \ldots, x_n) = h^d \cdot f\left(\frac{x_1}{d}, \ldots, \frac{x_n}{d}\right)$$

Homogenized polynomials can be de-homogenized: $f(x_1, \ldots, x_n) = f^h(x_1, \ldots, x_n, 1)$. We may therefore work with homogeneous ideals without lost of (much) generality. Given an ideal $\mathcal{I} = \langle f_1, \ldots, f_n \rangle$, we may consider its homogenized version $\mathcal{I}^h = \langle f_1^h, \ldots, f_n^h \rangle$. As expected, the image of $\mathcal{I}^h$ through the function that sets the last variable to 1 is indeed $\mathcal{I}$ [BKW93, lemma 10.53].

It must be noted that homogenizing a system of polynomial equation may introduce solutions "at infinity", *i.e.*, solutions with $h = 0$ that cannot be solutions of the original system and are thus parasitic.

10.2.6 Linearization

Before discussing techniques to solve multivariate systems of polynomial equations into further detail, we describe a folklore method for solving *very overdetermined* systems of multivariate quadratic polynomials. Given (at least) $(n+1)(n+2)/2$ linearly independent quadratic equations in n variables, a standard strategy is to consider each one of the $\binom{n+2}{2}$ monomial as an independent variable, and to solve the system of linear equations obtained this way. We therefore obtain the value of each monomial. When the system is inhomogeneous, this directly reveals the value of the variables. When the system is homogeneous, we only find the values of the quadratic monomials, but the actual values of the variables can be easily determined.

1. we extend our notation for the linear span to the "ideal" span.

10.3 Gröbner Bases

A **Gröbner basis** of an ideal is a generating set enjoying additional properties making it nice to work with. They are the nice form of polynomial systems, from which the solutions of the polynomial equations can be easily read off, to which we previously alluded to. They serve a handful of other purposes though, such as deciding the Ideal Membership problem (*i.e.*, testing if a given polynomial belongs to a given ideal). Our exposition is vastly inspired by [CLO91].

10.3.1 Monomial Orderings

In a univariate polynomial ring $\mathbb{K}[x]$, there is a natural (strict) total order relation between monomials: $1 < x < x^2 < x^3 < \dots$. This order is well-founded, and enjoy the additional property that if m_1, m_2 and m_3 are monomials such that $m_1 < m_2$, then $m_1 m_3 < m_2 m_3$. Moving on to multivariate polynomials, several such orders exist. Monomial orderings of the ring $\mathbb{K}[x_1, \dots, x_n]$ are in one-to-one correspondance with orderings of $\mathbb{N}^n$ (the bijection is examplified by $x^3yz^7 \leftrightarrow (3, 1, 7)$ in $\mathbb{K}[x, y, z]$).

Definition 10.11. Let $\alpha = (\alpha_1, \dots, \alpha_n)$ and $\beta = (\beta_1, \dots, \beta_n)$. The **lexicographic ordering** $>_{lex}$ is such that $\alpha >_{lex} \beta$ if, in the vector difference $\alpha - \beta \in \mathbb{Z}^n$, the leftmost non-zero entry is positive.

It follows that in $\mathbb{K}[x, y, z]$, we have $x >_{lex} y >_{lex} z$, but also $x >_{lex} y^5 z^{10}$, and in fact any variable dominates any monomial involving only smaller variables. The lexicographic order thus completely ignores the degree of the monomials. This feature can somewhat be bothersome, but there fortunately exist ordering that *refine the degree* (they are also called graded orderings). For instance, compare monomials according to their degree, and use the lexicographic ordering to break ties. In practice one of the most useful ordering is a twisted version of this one.

Definition 10.12. Let $\alpha = (\alpha_1, \dots, \alpha_n)$ and $\beta = (\beta_1, \dots, \beta_n)$. The **graded reverse lexicographic order** $>_{grevlex}$ is such that $\alpha >_{grevlex} \beta$ if

$$|\alpha| = \sum_{i=1}^{n} \alpha_i > |\beta| = \sum_{i=1}^{n} \beta_i$$

or $|\alpha| = |\beta|$ and the rightmost nonzero entry of $\alpha - \beta \in \mathbb{Z}^n$ is negative.

We can again check that $x >_{grevlex} y >_{grevlex} z$. The grevlex order looks at the rightmost variable and favors the smallest power: $x^5yz >_{grevlex} x^4yz^2$.

Given a multivariate polynomial f and a monomial ordering, we may sort the monomial occuring in the polynomial. The greatest one will be denoted by $LM(f)$, and is called the *leading monomial*. The coefficient of the leading monomial in f, denoted by $LC(f)$ is called the *leading coefficient*. The product $LC(f) \cdot LM(f)$ forms the *leading term* of f, denoted by $LT(f)$.

10.3.2 Multivariate Polynomial Division

In the univariate case, it is a well-know result that $\mathbb{K}[x]$ is a *principal* ring, meaning that all its ideals have a unique generator. Given an ideal $\mathcal{I} = \langle f \rangle$, a polynomial $g \in \mathbb{K}[x]$ belongs to $\mathcal{I}$ if and only if it is a multiple of f. This gives an easy algorithm to decide Ideal Membership: compute the euclidean division of g by f, and check whether the remainder is zero. Recall that the division algorithm works by repeatedly subtracting to g a suitable multiple of f, so that at each step the leading term of g is canceled out. The remainder cannot be affected in this way by f, because it is of strictly smaller degree.

It is quite tempting to extend this process to the multivariate setting. Algorithm 10.1 works the same way: given an (ordered) sequence of polynomials $f_1, \dots, f_t$ and a polynomial $g \in \mathbb{K}[x_1, \dots, x_n]$, repeatedly substract to g products $m \cdot f_i$, where m is a carefully chosen monomial such that $LT(m \cdot f_i) = LT(g)$. This algorithm shows that given our sequence of polynomials, every polynomial $g \in \mathbb{K}[x_1, \dots, x_n]$ can be written as:

$$g = a_1 f_1 + \dots + a_t f_t + r$$

where $a_i, r \in \mathbb{K}[x_1, \dots, x_n]$, and either $r = 0$, or r is a linear combination with coefficients in $\mathbb{K}$, of monomials, non of which is divisible by any of $LT(f_1), \dots, LT(f_s)$.

The problem is that, as opposed to what happens in the univariate case, the remainder r is *not uniquely defined* by this process. This may happen, for instance, if f_i and f_j have the same leading term, and could both be used at some point to cancel the leading term of g. This process cannot be used to test ideal membership because of the non-uniqueness of the remainder r.

Algorithm 10.1 Multivariate Polynomial Division.

```
1: function MultivariateDivision(f, f_1, ..., f_s)
2:     a_0 ← 0, ..., a_s ← 0
3:     p ← f
4:     while p ≠ 0 do
5:         i ← 1
6:         division-occured ← false
7:         while i ≤ s and division-occured= false do
8:             if LT(f_i) divides LT(p) then
9:                 a_i ← a_i + LT(p)/LT(f_i)
10:                p ← p − (LT(p)/LT(f_i)) · f_i
11:                division-occured ← true
12:            else
13:                i ← i + 1
14:            end if
15:            if division-occured = false then
16:                r ← r + LT(p)
17:                p ← p − LT(p)
18:            end if
19:        end while
20:    end while
21:    return (a_s, a_{s−1}, ..., a_1), r
22: end function
```

10.3.3 Definition of Gröbner Bases

Given an ideal $\mathcal{I} \subset \mathbb{K}[x_1, \ldots, x_n]$ other than $\{0\}$, we consider the set $LT(\mathcal{I})$ of leading terms of elements of $\mathcal{I}$. Thus,

$$LT(\mathcal{I}) = \{LT(f) \mid f \in \mathcal{I}\}$$

The *ideal of leading terms* of $\mathcal{I}$ is the ideal spanned by $LT(\mathcal{I})$. By a slight abuse of notation, we write it $\langle LT(\mathcal{I})\rangle$. In general, given an ideal $\mathcal{I} = \langle f_1, \ldots, f_m\rangle$, the ideal of leading terms of $\mathcal{I}$ is not necessarily equal to $\langle LT(f_1), \ldots, LT(f_m)\rangle$, as the former may be strictly bigger than the latter.

Definition 10.13. Fix a monomial order $<$. A finite subset $G = \{g_1, \ldots, g_t\}$ of an ideal $\mathcal{I}$ is said to be a **Gröbner basis** for $\mathcal{I}$ with respect to $<$ if:

$$\langle LT(g_1), \ldots, LT(g_t)\rangle = \langle LT(\mathcal{I})\rangle$$

Proposition 10.5. *If $G = \{g_1, \ldots, g_s\}$ is a Gröbner basis of $\mathcal{I} \subset \mathbb{K}[x_1, \ldots, x_n]$, then $\mathcal{I} = \langle G\rangle$.*

A Gröbner basis is thus a "good" generating set of an ideal. In the remaining of this section, we shall explore further what good properties they have, and why there are so useful.

Proposition 10.6. *Let $\mathcal{I} \subset \mathbb{K}[x_1, \ldots, x_n]$, $<$ a monomial order, and G a finite subset of $\mathcal{I}$. Then G is a Gröbner basis with respect to $<$ if for all $f \in \mathcal{I}$, there exists $g \in G$ such that $LT(g)$ divides $LT(f)$.*

The *Hilbert Basis Theorem* [CLO91, chapter 2, §2, theorem 4] states that every ideal of $\mathbb{K}[x_1, \ldots, x_n]$ admits a Gröbner basis. This shows in passing that all multivariate ideals are finitely generated, a non-obvious, powerful and strong mathematical result. One of the crucial features of Gröbner bases is that the result of the multivariate division algorithm is *uniquely defined* when dividing by a Gröbner basis.

Lemma 10.7. *Let $G = \{g_1, \ldots, g_s\}$ be a Gröbner basis for an ideal $I \subset \mathbb{K}[x_1, \ldots, x_n]$ and let f be a polynomial in $\mathbb{K}[x_1, \ldots, x_n]$. Then there is a unique $r \in \mathbb{K}[x_1, \ldots, x_n]$ such that:*

(i) No term of r is divisible by any of $LT(g_1), \ldots, LT(g_s)$.

(ii) There is $g \in \mathcal{I}$ such that $f = g + r$.

We give the proof of this result (taken from [CLO91]), as it illustrates the connection between the definition of Gröbner bases and the division algorithm, besides being relatively simple.

Proof. The existence of such an r is guaranteed by the division algorithm. To prove uniqueness, suppose that $f = g + r = g' + r'$ both satisfy the conditions of the lemma. Then $r - r' = g - g' \in I$, so that if $r \neq r'$, then $LT(r - r') \in \langle LT(\mathcal{I})\rangle = \langle LT(g_1), \ldots, LT(g_s)\rangle$ (because G is a Gröbner basis). Now $\langle LT(\mathcal{I})\rangle$ is a *monomial ideal* (*i.e.*, an ideal generated solely by monomials), and it is not difficult to check that the monomials belonging to a monomial ideal are precisely the multiples of the generators. This means that $LT(r - r')$ is a multiple of some $LT(g_i)$, but this is impossible because no terms of r, r' is divisible by one of $LT(g_1), \ldots, LT(g_s)$. Thus, $r - r'$ must be zero. □

Note that the uniqueness of the remainder means that provided we know a Gröbner basis of an ideal $\mathcal{I}$, we may use it to decide the Ideal Membership problem (but we do not know with which complexity yet).

For various reasons, it is often *very* convenient to work with homogeneous ideals. We have already seen that considering the homogenized version of an ideal $\mathcal{I}$, we may revert to the original by fixing the extra variable to one. The good thing is that this operation preserves Gröbner bases.

Theorem 10.8 ([BKW93], lemma 10.57, (*vii*))**.** *Let $\mathcal{I}$ be an ideal of $\mathbb{K}[x_1, \ldots, x_n]$, $\mathcal{I}^h$ its homogenized version and G^h a Gröbner basis of $\mathcal{I}^h$. Then setting the last variable of all the polynomials of G^h to 1 yields a Gröbner basis of $\mathcal{I}$.*

It is natural to question the unicity of Gröbner bases. In vector spaces, bases are not unique, but echelonized bases are unique. Gröbner bases are not unique, and can even be somewhat redundant. However, while the situation is again a bit more complicated, it is possible to define an analogous notion of "echelonized" Gröbner basis.

Definition 10.14. A **minimal Gröbner basis** for a polynomial ideal $\mathcal{I}$ is a Gröbner basis such that:

(*i*) $LC(p) = 1$ for all $p \in G$.

(*ii*) For all $p \in G$, $LT(p) \notin \langle LT(G - \{p\})\rangle$.

To compute a minimal Gröbner basis, we may simply compute a Gröbner basis and remove from it all the polynomials not matching the criterion. Even like that, there may be several minimal Gröbner bases, but it is possible to single one out.

Definition 10.15. A **reduced Gröbner basis** for a polynomial ideal $\mathcal{I}$ is a Gröbner basis such that:

(*i*) $LC(p) = 1$ for all $p \in G$.

(*ii*) For all $p \in G$, no monomial of p lies in $\langle LT(G - \{p\})\rangle$.

If $\mathcal{I} \neq \{0\}$ is a polynomial ideal, then given a monomial order, $\mathcal{I}$ has a unique reduced Gröbner basis [CLO91, chapter 2, §7, proposition 6]. Most computer algebra systems compute reduced Gröbner bases. This is convenient, if only because then comparing the reduced Gröbner bases allows to decide whether two ideals are equal.

10.3.4 Gröbner Bases as "Linear" Bases

Following [Laz83], we outline the connection between Gröbner basis and Gaussian elimination. Consider an ideal $\mathcal{I} = \langle f_1, \ldots, f_t\rangle$. It can be seen as a $\mathbb{K}$-vector space of infinite dimension, spanned by $m \cdot f_i$ when i ranges across $1, \ldots, t$ and m ranges across all monomials. As a vector space, $\mathcal{I}$ does not have a finite basis, but it is possible to write down a program that returns in finite time a sparse representation of the i-th vector of the basis. On the (infinite) basis of $\mathbb{K}[x_1, \ldots, x_n]$ consisting of all the monomials, the generators of $\mathcal{I}$ define an infinite matrix, each row of which represent a given $m \cdot f_i$. Each row thus has a finite number of non-zero entry (as many as f_i). On the column corresponding to some monomial m, the only coefficient that appear are those in $m' f_i$, where m' divides m. This shows that they are also in finite number.

This finiteness property makes it possible to triangulate the matrix by row operations. This is a finite process for each pivot, which enumerates a basis of $\mathcal{I}$ as a vector space. Another way of expressing the same idea would be to say that a the list of basis vectors of $\mathcal{I}$ is *coinductive* and can be enumerated by a *corecursive* function. The whole list cannot be generated in finite time, but the "next" basis vector can be obtained in finite time. However, while a "linear" basis of $\mathcal{I}$ (*i.e.*, a basis of $\mathcal{I}$ seen as a vector space) is not finite, it admits a finite description, and (big surprise!) Gröbner bases are showing up again. Notice the connection between the following result and proposition 10.6.

Proposition 10.9 ([Laz83])**.** *$G = \{g_1, \ldots, g_s\}$ is a Gröbner basis of the ideal $\mathcal{I} = \langle f_1, \ldots, f_t\rangle$ if and only if the following set of polynomials is a basis of $\mathcal{I}$ seen as a vector space*

$$\{mf_i \mid 1 \leq i \leq t, \quad m \text{ is a monomial}, \quad LT(mf_i) \text{ is not a multiple of } LT(f_j) \text{ for } j < i\}$$

10.4 Solving Polynomial Systems Using Gröbner Bases

Given a set of polynomial equations, the killer tool to derive its solution is a Gröbner basis of the ideal spanned by the equations with respect to the lexicographic order. This is notably due to the following theorem.

Theorem 10.10 (The Elimination Theorem)**.** *Let $\mathcal{I} \subset \mathbb{K}[x_1, \ldots, x_n]$ be an ideal and let G be a Gröbner basis of $\mathcal{I}$ with respect to lexicographic order $x_1 > x_2 > \cdots > x_n$. Then, for every $0 \leq \ell \leq n$, the set*

$$G_\ell = G \cap \mathbb{K}[x_\ell, \ldots, x_n]$$

is a Gröbner basis of the ℓ-th ***elimination ideal*** *$\mathcal{I}_\ell = \mathcal{I} \cap \mathbb{K}[x_\ell, \ldots, x_n]$.*

Given a Gröbner basis G for an ideal $\mathcal{I} = \langle f_1, \ldots, f_s \rangle$ with respect to the lexicographic order, $G \cap \mathbb{K}[x_n]$ is a Gröbner basis of the (potentially empty) *univariate* ideal $\mathcal{I} \cap \mathbb{K}[x_n]$. All the solutions of $f_1 = 0, \ldots, f_s = 0$ in particular satisfy the univariate polynomial equation(s) in x_n contained in the n-th elimination ideal $G \cap \mathbb{K}[x_n]$. If this last ideal is not empty, then we can find all the possible values of the last variable by determining the roots of univariate polynomial(s). We can then substitute these values in the bivariate equations in $G \cap \mathbb{K}[x_{n-1}, x_n]$, thus making them univariate. Solving in x_{n-1} will reveal all the possible values of both x_{n-1} and x_n. The process can be repeated with x_{n-2}, then $x_{n-3}, \ldots, x_1$. The only problem is that at some point the ℓ-th elimination ideals may not reveal more information than the previous ones, so that we would not be able to determine the possible values of the ℓ-th variable. We would then get stuck.

10.4.1 Zero-Dimensional Ideals

This issue is nicely resolved when the equations have a finite number of solutions. Generally speaking, it is possible to associate a notion of *dimension* to ideals and affine varieties that generalizes the usual notion in linear algebra, but we do not want to get into too much detail. The important point is that affine varieties associated to ideals of dimension zero only contain a finite number of points in the algebraic closure of the field. Conversely, the affine variety associated to a positive-dimensional ideal has an infinite number of points in the algebraic closure of the field. Most if not all polynomial systems we will encounter in the next chapter are zero-dimensional. The Gröbner bases of such ideals have special properties, resolving the above problem.

Theorem 10.11 ([BKW93], theorem 6.54)**.** *Let $\mathcal{I} \subset \mathbb{K}[x_1, \ldots, x_n]$ be a proper ideal, and $\mathbb{K}$ be algebraically closed. Then the following conditions are equivalent:*

(i) $\mathbf{V}(\mathcal{I})$ *is finite*

(ii) $\mathcal{I}$ *is of dimension zero*

(iii) For every ordering and every Gröbner basis G of $\mathcal{I}$ with respect to the chosen ordering there exist, for each $1 \leq i \leq n$, $g_i \in G$ with $LT(g_i) = X_i^{\nu_i}$, for some $0 < \nu_i \in \mathbb{N}$.

This guarantees that $G \cap \mathbb{K}[x_\ell, \ldots, x_n]$ always contains an equation of the form $x_\ell^\nu + \ldots$. Therefore, substituting the possible values of $x_{\ell+1}, \ldots, x_n$ into the ℓ-th elimination ideal does result in (at least) one polynomial equation(s) in x_ℓ, which thus yields the set of possible values of x_ℓ.

If $\mathbb{K}$ is finite, then any system of polynomial equations has a finite number of solutions, but it does *not* mean that the ideal spanned by the equations is zero-dimensional, because the field is obviously not algebraically closed. We may of course always embed our equations in the algebraic closure of $\mathbb{K}$, but then the number of solutions may become infinite. Theorem 10.11 tells us that when the number of solutions in the algebraic closure is finite, then we may expect any Gröbner basis to have the nice shape property from item *(iii)*. Note that the number of solutions in the algebraic closure may very well be exponentially bigger than the number of solutions in the base field. This is essentially a problem when using the FGLM order-changing algorithm discussed below.

When $\mathbb{K} = \mathbb{F}_q$ is a finite field, it is possible to enforce that the solutions in the algebraic closure are exactly those over the ground field. This is achieved by adding to the system the so-called *field equations*, *i.e.*, the equations $x_i^q - x_i = 0$, where q is the cardinality of the field. If $\mathbb{L}$ is an overfield of $\mathbb{F}_q$, then the only elements of $\mathbb{L}$ satisfying $x^q - x = 0$ are the elements of $\mathbb{F}_q$. Therefore, any ideal containing the field equations is guaranteed to be zero-dimensional.

Adding the field equations is usually beneficial to the complexity of Gröbner basis computations when the field is "not too big" compared to the degree of the polynomials reached during the computation. When $q = 2$, it often speeds up Gröbner basis computations quite a lot. When q is large (say $q = 256$), it may however slow them down.

10.4.2 Worst-Case Complexity of Gröbner Bases Computation

We will not describe algorithms to compute Gröbner bases. First of all, the description of the simplest one, Buchberger's algorithm [Buc65] [2], can be found in any good textbook [CLO91, BKW93]. The descriptions of the more sophisticated ones, Faugère's $\mathbf{F_4}$ and $\mathbf{F_5}$ can be found in [Fau99, Fau02]. Our main reason not to describe these algorithms is that it actually reflects the way we have been thinking about them: we used them as black-boxes in a more general polynomial system-solving procedure. Their only property that matters to us is their running time.

We will discuss this delicate issue more in-depth later, and we first mention some general results for the sake of completeness. Ideal Membership is **EXPSPACE**-complete over $\mathbb{Q}$ [May89]. In addition, there are infinite families of ideals, over any field, such that any Gröbner basis thereof contains a polynomial of degree $2^{2^{\Omega(n)}}$ (resp. contains $2^{2^{\Omega(n)}}$ polynomials) [MM82, Huy86] [3]. Later on, Kühnle and Mayr gave an exponential-space (but completely impractical) algorithm to compute Gröbner bases [KM96], thereby matching the space lower-bound.

When dealing with *homogeneous ideals* (*i.e.*, ideals generated by homogeneous polynomials), things are a bit better, since Homogeneous Ideal Membership is "only" **PSPACE**-complete [May95]. Unfortunately, computing a Gröbner basis provably requires exponential space (thus doubly exponential time) in this case.

These alarming results are *worst-case* results though, and they do not fully reflect the "average" behavior of Gröbner basis algorithm. The oldest and simplest Gröbner basis algorithm was designed by Buchberger [Buc65], and can be practical under certain circumstances (but may take more than exponential space under some others). Faugère's $\mathbf{F_4}$ algorithm [Fau99] is essentially an smart reformulation of Buchberger's algorithm using efficient linear algebra to perform polynomial reductions in the spirit of Lazard [Laz83]. It is often several orders of magnitude faster than Buchberger's algorithm when implemented with care, and several implementations of very high quality are available commercially, most notably in the MAGMA computer algebra system [BCP97]. Faugère's $\mathbf{F_5}$ algorithm [Fau02] is somewhat different, and uses a more sophisticated criterion than its predecessors to avoid useless computations. Only toy implementations of $\mathbf{F_5}$ are available for cryptographic purposes though. A simplified version of $\mathbf{F_5}$ is very cleanly described in Bardet's thesis [Bar04], but is less efficient than the full-fledged algorithm.

10.4.3 D-Gröbner Bases

Sometimes the computation of a full Gröbner basis is not necessary to work with a given ideal. When dealing with homogeneous ideals, a satisfying notion of "partial Gröbner Basis" is available.

Definition 10.16. A finite subset G of a homogeneous ideal $\mathcal{I}$ is a **D-Gröbner basis** if for any f of degree at most D, $LT(f)$ is divisible by $LT(g_i)$ for some $g_i \in G$ (or equivalently if the division of every polynomial f of degree at most D in $\mathcal{I}$ by G always yields a zero remainder).

Thus, a D-Gröbner basis is capable of deciding the Ideal Membership problem for polynomials of degree at most D. In addition, D-Gröbner bases can be computed in time polynomial in n^D, for instance, by dropping any computation with polynomial of degree higher than d in Buchberger's Algorithm. In fact, looking a bit into proposition 10.9 results in an algorithm to compute a D-Gröbner basis of the homogeneous ideal $\langle f_1, \ldots, f_t \rangle$ [Laz83, Bar04]:

1. Construct the *Macaulay matrix of degree* d, *i.e.*, the matrix M_d whose rows are the products $m \cdot f_i$ of degree at most d (where m ranges across all the monomials of the right degree)
2. Compute its reduced row-echelon form $\widetilde{M_d}$
3. Define G_d to be the set of polynomials found on the rows of $\widetilde{M_d}$ whose leading monomial does not match that of the same line of M_d.
4. Then $G = \bigcup_{d=0}^{D} G_d$ is a D-Gröbner basis

Given a homogeneous ideal $\mathcal{I}$ and a monomial ordering, there is a degree $D_{\max}$ such that a $D_{\max}$-Gröbner basis of $\mathcal{I}$ is in fact a "normal" Gröbner basis of $\mathcal{I}$. This $D_{\max}$ can be chosen to be the highest degree of any polynomial in a Gröbner basis of $\mathcal{I}$ with respect to the ordering. It follows that using the above Gaussian-elimination technique, a full Gröbner basis can be computed in time $\mathcal{O}\left((n + D_{\max})^{D_{\max}}\right)$.

2. By the way, Buchberger named the objects computed by his algorithm algorithm after his advisor Wolfgang Gröbner

3. While this apparently contradicts the fact that the problem is decidable in **EXPSPACE**, the contradiction is only apparent. In the model of space-bounded computation, an exponentially-space-bounded Turing machine is allowed to take doubly-exponential time to write a result of doubly-exponential size to the output tape, while using only a simply exponential working memory.

Looking at his algorithm reveals why the XL algorithm [CKPS00] invented by cryptographers is in fact essentially a Gröbner basis computation algorithm known for about twenty years, as was pointed out in [AFI+04].

10.4.4 Hilbert Function and the Degree of Regularity

To understand the complexity of Gröbner basis computation, it seems important to look into this degree $D_{\max}$ defined just above. The set of all homogeneous polynomials of degree exactly d is not a vector space because it misses zero. We therefore consider the following set:

$$\mathbb{K}[x_1, \ldots, x_n]_d = \{0\} \cup \left\{ f \in \mathbb{K}[x_1, \ldots, x_n] \mid f \text{ is homogeneous of degree } = d \right\}$$

And $\mathbb{K}[x_1, \ldots, x_n]_d$ is a vector-space. This allows us to define the vector space of homogeneous polynomials of degree d in $\mathcal{I}$:

$$\mathcal{I}_d = \mathcal{I}_d \cap \mathbb{K}[x_1, \ldots, x_n]$$

When $\mathcal{I}$ is homogeneous, then $\mathcal{I}_d$ is the row-space of the Macaulay matrix of degree d. This matrix has $\binom{n+d}{d}$ columns, because this is the number of degree-d monomials on n variables. Yet, its rank is $\dim \mathcal{I}_d$, independently of the number of rows. There are indeed often non-trivial linear dependencies between the rows of the Macaulay matrices: if $f_i = \sum \alpha_k m_k$ and $f_j = \sum \beta_k m'_k$ are both degree-d generators of $\mathcal{I}$, where the m_k's are monomials, then $f_i f_j = \sum \alpha_k m_k f_j = \sum \beta_k m'_k f_i$. As a consequence, we find that $\sum \alpha_k m_k f_j - \sum \beta_k m'_k f_i = 0$ in the Macaulay matrix of degree $2d$. Gauss-Reducing the Macaulay matrices is thus an inefficient way to compute Gröbner bases, as it will often produce zero rows (just like Buchberger's algorithm will reduce S-polynomials to zero).

Definition 10.17. The **Hilbert function** of an homogeneous ideal $\mathcal{I} \subset \mathbb{K}[x_1, \ldots, x_n]$ is the function on the nonnegative integers d defined by

$$HF_{\mathcal{I}}(d) = \dim \mathbb{K}[x_1, \ldots, x_n]_d - \dim \mathcal{I}_d = \binom{n+d}{d} - \dim \mathcal{I}_d$$

When $\mathcal{I}$ is a *monomial ideal* (*i.e.*, an ideal generated by monomials only), then the Hilbert function counts the number of monomials of degree d *not* contained in $\mathcal{I}$. It is known that the Hilbert function coincide with a certain polynomial for sufficiently large values of d. This polynomial is called the *Hilbert polynomial*. It follows from proposition 10.11 that when $\mathcal{I}$ is a homogeneous ideal of dimension zero, then the Hilbert polynomial is identically zero.

Definition 10.18. The **degree of regularity** of $\mathcal{I}$ (also called **index of regularity**) is the smallest integer s_0 such that for all $s \geq s_0$ the Hilbert function is equal to the Hilbert polynomial. We denote it by $H(I)$ in the sequel (but note that it depends on the ordering).

It plays a crucial role in the complexity of many algorithms operating on ideals.

Proposition 10.12. *A reduced $H(I)$-Gröbner basis of a zero-dimensional homogeneous ideal $\mathcal{I}$ is in fact a full Gröbner basis of $\mathcal{I}$.*

Proof. Let G be the reduced Gröbner basis of $\mathcal{I}$, f be a polynomial of G of maximum degree, and d be its degree. Let us reason by contradiction and assume that $d > H(I)$. We will show that $LT(f)$ belongs to $\langle LT(G - \{f\})\rangle$, thus contradicting the fact that G is minimal (and reduced).

Because $\mathcal{I}$ is zero-dimensional, its Hilbert Polynomial is identically zero, and therefore $\mathcal{I}_{H(\mathcal{I})}$ is in fact equal to $\mathbb{K}[x_1, \ldots, x_n]_{H(\mathcal{I})}$. As a consequence, there is in $\mathcal{I}_{H(\mathcal{I})}$ a polynomial g whose leading monomial divides that of f. By definition of $\mathcal{I}_d$, g can be written as a polynomial combination of $G - \{f\}$ (because f is of greater degree than g). This shows that $LT(g) \in \langle LT(G - \{f\})\rangle$, and contradicts the minimality of G. □

10.4.5 Generic Properties

The depressing worst-case results mentioned above suggest that $H(\mathcal{I})$ can become exponential in some cases. However, the situation is *generically* better than what the worst-case result suggest.

Definition 10.19. Consider the vector space $\mathbb{K}^m$. A property $\mathcal{P}$ is **generically true** over $\mathbb{K}^m$ if it is true on all points of $\mathbb{K}^m$ except on a given proper affine variety (or, equivalently, a property is generic if it is true on a Zariski open set of $\mathbb{K}^m$).

The intuition is that generic properties are true "most" of the time. A non-trivial but interesting example: $n \times n$ matrices are *generically invertible* over any field $\mathbb{K}$. Indeed, non-invertible matrices have a zero determinant, and the determinant is a polynomial in the matrix coefficients. This means that "most" matrices are invertible, and this result is consistent with our expectations: invertible matrices over $\mathbb{R}$ form a *dense* subset of all the matrices over $\mathbb{R}$. Indeed, if we choose "at random" the coefficients of a matrix over $\mathbb{R}$, the probability that the polynomoial equation $\det = 0$ holds is negligible.

A funny consequence of the definition of generic properties, is that over an infinite field, if a given property is true except on an algebraic variety, then it suffices to show that it is true on one point to show that it is true *almost* everywhere.

This reasoning clearly does not extend to finite fields. The fact that a generic property holds *almost everywhere* is clearly not true, because the algebraic variety over which the property is not true represent a *fixed fraction* of the sampling space. A Generic property is thus true with some (unknown) probability. This is examplified again by invertible matrices: over $\mathbb{F}_{2^8}$, more than 99.5% of all matrices are invertible, yet over $\mathbb{F}_2$, on the other hand, only 28.88% of the matrices are invertible.

This example illustrates simultaneously the power and the limitations of this concept: the result about invertible matrices is mostly valid and makes sense, but we may have to be cautious, especially over small finite field. In this latter case, we expect generic property to hold with a non-negligible probability, but in general this is the best we can say.

10.4.6 Generic Behavior of Gröbner Bases

We are now ready to discuss the generic properties of Gröbner bases.

Theorem 10.13 ([Laz83], theorem 1)**.** *Let $\mathcal{I} = \langle f_1, \ldots, f_k \rangle \subset \mathbb{K}[x_1, \ldots, x_n]$ be a homogeneous ideal of dimension zero, where f_i is of degree d_i and $k \leq n$. Then after a generic change of variables, every Gröbner base for the lexicographic order contains a polynomial of degree $d_1 d_2 \ldots d_k$*

Theorem 10.14 ([Laz83], theorem 2)**.** *Let $\mathcal{I} = \langle f_1, \ldots, f_k \rangle \subset \mathbb{K}[x_1, \ldots, x_n]$ be a homogeneous ideal of dimension zero, where f_i is of degree d_i and $k \leq n$. Then after a generic change of variables, the elements of any minimal reduced Gröbner base for the grevlex ordering have degree at most*

$$\sum_{i=1}^{k} d_i - n + 1.$$

The bound of the second theorem was first given by Macaulay in the context of his work on multivariate resultants. Theorem 10.14 tells us that computing a Gröbner basis is in fact simply exponential "most of the time" when dealing with homogeneous ideals of dimension zero. This comes from the fact that we may obtain an actual Gröbner basis with respect to the grevlex order by computing only a D-Gröbner basis of the ideal, where D is the bound of the second theorem. When dealing with with n quadratic equations in n variables, theorem 10.14 tells us that we can (generically) compute a Gröbner basis by crunching polynomials of degree at most $2k - n + 1$. This is less than $n + 1$, and the bound is reached when we have $k = n$ equations.

10.4.7 Influence of the Ordering

The two previous theorems suggest that the choice of the ordering has a deciding influence on the complexity of computing a Gröbner basis. Consider for instance the ideal:

$$\mathcal{I} = \left\langle x^5 + y^4 + z^3 - 1, x^3 + y^3 + z^2 - 1 \right\rangle$$

A Gröbner basis of $\mathcal{I}$ with respect to the grevlex order is:

$$\begin{aligned} G \quad = \big\{ & y^6 + xy^4 + 2y^3z^2 + xz^3 + z^4 - 2y^3 - 2z^2 - x + 1, \\ & x^2y^3 - y^4 + x^2z^2 - z^3 - x^2 + 1, \\ & x^3 + y^3 + z^2 - 1 \big\} \end{aligned}$$

A Gröbner basis of $\mathcal{I}$ with respect to the lexicographic order, on the other hand, is too messy to be shown. One of its polynomial has 282 terms, total degree 25, and a largest coefficient of 170255391...

In most cases, the preferred strategy to compute a (more useful) lexicographic Gröbner basis is to first compute a Gröbner basis for the grevlex order, and then use an *ad hoc* to *change the order* of the Gröbner basis.

10.4.8 Order Changing Algorithms

There are two algorithms to change the order of an already-computed Gröbner basis: the FGLM algorithm [FGLM93] and the Gröbner Walk [CKM97]. The former only applies to zero-dimensional ideals, while the latter can always be used. However, except in very special cases its complexity is unknown, while the complexity of FGLM is well-understood.

Theorem 10.15 ([FGLM93]). *Let $\mathcal{I}$ be a zero-dimensional ideal, G_1 a reduced Gröbner basis of $\mathcal{I}$ with respect to $<_1$. There is an algorithm that, given G_1, computes a Gröbner basis of $\mathcal{I}$ with respect to $<_2$ with $\mathcal{O}\left(n \cdot D^3\right)$ arithmetic operations in $\mathbb{K}$, where D is the number of solutions of $\mathcal{I}$ in the algebraic closure of $\mathbb{K}$.*

Over finite fields, adding the field equations helps keeping D as low as possible, and thus makes the FGLM algorithm work faster.

10.5 Regular and Semi-Regular Sequences

What makes the behavior of the usual Gröbner basis algorithms difficult to understand is the possibility that the sequence of actions taken by the algorithm depends very much on the coefficients of the input polynomial system. This dependency also exist in, say, the euclidean algorithm to compute the Greatest Common Divisor, or in the Gaussian algorithm to put a matrix in row-echelon-form, but these cases the dependency less critical (*i.e.*, the Gaussian algorithm will perform $\mathcal{O}\left(n^3\right)$ operations no matter what the actual coefficients of the matrix are). In particular, *algebraic relations* between the coefficients of the polynomial may lead to unpredictable numerical cancellation in the course of the algorithm, or unpredictable polynomial relations where there should not be any.

10.5.1 Regular Sequences.

However, the behavior of the usual algorithms is better understood when these unfortunate events are guaranteed *not* to happen. This lead to the definition of sequence of polynomials that "behave".

Definition 10.20. Let $R = \mathbb{K}[x_1, \ldots, x_n]$. The sequence $f_1, \ldots, f_m \in R$ is a **regular sequence** if:

(*i*) $\langle f_1, \ldots, f_m \rangle \neq R$

(*ii*) for all $1 \leq i \leq m$, and all $g \in R$, then

$$g f_i \in \langle f_1, \ldots, f_{i-1} \rangle \Longrightarrow g \in \langle f_1, \ldots, f_{i-1} \rangle$$

This is equivalent to say that f_k must not be a zero divisor in $R/\langle f_1, \ldots, f_{k-1} \rangle$, or that multiplication by f_k must be injective on the quotient.

The notion of regular sequence of polynomials generalizes that of linearly independent sequence of vectors. If a family of vectors $\mathbf{x}_1, \ldots, \mathbf{x}_t$ are linearly independent, then there are no non-trivial linear relations of the form $\alpha_1 \mathbf{x}_1 + \cdots + \alpha_t \mathbf{x}_t = 0$. If $f_1, \ldots, f_m$ is a regular sequence, then there are no "non-trivial" polynomial relations between the f_i's. Of course the situation is more complicated, because some polynomial relations always exist: if we pick $P_k \in \langle f_1, \ldots, f_{k-1} \rangle$, then by definition of ideals $P_k f_k \in \langle f_1, \ldots, f_{k-1} \rangle$, and there exist a sequence of polynomials $P_1, \ldots, P_{k-1}$ such that $P_1 f_1 + \cdots + P_{k-1} f_{k-1} + P_k f_k = 0$. Such relations always exist, and are therefore "trivial".

If the f_i's are regular, then all polynomial relations between the f_i's are "trivial". More precisely, if there exist a sequence of P_i's such that $P_1 f_1 + \cdots + P_{k-1} f_{k-1} + P_k f_k = 0$, then by definition of ideal $P_k f_k$ belongs to $\langle f_1, \ldots, f_{k-1} \rangle$. In other terms, $P_k f_k$ is congruent to zero modulo this ideal. Because multiplication by f_k is injective in the quotient, then P_k itself is necessarily congruent to zero, *ergo* P_k belongs to $\langle f_1, \ldots, f_{k-1} \rangle$, and the polynomial relation is trivial. Regular sequences enjoy many additional nice properties, summarized in the following theorem.

Theorem 10.16 ([Bar04]). *Let $f_1, \ldots, f_m$ be a regular sequence of homogeneous polynomials of respective degree d_i in $\mathbb{K}[x_1, \ldots, x_n]$, with $m \leq n$, and let $\mathcal{I} = \langle f_1, \ldots, f_m \rangle$.*

(*i*) *$\mathcal{I}$ has dimension $n - m$.*

(*ii*) *The Hilbert Series of $f_1, \ldots, f_m$ is*

$$\sum_{d=0}^{\infty} HF_{\mathcal{I}}(d) \cdot z^d = \prod_{i=1}^{m} \left(1 - z^{d_i}\right) \Big/ (1 - z)^n$$

and reciprocally, every sequence of polynomials having this Hilbert series is a regular sequence

(iii) The degree of regularity of $\mathcal{I}$ *is* $\sum_{i=1}^{k}(d_i - 1) + 1$ *(Macaulay's bound)*

(iv) An arbitrary sequence of m *polynomials is* generically *regular.*

We have seen that linearization is capable of dealing with *very* overdetermined systems in polynomial time, which suggests that overdetermined systems are *easier* to solve, for instance by having a lower degree of regularity. The framework of generic sequences unfortunately cannot be used to explore this aspect of Gröbner basis complexity. The first item of the theorem indeed implies that a regular sequence cannot have more than n elements (observe again the analogy with linearly independent families of vectors). This is further annoying because most systems encountered in cryptography are overdetermined.

10.5.2 Semi-Regular Sequences: Dealing With Overdetermined Systems

To overcome this difficulty, Bardet and Faugère extended the notion of regular sequence to overdetermined systems.

Definition 10.21. Let $R = \mathbb{K}[x_1, \ldots, x_n]$. The sequence of homogeneous polynomials $f_1, \ldots, f_m \in R$ is a **semi-regular sequence** if:

(i) $\mathcal{I} = \langle f_1, \ldots, f_m \rangle \neq R$

(ii) for all $1 \leq i \leq m$, and all $g \in R$, then

$$gf_i \in \langle f_1, \ldots, f_{i-1} \rangle \text{ and } \deg(gf_i) < H(\mathcal{I}) \Longrightarrow g \in \langle f_1, \ldots, f_{i-1} \rangle$$

Informally speaking, semi-regular sequences behave like regular sequences "up to the degree of regularity". The polynomials in a semi-regular sequence are thus "as independent as possible".

Theorem 10.17 ([Bar04])**.** *The sequence* $f_1, \ldots, f_m$ *of homogeneous polynomials of respective degrees* $d_1, \ldots, d_m$ *in* $\mathbb{K}[x_1, \ldots, x_n]$ *is semi-regular if and only if the Hilbert series of the corresponding ideal is*

$$\left[\prod_{i=1}^{m}\left(1 - z^{d_i}\right)^m \Big/ (1-z)^n\right],$$

where $\left[\sum a_i z^i\right] = \sum b_i z^i$ *with* $b_i = a_i$ *when* $a_j > 0$ *for all* $j \leq i$*, and* $b_i = 0$ *otherwise.*

It is shown in [Bar04] that the set of semi-regular sequences of m polynomial in $\mathbb{K}[x_1, \ldots, x_n]$ is a Zariski open set. The problem is to show that it is non-empty for any value of n and m. An explicit example is known when $m = n + 1$ [Frö85], but no general example are known for other values of m. In addition, generic properties has little meaning other the finite fields that interest us. It is conjectured, on the basis of extensive experiments, in [Bar04] that the proportion of semi-regular systems becomes 1 when n goes to $+\infty$. Therefore, we will often assume that for large n a random system is almost surely semi-regular (which is certainly not a best-case assumption, as it means that the system is not easier to solve than the average).

It follows from theorem 10.17 that when $f_1, \ldots, f_m$ are homogeneous and form a semi-generic sequence, then the degree of regularity of $\langle f_1, \ldots, f_m \rangle$ is the smallest degree d such that the coefficient of degree d in the series expansion of $\prod_{i=1}^{m}\left(1 - z^{d_i}\right)^m/(1-z)^n$ is not strictly positive. This property enables an explicit computation of the degree of regularity of semi-regular sequences for given values of m and n, thus allowing to determine the complexity of computing a Gröbner basis.

Furthermore, Bardet *et al.* [Bar04, BFSY05] give asymptotic developments of the expression of the degree of regularity in the case of $\alpha \cdot n$ equations in n variables, for any constant α greater than 1. When there are $\alpha \cdot n$ semi-regular quadratic equations in n variables, [BFSY05] gives:

$$D_{\text{reg}} = n\left(\alpha - \frac{1}{2} - \sqrt{\alpha(\alpha-1)}\right) - \frac{a_1}{2(\alpha(\alpha-1))^{\frac{1}{6}}} n^{\frac{1}{3}} - \left(2 - \frac{2\alpha - 1}{4\sqrt{\alpha(\alpha-1)}}\right) + \mathcal{O}\left(1/n^{1/3}\right),$$

$$\text{with } a_1 \approx -2.33811. \quad (10.2)$$

10.6 Complexity of Gröbner Bases Computation

These results on semi-regular sequences enable us to estimate the complexity of a Gröbner basis computation without actually running it.

We have seen that the computation of a Gröbner basis essentially amounts to echelonize a sparse matrix with M columns and potentially many more rows, where M is the number of monomials of degree D_{reg}

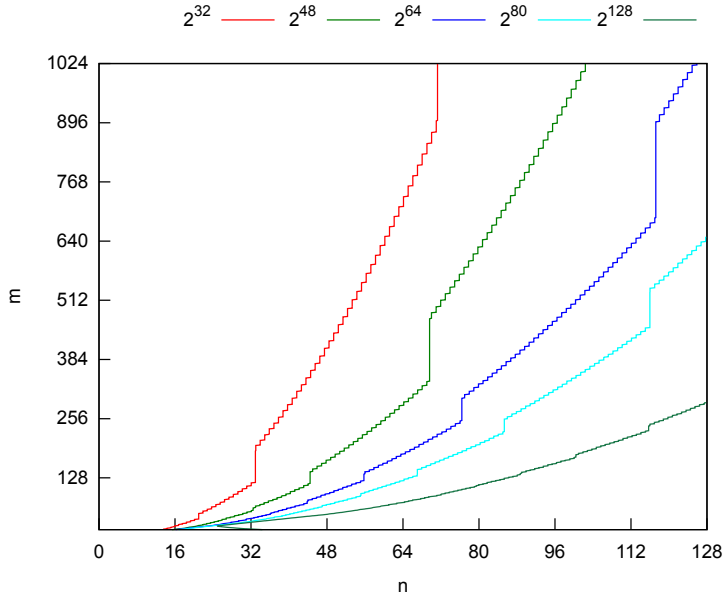


Figure 10.1: Projected number of field operations required to compute a Gröbner basis of a semi-regular sequence of m quadratic polynomials in n variables. The graph shows iso-complexity lines (top-left is lower, bottom-right is higher).

in n variables. The $\mathbf{F_5}$ algorithm tries very hard to reduce the number of rows to the minimum, and it is shown in [Fau02, Bar04] that on a regular or semi-regular sequence it produces matrices without linear dependencies between the rows. The complexity of echelonizing these matrices is roughly the equivalent of performing $\mathcal{O}\left(n^{\omega \cdot D_{\text{reg}}}\right)$ arithmetic operations in $\mathbb{K}$, with $2 < \omega \leq 3$ being the linear algebra constant, and n the number of variables of ideal considered. Because the matrices are very sparse, it is not unreasonable to consider that $\omega = 2$. A distinctive feature of Gröbner basis computation as a way of solving multivariate polynomial systems is that the complexity of computing a Gröbner basis is vastly independent from the cardinality of the field. Exhaustive search, on the other hand, becomes quickly very impractical when the size of the field increases.

In her PhD thesis [Bar04], Bardet determined that a simplified version of the $\mathbf{F_5}$ algorithm computes a Gröbner basis of a regular sequence of n quadratic polynomial in n variables in $2^{4.295n}$ arithmetic operations in $\mathbb{K}$. For the sake of comparison, echelonizing the Macaulay matrix of degree given by the Macaulay bound requires about 2^{6n} field operations. This suggests that exhaustive search is an asymptotically faster way to solve a system of n quadratic equations in n variables over $\mathbb{F}_q$ if and only if $q \leq 20$. If q is bigger, then computing a Gröbner basis will be faster. With more than n equations, Macaulay's bound is improved by equation (10.2). In any case, the exact degree of regularity can always be read off the Hilbert series. Figure 10.1 shows how the complexity of computing a Gröbner basis of a semi-regular sequence of quadratic polynomials evolves with the parameters.

10.6.1 The Case of very Overdetermined Systems

The linearization technique is capable of solving in time $\mathcal{O}\left(n^6\right)$ systems of $\approx n^2/2$ equations in n variables. It unfortunately breaks down if we have (say) only $n^2/3$ equations. How hard is this particular case? It is particularly relevant for multivariate cryptanalysis, because it is one of the rare situations where Gröbner basis computations might solve systems of multivariate quadratic equations that would not be tractable by exhaustive search or any other technique. We may always use the Hilbert series to determine the degree of regularity of corresponding semi-regular sequences and derive the complexity of a Gröbner basis computation. This for instance enables us to draw the points on Figure 10.1.

While we are well-aware that it is not theoretically justified (because equation (10.2) is established for a constant α), we now set $\alpha = \beta n$, and express d_{reg} as a function of β in (10.2). This yields

$$d_{reg}(\beta) = \frac{1}{8\beta} - \frac{a_1}{2\beta^{1/3}} - \frac{3}{2} + \mathcal{O}\left(1/n\right). \tag{10.3}$$

We compared the result predicted by this equation to the actual values (given by the Hilbert series), for sufficiently big n. Figure 10.2 shows both values. We (empirically) found the constant term of (10.3) to be

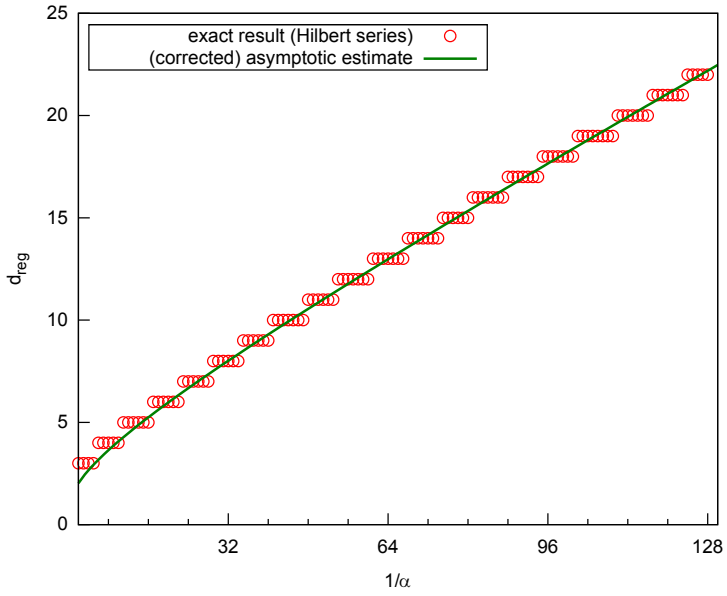


Figure 10.2: Degree of regularity (for n sufficiently big) of $\alpha \cdot n^2$ semi-regular quadratic polynomials in n variables.

inaccurate (the best fit seems to be 1/3). We therefore estimate that given $\alpha \cdot n^2$ semi-regular quadratic equations in n variables we should expect for n big enough a degree of regularity of:

$$D_{\text{reg}} = \frac{1}{8\alpha} - \frac{a_1}{2\alpha^{1/3}} + \frac{1}{3}$$

10.7 Finite Vector Spaces Combinatorics

We conclude this "toolbox" chapter by determining the probability that a uniformly random endomorphism of $(\mathbb{F}_q)^n$, potentially conditioned on meeting certain conditions, satisfies some given properties. The probability that a random endomorphism is invertible is well-known. The probability that a random endomorphism has a given rank is less well-known but is readily found in the literature [Coo00], even in the case where the entries of the matrix are not uniformly distributed. The probability that a given vector belongs to the image of a random linear map is (apparently) less well-known and much less readily available in the existing litterature. We also investigate the probability that a random endomorphism satisfies combinations of these events. We establish all these results by *counting* the number of endomorphisms meeting the conditions at hand. The results presented here are heavily inspired by Dubois's PhD thesis [Dub07].

Goldman and Rota first counted the number of subspaces of a vector space over $\mathbb{F}_q$ [GR69]. After that, the analogy between set combinatorics and finite vector space combinatorics has been investigated by many researchers. The toolbox of combinatoricians, including Möbius inversion [BG75] and the exclusion-inclusion principle [CR92], has been progressively adapted to finite vector spaces, as q-analogs of their set-combinatorics counterparts.

q-Analogs. We will try to express our results using the formalism of *q-analogs*. A q-analog is "a mathematical expression parameterized by a quantity q that generalizes a known expression and reduces to the known expression in the limit $q \to 1$" [Wei]. The starting point is the observation that:

$$\lim_{q \to 1} \frac{1 - q^n}{1 - q} = n$$

This leads to the definition of $[n]_q$ (read: "the q-bracket of n"), which is the q-analog of the integer n:

$$[n]_q = \frac{1 - q^n}{1 - q}$$

It is then possible to define q-analogs of the usual combinatorial quantities.

q-Factorial. It seems natural to define the *q-factorial* to be:

$$[n]_q! = [1]_q[2]_q \dots [n-1]_q[n]_q$$

We readily verify that we recover the usual factorial when $q \to 1$, but the analogy is in fact deeper. For instance, $[n]_q!$ is the number of *complete flags* of $(\mathbb{F}_q)^n$, *i.e.*, of sequences of subspace $S_0, \dots, S_n$ with the strict inclusions:

$$\langle 0 \rangle = S_0 \subset S_1 \subset \dots \subset S_n = (\mathbb{F}_q)^n$$

This result is easy to establish by induction on n [Sta86]: given such a flag,

$$\langle 0 \rangle = S_1/S_1 \subset S_2/S_1 \subset \dots \subset S_n/S_1 \cong (\mathbb{F}_q)^{n-1}$$

is a complete flag of $(\mathbb{F}_q)^{n-1}$. There are $[n]_q$ possibles choices for S_1: a one-dimensional vector space is fully specified by the choice of a non-zero vector, which makes $q^n - 1$ possibilities, but $q - 1$ colinear vectors span the same subspace. When $q \to 1$, we recover the usual result that $n!$ is the number of sequences of n subsets of $[1; n]$ such that:

$$\emptyset = S_0 \subset S_1 \subset \dots \subset S_n = [1; n]$$

Such sequences in fact define a total ordering of $[1; n]$, or, in other terms, a permutation.

q-Binomial. Armed with the q-factorial, we may define the *q-binomial coefficient* just like the usual thing:

$$\binom{n}{k}_q = \frac{[n]_q!}{[k]_q![n-k]_q!}$$

It is well-known that $\binom{n}{k}$ is the number of k-element subsets of $[1; n]$. We claim that $\binom{n}{k}_q$ is the number of k-dimensional subspaces of $(\mathbb{F}_q)^n$. To see why, let E_k be a subspace of dimension k, and observe that $[n]_q!$ is the product of the number of k-dimensional subspaces and of the number of complete flags of $(\mathbb{F}_q)^n$ such that $S_k = E_k$. Such a complete flag completely determines (and is completely determined by) two complete flags:

$$\begin{array}{ccccccccccc} \langle 0 \rangle & = & S_0 & \subset & S_1 & \subset & \dots & \subset & E_k & \cong & (\mathbb{F}_q)^k \\ \langle 0 \rangle & = & E_k/E_k & \subset & S_{k+1}/E_k & \subset & \dots & \subset & S_n/E_k & \cong & (\mathbb{F}_q)^{n-k} \end{array}$$

Therefore, the number of complete flags of $(\mathbb{F}_q)^n$ such that $S_k = E_k$ is $[k]_q![n-k]_q!$, and the expression of the q-binomial coefficient is established. A different but equally interesting way to establish the same result would be to remark that

$$\binom{n}{k} = \frac{\#\text{sequences of } k \text{ distinct elements in an } n\text{-set}}{\#\text{sequences of } k \text{ distinct elements in a } k\text{-set}},$$

and showing that:

$$\binom{n}{k}_q = \frac{\#\text{sequences of } k \text{ independent vectors in } (\mathbb{F}_q)^n}{\#\text{sequences of } k \text{ independent vectors in } (\mathbb{F}_q)^k}.$$

It is also notable that the q-binomial coefficients enjoy q-analog versions of the usual identities:

$$\begin{aligned} \binom{n}{k}_q &= \binom{n}{n-k}_q \\ \binom{n}{k}_q &= \frac{q^n - 1}{q^k - 1}\binom{n-1}{k-1}_q \\ \binom{n}{k}_q &= q^k\binom{n-1}{k}_q + \binom{n-1}{k-1}_q \\ \binom{n}{k}_q &= \binom{n-1}{k}_q + q^{n-k}\binom{n-1}{k-1}_q \end{aligned}$$

Automorphisms of $(\mathbb{F}_q)^n$. The number of automorphisms of $(\mathbb{F}_q)^n$ is easy to derive from the q-analogs. An automorphism L determines in a unique way a complete flag of $(\mathbb{F}_q)^n$, by setting $S_i = S_{i-1} + \langle L(e_i) \rangle$. However, the converse is not true, and many possible L's generate the same flag. Given a complete flag, we consider the n one-dimensional subspaces L_i of $(\mathbb{F}_q)^n$ such that L_i is a supplementary of S_{i-1} in S_i, for $i = 1, 2, \dots, n$ (the L_i's are not unique but we choose some). The image of L on e_i must be the sum of a

non-zero vector of L_i and of any vector of S_{i-1}. There are therefore $(q-1)q^{i-1}$ choices for $L(e_i)$. This shows that:

$$|\mathrm{GL}_n(\mathbb{F}_q)| = [n]_q! \cdot q^{n(n-1)/2} \cdot (q-1)^n$$

The q-analogy is not as satisfying as we would like, because it would seem natural that the number of automorphisms (*i.e.*, the linear permutations of an n-dimensional vector space) of $(\mathbb{F}_q)^n$ degenerates to $n!$ (*i.e.*, the number of actual permutation of an n-element set) when $q \to 1$, yet it is not the case. We therefore introduce an *ad hoc* notation for the number of automorphisms of $(\mathbb{F}_q)^n$:

$$\langle\!\langle n \rangle\!\rangle_q! = |\mathrm{GL}_n(\mathbb{F}_q)| = [n]_q! \cdot q^{n(n-1)/2} \cdot (q-1)^n$$

It must be noted that expanding this expression yields the expected:

$$\langle\!\langle n \rangle\!\rangle_q! = \prod_{i=1}^{n} \left(q^n - q^i\right)$$

This product is usually derived in the following way: since all the $L(e_i)$ must be non-zero and linearly independent, then $L(e_i)$ must be chosen in $(\mathbb{F}_q)^n - \langle L(e_1), \ldots, L(e_{i-1})\rangle$, and this set has $q^n - q^i$ elements.

Injective Linear Maps From $(\mathbb{F}_q)^k$ to $(\mathbb{F}_q)^n$. An injective linear map f from $(\mathbb{F}_q)^k$ to $(\mathbb{F}_q)^n$ uniquely determines a k-dimensional subspace E of $(\mathbb{F}_q)^n$ (its image). In turn, any basis of E determines f completely. Note that there is a one-to one correspondance between the bases of $(\mathbb{F}_q)^n$ and its automorphisms. This shows that the number of injective linear maps from $(\mathbb{F}_q)^k$ to $(\mathbb{F}_q)^n$ is $\langle\!\langle k \rangle\!\rangle_q! \binom{n}{k}_q$.

Endomorphisms of a Given Rank. We wish to count the number of endomorphisms of $(\mathbb{F}_q)^n$ whose kernel is of dimension k and contains a given subspace F (of dimension smaller than k). This result is well-known, and our exposition is vastly plagiarized from [Dub07]. We proceed in two steps, punctuated by the two following lemma.

Lemma 10.18. *The number of subspaces of $(\mathbb{F}_q)^n$ of dimension k containing a subspace F of dimension $s \leq n$ is $\binom{n-s}{k-s}_q$.*

Proof. The idea is that subspaces of $(\mathbb{F}_q)^n$ of dimension k containing F are in one-to-one correspondance with the subspaces of dimension $k-s$ of the quotient $(\mathbb{F}_q)^n / F$. Let us verify this assertion. Let G be a subspace of $(\mathbb{F}_q)^n$ containing F, and let H be a supplementary of F in G, so that $G = H \bigoplus F$, and H has dimension $k-s$. The point is that the image of F in the quotient is the trivial subspace, while the image of H is isomorphic to H (hence the one-to-one correspondance).

So, the number we are interested in is also the number of subspaces of dimension $k-s$ in the quotient, which has dimension $n-s$, and we know that this is $\binom{n-s}{k-s}_q$. □

Lemma 10.19. *The number of endomorphisms of $(\mathbb{F}_q)^n$ whose kernel a) is of dimension k and b) contains a given subspace F of dimension $s \leq k$ is:*

$$D(n,k,s) = \binom{n-s}{k-s}_q \binom{n}{k}_q \langle\!\langle n-k \rangle\!\rangle_q!$$

Proof. Let U be a subspace of $(\mathbb{F}_q)^n$ of dimension k containing F, and V a complementary subspace. Any endomorphism of $(\mathbb{F}_q)^n$ is uniquely defined by its restriction to U and V. An endomorphism admits U as its kernel if and only if it vanishes on U and is injective on V. We have seen that there are $\binom{n}{n-k}_q \langle\!\langle n-k \rangle\!\rangle_q!$ injective linear maps from V to $(\mathbb{F}_q)^n$. By lemma 10.18 there are $\binom{n-s}{k-s}$ subspaces U of dimension k containing F. The result of the lemma is the product of these two quantities (via an easy q-binomial identity), as U and the image on V can be chosen independently. □

Probability to be Invertible. The probability that a random endomorphism is invertible is a widely used result. It follows from the expression of $D(n,0,0)$ that:

$$\mathbb{P}\left[f \in \mathrm{GL}_n(\mathbb{F}_q)\right] = \frac{\langle\!\langle n \rangle\!\rangle_q!}{q^{n^2}} = \prod_{i=1}^{n} \left(1 - \frac{1}{q^i}\right)$$

Throughout the next chapters, we denote this probability by $\lambda(n)$. As a function of n, this quantity is positive and decreasing, thus it reaches a finite limit when $n \to \infty$. It is quite easy to obtain a reasonable approximation thereof:

$$\log \lambda(n) = \log \prod_{i=1}^{n} \left(1 - \frac{1}{q^i}\right) = \sum_{i=1}^{n} \log \left(1 - \frac{1}{q^i}\right) \approx -\sum_{i=1}^{n} \frac{1}{q^i} \approx \frac{1}{1-q}$$

And thus:

$$\lim_{n\to\infty} \lambda(n) \approx e^{1/(1-q)}$$

This suggest that for large values of q, a random endomorphism is invertible with probability close to one. An alternate, more rigorous, and apparently less well-known way to obtain a much better approximation is to turn the infinite product into an infinite sum without using the logarithm, but instead through Euler's *pentagonal number theorem* [Bel10]:

$$\prod_{i=1}^{\infty} \left(1 - \frac{1}{q^i}\right) = \sum_{i=-\infty}^{+\infty} (-1)^i \cdot q^{-i(3i+1)/2} \tag{10.4}$$

Considering the partial sums yields a close approximation of the asymptotic probability, for any value of q. In particular, we find:

$$\lim_{n\to\infty} \lambda(n) = 1 - \frac{1}{q} + \frac{1}{q^2} + \frac{1}{q^5} + \frac{1}{q^7} - \frac{1}{q^{12}} + \mathcal{O}\left(\frac{1}{q^{15}}\right)$$

In the same vein, we deduce from lemma 10.19 the probability that a random endomorphism vanishing on a subspace F has a kernel of dimension $k \geq \dim F$:

$$\mathbb{P}\left[\dim \ker L = k \;\middle|\; F \subseteq \ker F\right] = \frac{D(n,k,s)}{q^{n(n-s)}} = \frac{\lambda(n)\lambda(n-s)}{\lambda(k)\lambda(k-s)\lambda(n-k)} q^{-k(k-s)} \tag{10.5}$$

The interest of this expression is that the ratio of the λ expressions lives in a small interval, independently of q, n, k and s, so that the probability is of order $q^{-k(k-s)}$.

Expected Cardinality of the Kernel. A more sophisticated result, and less readily available in the available litterature, is the expected cardinality of the kernel of a random endomorphism, and its variance.

Lemma 10.20. *Let f be a uniformly random endomorphism of $(\mathbb{F}_q)^n$, vanishing on a subspace F of $(\mathbb{F}_q)^n$, with $\dim F = s$. Then:*

$$\mathbb{E}\left[|\ker f| \;\middle|\; F \subseteq \ker f\right] = q^s + 1 - \frac{1}{q^{n-s}}$$

This result is quite strong, as it states that a random endomorphism known to vanish on q^k points in fact vanish on average on $q^k + 1$ points, even for large k.

Proof. By definition the expectation is:

$$E_n = \mathbb{E}\left[|\ker f| \;\middle|\; F \subseteq \ker f\right] = \sum_{k=s}^{n} \frac{D(n,k,s)}{q^{n(n-s)}} q^k$$

A combinatorial and/or elementary argument completely eluded us. We therefore use the method of "creative telescoping" to establish the result by induction on n. First, we notice that the announced results holds when $n = s$. Let us therefore assume $n > s$. We denote by $T(n,k,s)$ the hairy term under the sum. It is a q-hypergeometric term because if we set $X = q^n$ and $Y = q^k$, we see that the two following ratios are rational functions of X and Y:

$$\begin{aligned} \frac{T(n+1,k,s)}{T(n,k,s)} &= \frac{q^2X^2 - (q+q^{s+1})X + q^s}{q^2X^2 - qXY} \\ \frac{T(n,k+1,s)}{T(n,k,s)} &= q^{s+2} \frac{X+Y}{X\,(qY - q^s)\,(qY-1)} \end{aligned}$$

We thus used the q-analog of Zeilberger's algorithm [WZ92] (as implemented in Maple [MGH+05]), and it found the nice recurrence relation:

$$a \cdot T(n+1,k,s) - b \cdot T(n,k,s) = g(n,k+1,s) - g(n,k,s) \tag{$\star$}$$

where:

$$\begin{aligned} a &= q^{n+1} + q^{n+s+1} - q^{s+1} \\ b &= q^{n+1} + q^{n+1+s} - q^s \\ g(n,k,s) &= \frac{\left(q^k - q^s\right)\left(q^k - 1\right)\left(q^{n+s+1} - q^{n+s+2} - q^{k+s} + q^{n+k+1} + q^{n+k+s+1}\right)}{q^{2k}\left(q^{n+1} - q^k\right)} T(n,k,s) \end{aligned}$$

The point is that summing $(\star)$ over $k = s, \ldots, n-1$ yields:

$$a\left(E_{n+1} - T(n+1,n+1,s) - T(n+1,n,s)\right) - b\left(E_n - T(n,n,s)\right) = g(n,n,s) - g(n,s,s)$$

At this point, it is easy to find that $g(n,s,s) = 0$, and we check (using a computer algebra system!) that:

$$g(n,n,s) + a \cdot \left(T(n+1,n+1,s) + T(n+1,n,s)\right) + b \cdot T(n,n,s) = 0$$

Thus, we have established that:

$$\left(1 + q^s - \frac{1}{q^{n-s}}\right) E_{n+1} = \left(1 + q^s - \frac{1}{q^{n+1-s}}\right) E_n$$

Thus, if the result holds at rank n, then it also holds at rank $n+1$. □

Variance of the Cardinality of the Kernel. Using the same proof technique, we also obtain the variance of the cardinality of the kernel of a random endomorphism conditioned to vanish on a fixed subspace.

Lemma 10.21. *Let f be a uniformly random endomorphism of $(\mathbb{F}_q)^n$, vanishing on a subspace F of $(\mathbb{F}_q)^n$, with $\dim F = s$. Then the variance of the cardinality of its kernel is:*

$$q^s(q-1)\left(1 - \frac{q^s+1}{q^n} + \frac{q^s}{q^{2n}}\right)$$

Before giving the proof, we observe that coupled with lemma 10.20, this result enables the use of Chebychev's inequality: the probability that the kernel of a random endomorphism known to vanish on a subspace of dimension s is quite concentrated around q^s.

Proof. We will use the same proof technique. The variance is:

$$V_n = \underbrace{\sum_{k=s}^{n} \frac{D(n,k,s)}{q^{n(n-s)}} q^{2k}}_{U_n} - \left(q^s + 1 - \frac{1}{q^{n-s}}\right)^2$$

We will first demonstrate by induction on $n \geq s$ that:

$$U_n = q^{2s} + 1 + (1+q)\left(q^s - \frac{1}{q^{n-s}} - \frac{1}{q^{n-2s}}\right) + \frac{1}{q^{2n-1-2s}} \qquad (\clubsuit)$$

When $n = s$, we should have $U_n = q^{2n}$, and looking at $(\clubsuit)$ carefully reveals that our expression of U_n simplifies to this value. Let us therefore assume $n > s$, and let us again denote by $T(n,k,s)$ the hairy term under the sum. It is again a q-hypergeometric term, and running the q-analog of Zeilberger's algorithm yields:

$$a \cdot T(n+1,k,s) - b \cdot T(n,k,s) = g(n,k,s) - g(n,k+1,s) \qquad (\star)$$

where:

$$a = -q^{n+s+2} + q^{s+1+2n} + q^{1+2n} + q^{2s+2} - q^{2s+n+1} - q^{s+1+n} - q^{2s+2+n} + q^{2s+2n+1} + q^{s+2+2n}$$

$$b = -q^{1+2n} + q^{n+s} - q^{s+1+2n} + q^{s+1+n} - q^{2s} + q^{2s+n+1} + q^{2s+n} - q^{2s+2n+1} - q^{s+2+2n}$$

g is a complicated term with a singularity when $n+1 = k$. We again notice that $g(n,s,s) = 0$ and that:

$$a \cdot T(n+1,n+1,s) + a \cdot T(n+1,n,s) - b \cdot T(n,n,s) = g(n,n,s)$$

So that summing $(\star)$ over $k = s, \ldots, n-1$ and exploiting the previous equation yields:

$$a \cdot U_{n+1} = b \cdot U_n$$

By induction hypothesis, $(\clubsuit)$ holds at rank n. Plugging the expression of U_n into this recurrence relation and simplifying shows that $(\clubsuit)$ holds at rank $n+1$ — please use a computer algebra system if you really want to verify this.

Moving back to the expression of V_n, it is not difficult to verify that the result of the lemma holds. □

Partly Specified Endomorphisms. Being able to estimate the probability that a random endomorphism is invertible is useful, but there are situations where this result cannot be applied, for instance when the endomorphisms are known to be singular.

We therefore move on to count the number of endomorphisms whose image has certain properties, without requiring that the kernel is trivial. The natural question we would like to answer is: "what is the probability that a random endomorphism contains a given vector in its image?". This question is much less frequently addressed in the literature. The only related result we could find is:

Theorem 10.22 ([CR92], example 2.4)**.** *Let U and W be subspaces of $(\mathbb{F}_q)^n$ of respective dimensions ℓ and m. The number $X_n(\ell, m)$ of endomorphisms f of $(\mathbb{F}_q)^n$ such that for any $0 \neq u \in U, f(u) \notin W$ is:*

$$X_n(\ell, m) = \sum_{k \geq 0} (-1)^k q^{\binom{k}{2}} q^{n(n-k)} q^{mk} \binom{l}{k}_q$$

We initially hoped that with $U = (\mathbb{F}_q)^n$ and $W = \langle \mathbf{x} \rangle$, $X_n(n, 1)$ would count the number of endomorphism such that $f(\mathbf{y}) \neq \mathbf{x}$. Unfortunately, this is not what $X_n(\ell, m)$ counts. The problem is that $0 \in W$, since W is a vector space, and therefore $X_n(\ell, m)$ counts the number of endomorphism that are injective on U, but avoid W. This means that setting $U = (\mathbb{F}_q)^n$ is useless, because then we would be counting permutations of $(\mathbb{F}_q)^n$, and they cannot avoid W. We thus resorted to an *ad hoc* reasoning.

Lemma 10.23. *Let F and G be two subspaces of $(\mathbb{F}_q)^n$ of dimensions s and t respectively. The number of endomorphisms f of $(\mathbb{F}_q)^n$ such that:*

1. *The kernel of f has dimension k (with $s \leq k \leq n - t$),*
2. *F is contained in the kernel of f and*
3. *G is contained in the image of f,*

is:

$$\binom{n-s}{k-s}_q \binom{n-t}{k}_q \langle\!\langle n-k \rangle\!\rangle_q!$$

Proof. Let us fix the the kernel of f: let us call it U, and assume that it is of dimension k, with $s \leq k \leq n-t$. By lemma 10.18, there are $\binom{n-s}{k-s}_q$ possible kernels of dimension k.

Let V be a supplementary of U. The endomorphisms f are uniquely determined by their restriction f_V to V. Because G is contained in the image of f, then G is also contained in the image of f_V. The question then boils down to: "how many injective maps from V to $(\mathbb{F}_q)^n$ whose image contains G are there?". By lemma 10.18 there are $\binom{n-t}{n-k-t}_q$ possible images of f_V. However, f_V determines a particular basis of its image, so there are therefore $\binom{n-t}{n-k-t}_q \cdot \langle\!\langle n-k \rangle\!\rangle_q!$ possible choices of f_V. Multiplying by the number of possible kernels yields the result. □

Interestingly, the same arguments yields dual results, when the inclusions are reversed. A duality argument has eluded us, and we therefore give a direct counting argument.

Lemma 10.24. *Let F and G be two subspaces of $(\mathbb{F}_q)^n$ of dimensions $(n-s)$ and $(n-t)$ respectively. The number of endomorphisms f of $(\mathbb{F}_q)^n$ such that the kernel of f has dimension k and is contained in F, and the image of f is contained in G is:*

$$\binom{n-s}{k}_q \binom{n-t}{k-t}_q \langle\!\langle n-k \rangle\!\rangle_q!$$

Proof. Let us first count such endomorphisms with a k-dimensional kernel, with $t \leq k \leq n-s$. There are $\binom{n-s}{k}_q$ possible kernels of dimension k. For each one, there are $\binom{n-t}{n-k}_q \langle\!\langle n-k \rangle\!\rangle_q!$ possible injective maps from a supplementary of the kernel to G, and thus we obtain the result by multiplying these two numbers and doing some easy "q-binomial" manipulations. □

As an immediate corollary, we may remove the condition on the dimension of the kernel by summing over all admissible kernel dimensions.

Corollary 10.25. *Let F and G be two subspaces of $(\mathbb{F}_q)^n$ of dimensions s and t respectively.*

i) The number of endomorphisms f of $(\mathbb{F}_q)^n$ such that F is contained in the kernel of f and G is contained in the image of f is:

$$I(n, s, t) = \sum_{k=s}^{n-t} \binom{n-s}{k-s}_q \binom{n-t}{k}_q \langle\!\langle n-k \rangle\!\rangle_q! \tag{10.6}$$

ii) The number of endomorphisms f of $(\mathbb{F}_q)^n$ such that the kernel of f is contained in F and the image of f is contained in G is $I(n, n-t, n-s)$.

We conclude by establishing two useful recurrence relations for $I(n,0,1)$ and $I(n,1,1)$.

Lemma 10.26. *For any $n \in \mathbb{N}$, we have:*

i) $I(n,0,1) = (q^n-1)\cdot\left(q^{(n-1)^2} + \left(q^{n-1}-1\right)\cdot I(n-1,0,1)\right)$

ii) $I(n,1,1) = q^{n(n-1)} - \dfrac{1}{q^n-1}\cdot I(n,0,1)$.

Proof. *i)* We first show that $I(n,0,1)$ is governed by the relatively simple recurrence relation announced in the statement of the lemma. Let M be the matrix representation (in any basis) of an endomorphism f such that $e_1 \in \operatorname{Im} f$. The first row of M is then non-zero. There are thus q^n-1 possible first rows, and we may assume w.l.o.g. that that M has the following shape:

$$M = \left(\begin{array}{c|c} 1 & 0 \\ \hline C & D \\ & \end{array}\right)$$

Indeed, given an arbitrary first row, we may choose a particular basis of $(\mathbb{F}_q)^n$ such that the matrix representation of f in this basis has this particular shape. There are then two cases: either $C=0$, and in that case $f(e_1)=e_1$ so that D is not constrained, and there are $q^{(n-1)^2}$ possible D's. Or $C \neq 0$, and D must not be arbitrary. To deal with this more complicated case, we observe that there are $q^{n-1}-1$ possible choices for C, and we consider a given choice. In this setting, e_1 belongs to the image of f if and only if $-C$ belongs to the image of D. Thus we have established the recurrence relation:

$$I(n,0,1) = (q^n-1)\cdot\left(q^{(n-1)^2} + \left(q^{n-1}-1\right)\cdot I(n-1,0,1)\right) \qquad (\star)$$

ii) We now prove that the number of matrices whose kernel contains a given non-zero vector $\mathbf{x}$ and whose image *does not* contain a given non-zero vector $\mathbf{y}$ is $I(n,0,1)/(q^n-1)$, and this will establish the second part of the lemma.
We choose two bases of $(\mathbb{F}_q)^n$, $\mathcal{B}_1 = (\mathbf{x}, \mathbf{x}_2 \ldots, \mathbf{x}_n)$, and $\mathcal{B}_2 = (\mathbf{y}, \mathbf{y}_2 \ldots, \mathbf{y}_n)$, and we consider the matrix representation of an endomorphism meeting our requirements, with the input coordinates expressed in $\mathcal{B}_2$ and expressing the output coordinates in $\mathcal{B}_1$. Because it vanishes on $\mathbf{x}$, M necessarily has the following shape:

$$M = \left(\begin{array}{c|c} 0 & A \\ \hline 0 & B \\ & \end{array}\right)$$

In order to have $\mathbf{x} \notin \operatorname{Im} M$, we can either have $A=0$, or have $A \neq 0$ and $\ker B \subseteq \ker A$. If $A=0$, then B can be arbitrary and this yields $q^{(n-1)^2}$ choices. If A is non-zero there are $q^{n-1}-1$ possible choices for A, and the kernel of A kernel has dimension $n-2$. By corollary 10.25, the number of possible B such that $\ker B \subseteq \ker A$ is $I(n-1,0,1)$ Thus we have found:

$$q^{n(n-1)} - I(n,1,1) = q^{(n-1)^2} + \left(q^{n-1}-1\right)\cdot I(n-1,0,1)$$

Then, using $(\star)$, we find:

$$I(n,1,1) = q^{n(n-1)} - \frac{1}{q^n-1}\cdot I(n,0,1)$$

□

Asymptotic Probability that a Given Vector is Contained in the Image. The probability p_n that a given vector $\mathbf{x}$ of dimension s is contained in the image of a random linear map is simply $I(n,0,1)/q^{n^2}$. This probability converges to a finite limit different from 0 and 1 when $n \to +\infty$ (in fact, this probability is greater than the probability that a random matrix is invertible, obviously). We have not been able to demonstrate any definitive result, but we have a fairly good idea of what is going on.

Conjecture 10.1. If f is a random endomorphism of $(\mathbb{F}_q)^n$, then:

$$\lim_{n\to\infty} \mathbb{P}\left[e_1 \in \operatorname{Im} f\right] = \lim_{n\to\infty} \frac{I(n,0,1)}{q^{n^2}} = \sum_{k=0}^{\infty} (-1)^k q^{-k(k+1)/2}$$

We now give evidence that the conjecture is true. We check that $I(1,0,1) = q-1$, and we set:

$$J_n = \frac{I(n,0,1)}{q^{n^2} - q^{n^2-n}},$$

and we find out that lemma 10.26, item i) translates to:

$$J_{n+1} = q^{-n} + \left(1 - q^{-n}\right)^2 J_n$$

With $J_1 = 1$. We therefore define the sequence of polynomials:

$$P_1 = 1 \qquad P_{n+1} = X^n + (1 - X^n)^2 P_n$$

and it is obvious that $J_n = P_n(1/q)$. It is also quite obvious that $P_{n+1} - P_n$ is a multiple of X^n. This naturally extends by induction, so that $P_n - P_{n_0}$ is a multiple of X^{n_0} for any $n > n_0$. This means that $P_n(1/q)$ approximates the limit of $I(n,0,1)/q^{n^2}$ with precision q^{-n}. The key (empirical) observation is that P_n coincides with $\sum(-1)^k X^{k(k-1)/2}$ up to degree $n-1$ (*i.e.*, all the terms of P_n of degree less than n are correct). For instance:

$$\begin{aligned}
P_{10} &= 1 - X + X^3 - X^6 + \dots \\
P_{30} &= 1 - X + X^3 - X^6 + X^{10} - X^{15} + X^{21} - X^{28} + \dots \\
P_{60} &= 1 - X + X^3 - X^6 + X^{10} - X^{15} + X^{21} - X^{28} + X^{36} - X^{45} + X^{55} + \dots \\
P_{100} &= 1 - X + X^3 - X^6 + X^{10} - X^{15} + X^{21} - X^{28} + X^{36} - X^{45} + X^{55} + X^{78} - X^{91} \dots
\end{aligned}$$

We thus in fact conjecture that:

$$\lim_{n\to\infty} P_n(1/q) = \sum_{k=0}^{\infty} (1)^k q^{-k(k+1)/2}$$

The problem remains to find a proof that the first terms of P_n actually coincide with those of $\sum(1)^k q^{-k(k+1)/2}$. A proof of this assertion precisely remains to be found...

Sums of Singular Endomorphisms. We conclude this section by counting the number of pairs of endomorphisms vanishing on certain points, and whose sum is singular.

Lemma 10.27. *Let* $\mathbf{x}$ *be a non-zero vector of* $(\mathbb{F}_q)^n$*. The number of pairs of endomorphisms* f, g *of* $(\mathbb{F}_q)^n$ *such that* $f(\mathbf{x}) = 0$*,* $g \notin \mathrm{GL}_n(\mathbb{F}_q)$*, and* $f + g \notin \mathrm{GL}_n(\mathbb{F}_q)$ *is:*

$$q^{2n(n-1)} \cdot \left(1 + (q^n - 1) \cdot (1 - \lambda(n-1))^2\right)$$

Proof. Without loss of generality, we assume that $\mathbf{x} = e_1$. Let A and B be matrix representations of f and g. If $B \cdot \mathbf{x} = 0$, then the sum $A + B$ vanish on $\mathbf{x}$ and is thus singular. This already makes $q^{2n(n-1)}$ pairs of matrices. Now, if $B \cdot \mathbf{x} \neq 0$, then we fix the image of B on $\mathbf{x}$ ($q^n - 1$ choices), and we assume that it is e_1. This yields:

$$A = \left(\begin{array}{c|c} 0 & A_1 \\ \hline 0 & A_2 \end{array}\right) \qquad B = \left(\begin{array}{c|c} 1 & B_1 \\ \hline 0 & B_2 \end{array}\right)$$

Since B has a non-trivial kernel, then B_2 necessarily has a non-trivial kernel as well. For the same reason, the dimension of the kernel of the sum $A + B$ is in fact the dimension of the kernel of $A_2 + B_2$:

$$A + B = \left(\begin{array}{c|c} 1 & A_1 + B_1 \\ \hline 0 & A_2 + B_2 \end{array}\right)$$

Thus, A_1 and B_1 are arbitrary (q^{n-1} choices for each). The submatrix B_2 is a arbitrary but non-singular (and there are $q^{(n-1)^2} - \langle\!\langle n-1 \rangle\!\rangle_q!$ of those), and for each possible choice of B_2, the number of possibles A_2 is precisely the number of non-singular matrices of dimension $n-1$, because $M \mapsto M + B_2$ is a permutation of $\mathcal{M}_{n-1}(\mathbb{F}_q)$. Putting things together we find:

$$N_n = q^{2n(n-1)} \cdot \left(1 + (q^n - 1) \cdot (1 - \lambda(n-1))^2\right)$$

□

10.7.1 Random Quadratic Forms

We conclude by quickly investigating a few properties of random quadratic forms, *i.e.*, of quadratic forms whose coefficient have been uniformly chosen at random in $\mathbb{F}_q$. We start by discussing the probability that a random quadratic forms $\mathbf{f}$ is non-degenerate. In odd characteristic (resp. in characteristic two), the polar form is a random symmetric matrix (resp. skew-symmetric). We therefore use the two following results about random (skew-)symmetric matrices.

Lemma 10.28 ([Mac69], theorem 2)**.** *Let $N(n,r)$ denote the number of symmetric matrices of size $n \times n$ over $\mathbb{F}_q$ and of rank r.*

$$\begin{aligned} N(n,2s) &= \prod_{i=1}^{s} \frac{q^{2i}}{q^{2i}-1} \cdot \prod_{i=0}^{2s-1} \left(q^{n-i} - 1\right) \\ N(n,2s+1) &= \prod_{i=1}^{s} \frac{q^{2i}}{q^{2i}-1} \cdot \prod_{i=0}^{2s} \left(q^{n-i} - 1\right) \end{aligned}$$

Lemma 10.29 ([Mac69], theorem 3)**.** *Let $N_0(n,r)$ denote the number of skew-symmetric matrices of size $n \times n$ over $\mathbb{F}_{2^k}$ and of rank $2s$.*

$$\begin{aligned} N_0(n,2s) &= \prod_{i=1}^{s} \frac{q^{2i-2}}{q^{2i}-1} \cdot \prod_{i=0}^{2s-1} \left(q^{n-i} - 1\right) \\ N_0(n,2s+1) &= 0 \end{aligned}$$

In odd characteristic, lemma 10.28 is the tool we need, as non-degenerate quadratic forms are in one-to-one correspondance with non-singular symmetric matrices.

$$\mathbb{P}\left[\mathbf{f} \text{ non degenerate}\right] = q^{-\frac{n(n+1)}{2}} \cdot \prod_{i=1}^{n/2} \frac{q^{2i}}{q^{2i}-1} \cdot \prod_{i=1}^{n} \left(q^i - 1\right) = \frac{\lambda(q,n)}{\lambda\left(q^2, n/2\right)} \tag{10.7}$$

We find thanks to equation (10.4) that:

$$\lim_{n \to +\infty} \mathbb{P}\left[\mathbf{f} \text{ non degenerate}\right] = 1 - \frac{1}{q} - \frac{1}{q^3} + \frac{1}{q^4} + \mathcal{O}\left(\frac{1}{q^5}\right) \tag{10.8}$$

In characteristic two, if n is odd, then the quadratic forms are necessarily degenerate. On the other hand, if n is even, then lemma 10.29 tells us something nice:

$$\mathbb{P}\left[\mathbf{f} \text{ non degenerate}\right] = q^{-\frac{n(n-1)}{2}} \cdot \prod_{i=1}^{n/2} \frac{q^{2i-2}}{q^{2i}-1} \cdot \prod_{i=0}^{n-1} \left(q^{n-i} - 1\right) = \frac{\lambda(q,n)}{\lambda\left(q^2, n/2\right)}$$

We also determine, when q is even and n is odd, the probability that the polar form of a random quadratic form is of maximal rank (*i.e.*, $n-1$).

$$\begin{aligned} \mathbb{P}\left[\operatorname{rank} \mathbf{f} = n-1 \;\middle|\; n = 1 \bmod 2\right] &= q^{-\frac{n(n-1)}{2}} \cdot \prod_{i=1}^{(n-1)/2} \frac{q^{2i-2}}{q^{2i}-1} \cdot \prod_{i=2}^{n} \left(q^i - 1\right) \\ &= \frac{\lambda(q,n)}{\lambda(q^2,(n-1)/2)} \cdot \frac{1}{1-1/q} \end{aligned}$$

Chapter 11

Exhaustive Search for Boolean Equations

In this chapter we discuss the practical complexity of the problem of solving multivariate quadratic equations over $\mathbb{F}_2$. Over such a small field, Gröbner basis algorithms are usually less efficient than exhaustive search. We revisit this venerable technique and push it to its limits. This joint work with Hsieh-Chung Chen, Chen-Mou Cheng, Tung Chou, Ruben Niederhagen, Adi Shamir and Bo-Yin Yang resulted in a publication at CHES'2010 [BCC+10].

Solving a system of m polynomial equations in n variables over $\mathbb{F}_q$ is a natural mathematical problem that has been investigated by several research communities.

The hardness of this problem, and in particular the fact that its random instances seems as hard as they get, has been used by cryptographers to design many public-key cryptographic schemes [MI88, Pat96b, PGC98a, KPG99, PCG01a, PCG01b, WHL+05, DWY07, GMK08, BCD08, CBD+09], but also symmetric primitives, such as the stream cipher `QUAD` [BGP06] or the hash function `MQ-Hash` [BRP07].

Conversely, from a cryptanalytic point of view, it is sometimes possible to attack cryptosystems by writing down a system of equations and finding its solutions using off-the-shelf generic solvers (such as SAT solvers or Gröbner basis algorithms). The AES has been claimed to be broken this way [CP02], but the claim has been debunked in [CL05]. However, KeeLoq [CBW08] and some other weak block ciphers can be broken this way. Furthermore, an algebraic technique is claimed to yield a faster collision attack on SHA-1 reduced to 58 rounds [SKPI07], as well as on the Hamsi hash function [DS10].

Since the pioneering work of Buchberger [Buc65], Gröbner basis techniques have been the most prominent tool for solving systems of polynomial equations, especially after the emergence of faster algorithms such as $\mathbf{F_4}$ or $\mathbf{F_5}$ [Fau99, Fau02], which broke the first HFE challenge [Pat96b, FJ03]. Cryptographers independently rediscovered some of the ideas underlying several Gröbner basis algorithms under the form of the XL algorithm [CKPS00] and its "mutant" variants. They also introduced techniques to deal with special cases, particularly that of sparse systems [vRS06, Sem07, Rad07].

One of the striking features of Gröbner basis algorithms is that their complexity is only marginally affected by the size of the field. In her PhD thesis [Bar04], Bardet determined the number of field operations performed by a simplified version of the $\mathbf{F_5}$ algorithm to compute the Gröbner basis of an ideal spanned by n quadratic equations in n variables. She finds this number to be of order $2^{4.295n}$, independently of the size of the field. Therefore, as soon as $q \geq 20$, running a Gröbner basis algorithm should be asymptotically faster than exhaustive search. When $q = 2$, conversely, we expect Gröbner basis algorithm to be slower than exhaustive search. This is illustrated in a dazzling way by the following example: the implementation of $\mathbf{F_4}$ in `MAGMA-2.16` [BCP97], often cited as the best Gröbner-basis solver commercially available at the time of this writing [1], will completely use up all the available RAM (64 GB) to solve "just" 25 cubic equations in as many variables over $\mathbb{F}_2$, and eventually fail. Exhaustive search on the other hand takes 0.001 seconds to solve the same problem. This also illustrates the fact that the computation of a Gröbner basis is often a memory-bound process. Since memory is usually a scarce resource, sophisticated techniques can be inferior in practice to exhaustive search, which uses almost no memory (and is parallelizable at will).

This is not completely the end of the story though. In particular, we know from §10.6.1 that Gröbner-basis methods have an advantage on overdetermined systems (with man more equations than unknowns) and on systems with certain algebraic "weaknesses", such as HFE public keys.

All-in-all, small fields, in particular $\mathbb{F}_2$, are an unfavorable setting for Gröbner basis algorithms, and a favorable setting for brute force. It is also an interesting setup, because field operations in $\mathbb{F}_2$ are particularly simple to implement, making it a tempting ground to design new cryptographic primitives both in hardware and software. Multivariate public-key cryptography has often been advertized to be easy to implement on smartcards.

1. no competitive implementation of $\mathbf{F_5}$ is publicly available

Questions. Our purpose in this chapter is to investigate, theoretically and practically, how fast we can solve systems of multivariate polynomial equations over $\mathbb{F}_2$. We focus our study on *exhaustive search*, *i.e.*, algorithms that essentially enumerate $(\mathbb{F}_2)^n$ and check whether each individual assignment of the variable satisfies all the equations. The question we answer are: how fast can it get in practice? is it possible to improve on the folklore exhaustive search techniques? Can we get better performance using different hardware such as GPUs? Is it possible to solve *in practice*, with a modest budget, a system of 64 equations in 64 unknowns over $\mathbb{F}_2$? Less than 15 years ago, this was considered so difficult that it even underlied the security of a particular signature scheme [Pat96a].

To illustrate our point, let us consider the recent algebraic attack against Hamsi of Dinur and Shamir [DS10]. The attack critically relies on the ability to enumerate all the possible values of 62 degree-6 polynomials in 32 variables in less time than required to evaluate the Hamsi compression function 2^{32} times, which would require 2^{45} bit operations (because one compression function evaluation requires 10500 bit operations). If this polynomial evaluation were done (very) naively, it would require $62 \cdot \binom{32}{6} \cdot 2^{32} = 2^{57}$ bit operations, and there would be no attack. Using an optimized Moebius transform (see §11.2), Dinur and Shamir bring this cost down to $62 \cdot 7 \cdot 2^{32} = 2^{40.7}$ bit operations, and their procedure is faster than exhaustive search on the compression function. This illustrates that there are several, more-or-less optimal ways to perform the exhaustive search on boolean polynomial, and that the speed of exhaustive search *does* matter.

Contributions. Our contribution is twofold. On the theoretical side, we present a new type of exhaustive search algorithm which is both asymptotically and practically faster than existing techniques (but competes, probably in a favorable way, with [DS10]). In particular, we show that finding *all* the zeroes of a single degree-d polynomial in n variables over $\mathbb{F}_2$ requires just $d \cdot 2^n$ bit operations. We then extend this technique and show how to find all the common zeroes of m random quadratic polynomials in $\log_2 n \cdot 2^{n+2}$ bit operations, which is only slightly higher. Surprisingly, this complexity is *independent of the number of equations* m.

On the practical side, we have implemented our new algorithms on x86 CPUs and on NVIDIA GPUs. Our CPU implementation is fairly optimized, and is capable of testing a bit more than two $(\mathbb{F}_2)^n$-vectors *per CPU cycle*. Yet our GPU implementation running on one single NVIDIA GTX 295 graphics card runs up to 9 times faster than the CPU implementation using all four cores of an Intel Core i7 at 3 GHz, one of the fastest CPUs available at the time of this study (summer 2010). We can solve 48+ quadratic equations in 48 binary variables using just an NVIDIA GTX 295 graphics card in 21 minutes. This device is presenlty available for about $500. It would be 36 minutes for cubic equations and two hours for quartics.

To evaluate the practical relevance of this level of performance, we determine the budget necessary to solve several size of instances on the cloud (more precisely, on the Amazon Elastic Compute Cloud using GPU instances). At the time of this writing, each "GPU instance" (on the EC2 Cloud) is equipped with 8 Xeon core at 2.93Ghz and two NVIDIA Tesla M2050 GPUs (which are at least as fast as the NVIDIA GTX 295 that we tested). Such an instance is thus capable of testing about 2^{39} candidate per second, and is available at the rate of $0.74 per hour. It follows that the 64-bit signature challenge [Pat96a] can thus be broken in 9320 computing hours, with a budget of $7000. This computation may be completed in a single month using 12 instances in parallel. This show with a concrete example that (relatively simple) computations requiring 2^{64} operations could be carried out in practice with readily available hardware and a modest budget.

More interestingly, let us consider a quadratic system of 80 equations in 80 variables over $\mathbb{F}_2$, supposedly offering 80 bits of security, which is the standard security level as of 2011. Solving this system requires 610 millions computing hours, which is certainly large, and which would cost about 450 millions of dollars. It would nevertheless require a bit more than 20 years to a present actual supercomputer (the Tiahne-1A) to solve the problem...

In addition, our results show that the performance/price ratio of the GPU implementation running on an NVIDIA GTX 295 board is three times better than that of the CPU implementation running on the most price-efficient CPU (an AMD Phenom). We also highlight the fact that GPUs can been used successfully by the cryptographic community to obtain very efficient implementations of combinatorial algorithms or cryptanalytic attacks, in addition to the more numeric-flavored cryptanalysis algorithm demonstrated by the implementation of the ECM factorization algorithm on GPUs [BCC+09].

Lastly, our implementations are (by some orders of magnitude) the fastest existing generic solvers for (low-degree, dense) systems of polynomial equations over $\mathbb{F}_2$.

11.1 Generalities

In this chapter, we will mostly be working over the finite vector space $(\mathbb{F}_2)^n$. The canonical basis is denoted by $(e_0, \ldots, e_{n-1})$. We use $\oplus$ to denote addition in $(\mathbb{F}_2)^n$, and $+$ to denote integer addition. We use $i \ll k$ (resp. $i \gg k$) to denote binary left-shift (resp. right shift) of the integer i by k bits.

i	$\textsc{GrayCode}(i)$
0	000
1	001
2	011
3	010
4	110
5	111
6	110
7	100
8	101

Table 11.1: First 8 numbers in our "usual" Gray Code.

11.1.1 Gray Code

Gray Codes play a crucial role in this chapter. Recall that a Gray Code is binary numeral system where two successive values differ in only one bit (or equivalently, a permutation of $(\mathbb{F}_2)^n$ such that two consecutive vectors differ in only one bit). There are many Gray Codes with various properties, but we will use the most "standard" one

Definition 11.1. $\textsc{GrayCode}(i) = i \oplus (i \gg 1)$.

Table 11.1 shows the first 8 numbers of the Gray Code. It requires some thinking to get convinced that definition 11.1 actually yields a Gray Code. However, it is a fairly standard construction therefore we will not elaborate further.

Let us denote by $b_k(i)$ the index of the k-th lowest-significant bit of i set to 1, or -1 if the hamming weight of i is less than k. For example, $b_k(0) = -1$, $b_1(1) = 0, b_1(2) = 1$ and $b_2(3) = 1$. The following property is also standard (this could be an alternate definition).

Proposition 11.1. *For $i \in \mathbb{N}$,* $\textsc{GrayCode}(i+1) = \textsc{GrayCode}(i) \oplus e_{b_1(i+1)}$.

Gray Codes have other useful properties. For instance, it easily follows from the definition that $\textsc{GrayCode}(i \oplus j) = \textsc{GrayCode}(i) \oplus \textsc{GrayCode}(j)$. The following result expresses how the Gray Code behaves with respect to left-shifting.

Lemma 11.2. *For $j \in \mathbb{N}$:*

$$\textsc{GrayCode}\left(2^k + j \cdot 2^{k+1}\right) = \begin{cases} \textsc{GrayCode}\left(2^k\right) \oplus (\textsc{GrayCode}(j) \ll (k+1)) & \text{if } j \text{ is even} \\ \textsc{GrayCode}\left(2^k\right) \oplus (\textsc{GrayCode}(j) \ll (k+1)) \oplus e_k & \text{if } j \text{ is odd.} \end{cases}$$

Proof. It should be clear that $2^k + j \cdot 2^{k+1}$ and $2^k \oplus j \cdot 2^{k+1}$ in fact denote the same number. Also, $\textsc{GrayCode}$ is a linear function on $(\mathbb{F}_2)^n$. Thus it remains to establish that $\textsc{GrayCode}\left(j \cdot 2^{k+1}\right) = \textsc{GrayCode}(j) \ll k+1$ (resp. $e_k \oplus (\textsc{GrayCode}(j) \ll k+1)$) when j is even (resp. odd). Again, $j \cdot 2^{k+1} = j \ll (k+1)$, and by definition we have:

$$\textsc{GrayCode}\left(j \cdot 2^{k+1}\right) = \textsc{GrayCode}(j \ll (k+1)) = (j \ll (k+1)) \oplus ((j \ll (k+1)) \gg 1)$$

Now, we have :

$$(j \ll k+1) \gg 1 = \begin{cases} (j \gg 1) \ll k+1 & \text{when } j \text{ is even} \\ ((j \gg 1) \ll k+1) \oplus e_k & \text{when } j \text{ is odd} \end{cases}$$

and the result follows. □

11.1.2 A Framework for Enumeration Algorithms

We are interested in *enumeration algorithms*, *i.e.*, algorithms that evaluate a polynomial f over all the points of $(\mathbb{F}_2)^n$ to find its zeroes. Such an enumeration algorithm is composed of two functions: $\textsc{Init}$ and $\textsc{Next}$. $\textsc{Init}(f, x_0, k_0)$ returns a *State* containing all the information the enumeration algorithm needs for the remaining operations. The resulting State is configured for the evaluation of f over $x_0 \oplus (\textsc{GrayCode}(i) \ll k_0)$, for increasing values of i. $\textsc{Next}(State)$ advances to the next value and updates *State*. Three values can be directly read from the state: $State.\mathbf{x}, State.\mathbf{y}$ and $State.i$. These are linked at all times by the following three invariants:

i) $State.\mathbf{y} = f(State.\mathbf{x})$

ii) $State.\mathbf{x} = x_0 \oplus (\textsc{GrayCode}(State.i) \ll k_0)$.

iii) $\textsc{Next}(State).i = State.i + 1$.

Algorithm 11.1 Main loop common to all enumeration algorithms.

```
1: procedure Zeroes(f)
2:     State ← Init(f, 0, 0)
3:     for i from 0 to 2^n − 1 do
4:         if State.y = 0 then State.x is a zero of f
5:         Next(State)
6:     end for
7: end procedure
```

Finding all the zeroes of f is then achieved with Algorithm 11.1. In this chapter, most functions are described in an "object-oriented" way: Init and Next explicitly handle States, and all the (global) variables occuring inside a function are in fact part of the state. This allows to unambiguously run several copies of the enumeration algorithm on (say) different polynomials, and disambiguates some intricate recursive situations. We could have made the dependence on the state explicit:

```
1: function Some-Function(State)
2:     State.x ← 2State.x + √State.y − 1
3: end function
```

But for the sake of less cumbersome notations, we make this dependence implicit:

```
1: function Some-Function(State)
2:     x ← 2x + √y − 1
3: end function
```

11.2 Known Techniques for Quadratic Polynomials

We briefly discuss the enumeration techniques we know of.

11.2.1 Naive Evaluation.

The simplest way to implement an enumeration algorithm is to evaluate the polynomial f from scratch at each point of $(\mathbb{F}_2)^n$. If f is of degree d, this requires $(d-1)$ AND per monomial, and nearly one XOR per monomial. Since the evaluation takes place many times for the same f with different values of the variables, we will usually assume that the polynomial can be *hard-coded*, and that multiplication of a monomial by its coefficient comes for free. Each call to Next would then require at most $d \cdot \binom{n}{d}$ bit operations, $1/d$ of which being XORs and the rest being ANDs (not counting the cost of enumerating $(\mathbb{F}_2)^n$, *i.e.*, incrementing a counter). This can be improved a bit, using what is essentially a multivariate Hörner evaluation technique. If f is quadratic, it can be written:

$$f(\mathbf{x}) = c \oplus \sum_{i=0}^{n-1} \mathbf{x}_i \cdot \left(c_j \oplus \sum_{j=i+1}^{n-1} a_{ij} \cdot \mathbf{x}_j \right) \tag{11.1}$$

If f is cubic, it can be written:

$$f(\mathbf{x}) = c \oplus \sum_{i=0}^{n-1} \mathbf{x}_i \cdot \left(c_j \oplus \sum_{j=i+1}^{n-1} \mathbf{x}_j \cdot \left(c_{ij} \oplus \sum_{k=j+1}^{n-1} a_{ijk} \cdot \mathbf{x}_k \right) \right)$$

And so on and so forth. The required numbers of operations in this representation is given by:

$$N_{AND} = \sum_{k=1}^{d-1} \binom{n}{k} \qquad N_{XOR} = \sum_{k=1}^{d} \binom{n}{k}$$

This naive method is not without its advantages, chiefly (a) insensitivity to the order in which the points of $(\mathbb{F}_2)^n$ are enumerated, and (b) we can bit-slice and get a speed up of nearly ω, where ω is the maximum width of the CPU logical instructions.

11.2.2 The Folklore Differential Technique.

It was pointed out in §10.2.3 that once $f(\mathbf{x})$ is known, computing $f(\mathbf{x} \oplus e_i)$ amounts to compute $\frac{\partial f}{\partial i}(\mathbf{x})$, and then perform an addition. If f is quadratic, and in this case only, this derivative happens to be an affine function which can be efficiently evaluated by computing a vector-vector product and a few scalar additions. This strongly suggests to evaluate f on $(\mathbb{F}_2)^n$ using a Gray Code, and leads to Algorithm. 11.2.

Algorithm 11.2 The Folklore differential enumeration algorithm.

function INIT($f, k_0, \mathbf{x}_0$)
  $i \leftarrow 0$
  $\mathbf{x} \leftarrow \mathbf{x}_0$
  $\mathbf{y} \leftarrow f(\mathbf{x}_0)$
  for all $0 \leq k \leq n-1$ **do**
    $D_k \leftarrow \mathrm{D}_{e_k} f$
    $c_k \leftarrow f(0) \oplus f(e_k)$
  end for
  return State
end function

function NEXT($State$)
  $i \leftarrow i + 1$
  let $k = b_1(i)$ **in**
  let $\mathbf{z} = \mathrm{DOTPRODUCT}(D_k, \mathbf{x}) \oplus c_k$ **in**
  $\mathbf{y} \leftarrow \mathbf{y} \oplus \mathbf{z}$
  $\mathbf{x} \leftarrow \mathbf{x} \oplus e_{k+k_0}$
end function

We believe this technique to be folklore (it was told to us by J.-C. Faugère), and in any case it appears more or less explicitly in the existing literature [BBG07]. Each call to NEXT requires n ANDs, as well as $n+2$ XORs, which makes a total bit operation count of $2(n+1)$. This is about $n/4$ times less than the naive method applied to a quadratic f.

11.2.3 A Fast Fourrier Transform for Boolean Functions : the Moebius Transform

Any boolean function $f : (\mathbb{F}_2)^n \to \mathbb{F}_2$ can be represented by a multivariate polynomial in n variables with coefficients in $\mathbb{F}_2$. This particular representation of f is usually called the *algebraic normal form* of f. Other representations of f would be possible, for instance by its truth table. In algebraic normal form, f can be written:

$$f(\mathbf{x}_1, \ldots, \mathbf{x}_n) = \sum_{\mathbf{a} \in (\mathbb{F}_2)^n} g(\mathbf{a}_1, \ldots, \mathbf{a}_n) \prod_{i=1}^{n} \mathbf{x}_i^{\mathbf{a}_i}$$

The function that given the truth table of f returns the truth table of g (or equivalently the coefficients of the ANF of f) is called the *Moebius transform* (observe the analogy with the Discrete Fourier Transform). Unlike the usual DFT on polynomials, the Moebius transform is involutive (*i.e.*, it is its own inverse), and just like the DFT it can be computed by a "fast" algorithm with $\mathcal{O}(n \cdot 2^n)$ XOR operations, given in [Jou09b, § 9.2]. Algorithm 11.3 shows the pseudo-code. Note that this is not an enumeration algorithm, but can be used for this purpose (in the g-to-f direction). Its total (time) complexity is $n \cdot 2^n$ XOR operations on one bit, plus $n \cdot 2^n$ integers additions, and its total space complexity is 2^n bits. This algorithm can be implemented with decent efficiently (see [Jou09b]), but we suspect that its performances are affected by the available memory bandwidth. Also, it cannot take advantage of the possibility that the degree of f is small (*i.e.*, that g is mostly zero).

Dinur and Shamir subsequently proposed in [DS10] an improved version of the Moebius Transform, which is degree-aware and offer better performances on quadratic polynomials.

11.3 A Faster Enumeration Algorithm

In this section, we present a new algorithm which is both asymptotically and practically faster than other known exhaustive search techniques in evaluating a polynomial of any degree on all the points of $(\mathbb{F}_2)^n$.

Algorithm 11.3 Moebius Transform algorithm.

Require: Input truth table of the boolean function f, with 2^n entries
Ensure: Overwrites the input with Moebius transform of f.

```
1: for i from 0 to n − 1 do
2:     let Sz = 2^i and Pos = 0 in
3:     while Pos < 2^n do
4:         for j from 0 to Sz − 1 do
5:             f[Pos + Sz + i] ← f[Pos + Sz + j] ⊕ f[Pos + j]
6:         end for
7:         Pos ← Pos + 2 · Sz
8:     end while
9: end for
```

11.3.1 Faster Enumeration in Degree Two

In Algorithm 11.2, the dominating part is the dot product computed in Next (on line 14). It would be great if it were possible to exploit the fact that $\mathbf{x}$ is only slightly changed between two calls to Next. The problem is that k (defined on line 13) is never the same in two consecutive iterations. Now assume we modify the function, by storing the last value of $\mathbf{z}$ computed with each value of k:

```
1: function Next(State)
2:     i ← i + 1
3:     let k = b_1(i) in
4:     z[k] ← DotProduct(D_k, x) ⊕ c_k
5:     x[k] ← x
6:     y ← y ⊕ z[k]
7:     x ← x ⊕ e_k
8: end function
```

In the modified function, on line 4, the *previous* value of $\mathbf{z}[k]$, when it exists, is still available, and this value is the dot product of D_k with $\mathbf{x}[k]$ (which is the previous value of $\mathbf{x}$ for the same value of k). Thus, the new value of $\mathbf{z}[k]$ is going to be $\mathbf{z}[k] \oplus D_k \cdot (\mathbf{x} \oplus \mathbf{x}[k])$. The key observation is proposition 11.3 below, as its consequence is that the computation of the scalar product can be done in constant time, with two ANDs and one XOR. We will need to discuss the values of some variables inside functions, and in order to avoid any ambiguity, we denote by $x^\top$ the value of x at the begining of the function (before any instruction is exectued).

Proposition 11.3. *At the beginning of the function,* $\mathbf{x}^\top \oplus \mathbf{x}[k]^\top$ *has hamming weight two.*

Proof. $\mathbf{x}[k_0]^\top$ is only accessed and modified when $b_1\left(i^\top + 1\right) = k_0$, for any given k_0. The integers u such that $b_1(u) = k_0$ are precisely the integers written $u = 2^{k_0} + j \cdot 2^{k_0+1}$, for $j \geq 0$. Then, if we consider the values of the variables at the beginning of the function, by invariant ii, we have for some j:

$$\begin{aligned} \mathbf{x}^\top &= \textsc{GrayCode}\left(2^k + (j+1) \cdot 2^{k+1}\right) \\ \mathbf{x}[k]^\top &= \textsc{GrayCode}\left(2^k + j \cdot 2^{k+1}\right) \end{aligned}$$

Thus, it follows from lemma 11.2 that just before line 1 is executed, we have:

$$\begin{aligned} \mathbf{x}^\top \oplus \mathbf{x}[k]^\top &= e_k \oplus (\textsc{GrayCode}(j) \ll (k+1)) \oplus (\textsc{GrayCode}(j+1) \ll (k+1)) \\ &= e_k \oplus ((\textsc{GrayCode}\,(j) \oplus \textsc{GrayCode}\,(j+1)) \ll (k+1)) \end{aligned}$$

and by lemma 11.1,

$$\mathbf{x}^\top \oplus \mathbf{x}[k]^\top = e_k \oplus e_{k+1+b_1(j+1)} \tag{11.2}$$

□

By looking closely at the proof of proposition 11.3, we can write an *optimized differential algorithm.* However, before that, a few details still need to be addressed.

- The first time that $b_1(i) = k$, then $\mathbf{z}[k]$ is not defined. In this case, we in fact know that $i = 2^k$. Therefore, special care must be taken to initialize $\mathbf{z}[k]$ when $b_2(i) = -1$, which is equivalent to saying that the hamming weight of i is less than two. In that case, by invariant ii, we have:

$$\mathbf{x} = \begin{cases} e_0 & \text{if } i = 1 \\ e_k \oplus e_{k-1} & \text{if } i = 2^k \text{ and } k > 0 \end{cases}$$

– Also note that with the notation $k_1 = b_1(i)$ and $k_2 = b_2(i)$, then if $b_2(i) \neq -1$, equation (11.2) becomes:

$$\mathbf{x} \oplus \mathbf{x}[k] = e_{k_1} \oplus e_{k_2}$$

And thus,

$$\textsc{DotProduct}(D_{k_1}, \mathbf{x} \oplus \mathbf{x}[k_1]) = D_{k_1}[k_1] \oplus D_{k_1}[k_2]$$

This last formula can be further simplified by observing that $D_{k_1}[k_1] = 0$.

All these considerations lead to Algorithm 11.4. Note that the conditional statement could be removed by unrolling the loop carefully. The critical part of the algorithm is therefore an extremely reduced section of the code, that performs two XORs, increment a counter, and evaluate b_1 as well as b_2. The cost of maintaining i, k_1 and k_2 can again be reduced greatly by unrolling the loop.

Algorithm 11.4 An optimized differential enumeration algorithm for quadratic polynomials.

```
1: function Init(f, k_0, x_0)
2:     i ← 0
3:     x ← x_0
4:     y ← f(x_0)
5:     for all 0 ≤ k ≤ n − 1 do
6:         D_k ← D_{e_k} f
7:         c_k ← f(0) ⊕ f(e_k)
8:     end for
9:     z[0] ← c_0
10:    for all 1 ≤ k ≤ n − 1 do
11:        z[k] ← D_k[k − 1] ⊕ c_k
12:    end for
13: end function
14: function Next(State)
15:    i ← i + 1
16:    k_1 = b_1(i)
17:    k_2 = b_2(i)
18:    if k_2 ≠ −1 then z[k_1] ← z[k_1] ⊕ D_{k_1}[k_2]
19:    y ← y ⊕ z[k_1]
20:    x ← x ⊕ e_{k_0+k_1}
21: end function
```

11.3.2 Recursive Generalization to Any Degree

It is possible to generalize Algorithm 11.4 so that it handles polynomials of any degree. The core idea is that in this algorithm, a given derivative is evaluated on the consecutive points of something that looks very much like a Gray code. This suggest using the technique recursively.

To make this thing explicit, we introduce a new State for each of the derivatives of f used in the enumeration of f. Instead of storing $\mathbf{x}[k]$ and $\mathbf{z}[k]$, we will access $Derivative[k].\mathbf{y}$ and $Derivative[k].\mathbf{y}$. Also, $Derivative[k].i$ will count the number of times $b_1(k)$ happened. We now reformulate our new algorithm in this framework. However, now, the x_0 and k_0 parameters appearing in invariant ii will play a more important role. Algorithm 11.5 works for polynomials f of any degre. The two functions $\textsc{Init}^{[d]}$ and $\textsc{Next}^{[d]}$ are designed to handle degree-d polynomials.

11.3.3 Correctness

At first glance, it may not seem trivial that the combination of algorithms 11.1 and 11.5 results in a method for finding all the zeroes of f. We thus prove by induction on d (the degree of f) that the two functions $\textsc{Init}^{[d]}$ and $\textsc{Next}^{[d]}$ of Algorithm 11.5 maintain and preserve the three invariants of enumeration algorithms, defined in §11.1.2. The base case is when f is a constant polynomial (*i.e.*, $d = 0$). we hope that the reader will be convinced that the "base case" of the algorithm correctly enumerates the values of a constant polynomial...

In the recursive case where f is not constant, it is not difficult to check that the three invariants are enforced at the end of $\textsc{Init}^{[d]}$. Let us now assume that f has degree $d \geq 1$. Let us assume that we are in the middle of the main loop, and that the invariants defining our enumeration algorithm hold at the beginning of $\textsc{Next}^{[d]}$. Our objective is to show that they still hold at the end, and that the state has been updated

Algorithm 11.5 The recursive differential enumeration algorithm for all degrees.

1: **function** $\textsc{Init}^{[d]}(f, k_0, x_0)$
2: $\quad i \leftarrow 0$
3: $\quad \mathbf{x} \leftarrow x_0$
4: $\quad \mathbf{y} \leftarrow f(x_0)$
5: $\quad$ **if** $d > 0$ **then**
6: $\quad\quad Derivative[0] \leftarrow \textsc{Init}^{[d-1]}\left(\dfrac{\partial f}{\partial k_0}, k_0 + 1, x_0\right)$
7: $\quad\quad$ **for** k **from** 1 **to** $n - k_0 - 1$ **do**
8: $\quad\quad\quad Derivative[k] \leftarrow \textsc{Init}^{[d-1]}\left(\dfrac{\partial f}{\partial k + k_0}, k + k_0 + 1, x_0 \oplus e_{k_0+k-1}\right)$
9: $\quad$ **end if**
10: **end function**
11: **function** $\textsc{Next}^{[d]}(State)$
12: $\quad i \leftarrow i + 1$
13: $\quad$ **let** $k = b_1(i)$ **in**
14: $\quad \mathbf{x} \leftarrow \mathbf{x} \oplus e_{k+k_0}$
15: $\quad$ **if** $d > 0$ **then**
16: $\quad\quad \mathbf{y} \leftarrow \mathbf{y} \oplus Derivative[k].\mathbf{y}$
17: $\quad\quad \textsc{Next}^{[d-1]}\left(Derivative\left[k\right]\right)$
18: $\quad$ **end if**
19: **end function**

correctly. Let us then focus on the $\textsc{Next}^{[d]}$ part of algorithm 11.5. Invariant *iii* is easily seen to be enforced by line 12, while invariant *ii* follows from line 14, and from lemma 11.1. The non-trivial part is to show that invariant *i* holds. The three following lemma are devoted to this task. We will always denote by $x^\top$ (resp. $x^\perp$) the value that the x variable had at the begining of the execution of the function (resp. at the end).

Lemma 11.4. *After k is updated on line 13 of $\textsc{Next}^{[d]}$, we have:*

$$i^\top + 1 = 2^k + Derivative[k].i^\top \times 2^{k+1}.$$

Proof. It is not difficult to see that the ℓ-th value of j such that $b_1(j) = k$ is $2^k + \ell \times 2^{k+1}$. The statement of the lemma is equivalent to saying that $Derivative[k].i^\top$ counts the number of time where $b_1(i) = k$ happened since the beginning of the main loop (not counting $i^\top + 1$). This simply follows from the fact that $Derivative[k].i^\top$ counts the number of times $\textsc{Next}^{[d-1]}\left(Derivative[k]\right)$ has been called. □

Lemma 11.5. *Let $\Pi : (\mathbb{F}_2)^n \to (\mathbb{F}_2)^n$ denote the projection that sets the $(k+k_0)$-th coordinate to zero. After k is updated on line 13, and before $\mathbf{x}$ is updated on line 14 of $\textsc{Next}^{[d]}$, we have:*

$$\Pi\left(\mathbf{x}^\top \oplus Derivative[k].\mathbf{x}^\top\right) = 0$$

Proof. By assuming that invariant *ii* holds for the current state at the entry of $\textsc{Next}^{[d]}$, we have:

$$\mathbf{x}^\top = x_0 \oplus \left(\textsc{GrayCode}\left(i^\top\right) \ll k_0\right).$$

Because after line 13, k is set to $b_1\left(i^\top + 1\right)$, it follows from lemma 11.1 that:

$$\begin{aligned}\mathbf{x}^\top &= x_0 \oplus \left(\left(\textsc{GrayCode}\left(i^\top + 1\right) \oplus e_k\right) \ll k_0\right) \\ &= x_0 \oplus e_{k+k_0} \oplus \left(\textsc{GrayCode}\left(i^\top + 1\right) \ll k_0\right)\end{aligned}$$

Then, because lemma 11.4 grants us: $i^\top + 1 = 2^k + Derivative[k].i^\top \times 2^{k+1}$, this becomes:

$$\mathbf{x}^\top = x_0 \oplus e_{k+k_0} \oplus \left(\textsc{GrayCode}\left(2^k + Derivative[k].i^\top \times 2^{k+1}\right) \ll k_0\right)$$

Applying lemma 11.2 gives:

$$\Pi\left(\mathbf{x}^\top\right) = \Pi\left(x_0 \oplus \left(\textsc{GrayCode}\left(2^k\right) \ll k_0\right) \oplus \left(\textsc{GrayCode}(Derivative[k].i^\top) \ll (k_0 + k + 1)\right)\right)$$

We now distinguish two cases.

– Either $k > 0$, and since $\frac{\partial f}{\partial k+k_0}$ is of strictly smaller degree than f, then by induction hypothesis on $Derivative[k]$, invariant ii grants:

$$Derivative[k].\mathbf{x}^\top = x_0 \oplus \left(\textsc{GrayCode}\left(Derivative[k].i^\top\right) \ll k + k_0 + 1\right) \oplus e_{k+k_0-1}$$

And thus:

$$\begin{aligned}\Pi\left(\mathbf{x}^\top \oplus Derivative[k].\mathbf{x}^\top\right) &= \Pi\left(\left(\textsc{GrayCode}\left(2^k\right) \ll k_0\right) \oplus e_{k+k_0-1}\right)\\ &= \Pi\left(\left(e_{k+k_0} \oplus e_{k+k_0-1}\right) \oplus e_{k+k_0-1}\right)\\ &= 0\end{aligned}$$

– Or $k = 0$, and by induction hypothesis invariant ii yields:

$$Derivative[0].\mathbf{x}^\top = x_0 \oplus \left(\textsc{GrayCode}\left(Derivative[0].i^\top\right) \ll k_0 + 1\right)$$

Next,

$$\begin{aligned}\Pi\left(\mathbf{x}^\top \oplus Derivative[0].\mathbf{x}^\top\right) &= \Pi\left(\textsc{GrayCode}\left(2^k\right) \ll k_0\right)\\ &= \Pi\left(e_{k_0}\right)\\ &= 0\end{aligned}$$

□

Lemma 11.6. *We have $\mathbf{y}^\perp = f(\mathbf{x}^\perp)$. In other terms, invariant i is preserved.*

Proof. By induction hypothesis on $Derivative[k]$, invariant i :

$$Derivative[k].\mathbf{y}^\top = \frac{\partial f}{\partial k + k_0}(Derivative[k].\mathbf{x}^\top)$$

However, because we are in characteristic two, we have: $\frac{\partial f}{\partial k+k_0} = \frac{\partial f}{\partial k+k_0} \circ \Pi$, and lemma 11.5 in fact grants us:

$$Derivative[k].\mathbf{y}^\top = \frac{\partial f}{\partial k + k_0}\left(\mathbf{x}^\top\right)$$

So, this yields (using lemma 11.1):

$$\begin{aligned}\mathbf{y}^\perp &= \mathbf{y}^\top \oplus \frac{\partial f}{\partial k + k_0}\left(\mathbf{x}^\top\right)\\ &= f\left(\mathbf{x}^\top\right) \oplus f\left(\mathbf{x}^\top\right) \oplus f\left(\mathbf{x}^\top + e_{k+k_0}\right)\\ &= f\left(\mathbf{x}^\perp\right)\end{aligned}$$

□

11.3.4 Time and Space Complexity Considerations

It should be clear from the description of $\textsc{Next}^{[d]}$ that it has complexity $\mathcal{O}(d)$. Therefore, the complexity of enumerating all the values of f on $(\mathbb{F}_2)^n$ can be done with complexity $\mathcal{O}(d \cdot 2^n)$. What is the space requirement of the algorithm? The answer to this question is twofold: there is an *internal state* that gets modified by the algorithm, and that correspond to the $\mathbf{y}$ field of all the non-constant derivatives. There is also an array of *constants*, which is only read from the memory, and that correspond to the $\mathbf{y}$ field of degree-d derivatives.

Init stores one bit per degree-d derivative $\partial f/\partial i_1 \partial i_2 \dots \partial i_d$, with $1 \leq i_1 < i_2 < \cdots < i_d \leq n$. The number of such tuples $(i_1, i_2, \dots, i_d)$ is known to be $\binom{n}{d-1}$. This yields the following result:

Proposition 11.7. *The algorithm allocates $\sum_{i=0}^{d-1} \binom{n}{i}$ bits of internal state and $\binom{n}{d}$ bits of constants*

Numerical Results. For instance, to enumerate 64 quadratic polynomials in 64 variables simultaneously, 520 bytes of internal state and 16128 bytes of constants are needed. For the Hamsi algebraic attack, where 62 degree-6 polynomials in 32 variables are enumerated, 1.8 Megabytes of internal state and 6.7 Megabytes of constants are required.

11.3.5 An iterative Version

Algorithm 11.5 is (relatively) easy to prove correct, but does not lend itself well to an efficient implementation. We therefore move on to writing an iterative version of Algorithm 11.5. This iterative version allows more optimization, such as the removal of extra useless work, and is generally easier to think about with performance in mind.

Algorithm 11.6 An equivalent version of Next.

```
1: function Next_2(State)
2:     i ← i + 1
3:     let k = b_1(i) in
4:     if i ≠ 2^k then Next_2 (Derivative [k])
5:     x ← x ⊕ e_{k+k_0}
6:     y ← y ⊕ Derivative[k].y
7: end function
```

But first, we introducce the function Next_2 (Algorithm 11.6). It does exactly the same thing as Next, but in a slightly different way. Instead of calling Next at the end, it calls it at the beginning, except the first time a given value of k is reached (to avoid calling it an extra time at the begining). We can therefore work on Next_2 from now on. A first remark is that maintaining $\mathbf{x}$ is required by the invariants, but is otherwise useless for the actual computation. A first step is to completely remove $\mathbf{x}$ from the algorithm. Less obviously, we can also avoid maintaining i. To see that, we first need an equivalent of lemma 11.4 adapted to Next_2, the proof of which is left to the reader.

Lemma 11.8. *After k is updated on line 3 of Next_2, we have:*

$$i^\top + 1 = 2^k + \left(\mathit{Derivative}[k].i^\top + 1\right) \times 2^{k+1}.$$

It is an easy consequence of lemma 11.8 that in Next_2, after k is updated on line 3, we have for any j:

$$b_j(\mathit{Derivative}[k].i + 1) = b_{j+1}(i^\top + 1).$$

Thus, it is possible to avoid storing the i values, except in the main loop, and to re-generate them on-the-fly by evaluating b_j on the index of the main loop. These computations, although taking amortized constant time, can be made negligible by unrolling. To make notations less heavy, we introduce the following shorthand:

$$D[k_1, k_2, \ldots, k_\ell] := \mathit{State}.\mathit{Derivative}[k_1].\mathit{Derivative}[k_2]\ldots.\mathit{Derivative}[k_\ell].\mathbf{y}$$

With this notation, Algorithm 11.7 is just an unrolled version of the Algorithm 11.5 in which all the useless operations have been removed.

Algorithm 11.7 Iterative algorithm for all degrees.

```
1: procedure Zeroes(f)
2:     State ← Init^[d](f, 0, 0)
3:     for i from 0 to 2^n − 1 do
4:         if State.y = 0 then GrayCode(i) is a zero of f
5:         let k_1 = b_1(i + 1) in
6:         let k_2 = b_2(i + 1) in
7:         ⋮
8:         let k_d = b_d(i + 1) in
9:         if k_d > −1 then D [k_1, ..., k_{d−1}] ← D [k_1, ..., k_{d−1}] ⊕ D [k_1, ..., k_{d−1}, k_d]
10:        ⋮
11:        if k_3 > −1 then D [k_1, k_2] ← D [k_1, k_2] ⊕ D [k_1, k_2, k_3]
12:        if k_2 > −1 then D [k_1] ← D [k_1] ⊕ D [k_1, k_2]
13:        y ← y ⊕ D [k_1]
14:    end for
15: end procedure
```

11.4 Finding the Common Zeroes of Several Multivariate Polynomials

In the previous section, we discussed how to enumerate a single polynomial. We now move on to the enumeration of several polynomial simultaneously.

We will use several time the following simple idea: all the techniques we discussed previously perform a sequence of operations that is independent of the coefficients of the polynomials. Therefore, m instances of Algorithm 11.7 could be run in parallel on $f_1, \ldots, f_m$. All the parallel runs would execute the same instruction on different data, making the parallel combination efficiently implementable on any architectures with registers, not to mention vector or SIMD units. In each iteration of the main loop, it is easy to check if *all* the polynomials vanished on the current point of $(\mathbb{F}_2)^n$. Evaluating all the m polynomials in parallel using Algorithm 11.7 would require roughly $m \cdot d \cdot 2^n$ bit operations. The point of this section discussion is that it is possible to do better than this.

Note that if $m > n$, we can focus on the first n equations, since a system of n randomly chosen multivariate polynomial equations in n variables of constant degree d is expected to have a constant number of solutions, which can in turn be checked against the remaining equations efficiently. If $m < n$, then we can specialize $m-n$ variables, and solve the m equations in m variables for any possible values of the specialized variables. All-in-all, the interesting case is when $m = n$.

Let us first introduce a useful notation. Given an ordered set U, we denote the common zeroes of $f_1, \ldots, f_m$ belonging to U by $Z([f_1, \ldots, f_m], U)$. Let us also denote $Z_0 = (\mathbb{F}_2)^n$, and $Z_i = Z([f_i], Z_{i-1})$. It should be clear that $Z = Z_m$ is the set of common zeroes of the polynomials, and therefore the object we wish to obtain.

11.4.1 A General Technique: Splitting the Problem

A possible strategy is to compute the Z_i recursively: first Z_1, then Z_2, etc. However, while the algorithms of §11.3 can be used to compute Z_1, they cannot be used to compute Z_2 from Z_1, because they intrinsically enumerate all $(\mathbb{F}_2)^n$. In practice, the best results are in fact obtained by computing Z_k, for some well-chosen value of k, using k parallel runs of Algorithm 11.7, and then computing Z_m using a *secondary algorithm*. Computing Z_k requires $d \cdot k \cdot 2^n$ bit operations. It then remains to compute Z_m from Z_k, and to find the best possible value of k.

Also note that while it makes sense to choose k according to the targeted hardware platform (*e.g.*, $k = 32$ if 32-bit registers are available), it is an interesting theoretical problem choose k in order to minimize the global number of bit operations. We now move on to discuss several secondary algorithms to compute Z_m from Z_k, and discuss their relative merits.

11.4.2 Naive Secondary Evaluation

The simplest possibility is to compute Z_{i+1} from Z_i using naive evaluation, for $k \leq i \leq n-1$. It is clear that the expected cardinality of Z_i for random polynomial equations is 2^{n-i}. We will assume for the sake of simplicity that evaluating a degree-d polynomial requires $\binom{n}{d}$ bit op., following the reasoning of §11.2. Computing Z_{i+1} then requires about $\binom{n}{d} \cdot 2^{n-i}$ bit ops. The expected cost of computing Z is then approximately:

$$\sum_{i=k}^{n} \binom{n}{d} \cdot 2^{n-i} \approx \binom{n}{d} \cdot 2^{n-k+1} \text{ bit operations.}$$

Minimizing the global cost means solving the equation:

$$k \cdot d \cdot 2^n = \binom{n}{d} \cdot 2^{n-k+1}.$$

which is easily seen to be equivalent to:

$$(k \cdot \ln 2) \cdot \exp(k \cdot \ln 2) = 2 \cdot \binom{n}{d} \cdot \frac{\ln 2}{d}$$

Now, the Lambert W function is such that $W(x) \cdot \exp(W(x)) = x$. Thus, the solution of our equation is:

$$k = W\left(\binom{n}{d} \cdot \frac{2 \cdot \ln 2}{d}\right) / \ln 2$$

Using the known fact [dB61] that when x goes to infinity:

$$W(x) = \ln x - \ln \ln x + o(\ln \ln x)$$

we find that when $n \to \infty$:

$$k = 1 + \log_2 \left(\binom{n}{d} \cdot \frac{1}{d} \right) + \mathcal{O}(\ln \ln n)$$

The full cost of the algorithm is then approximately $d^2 \cdot \log_2 n \cdot 2^{n+1}$ bit operations.

11.4.3 Differential Secondary Evaluation

We only describe the quadratic case, but this could be extended to higher degrees. We can efficiently evaluate Z_{i+1} from Z_i using an easy consequence of equation (10.1): given $f(\mathbf{x})$, computing $f(\mathbf{x} + \Delta)$ takes $2|\Delta| \cdot n$ bit operations, where $|\Delta|$ denote the hamming weight of Δ, by computing $|\Delta|$ dot products with the derivatives. Let us order the elements of Z_i by writing: $Z_i = \left\{ \mathbf{x}_1^i, \ldots \mathbf{x}_{q_i}^i \right\}$ (the elements are ordered using the usual lexicographic order), and $\Delta_j^i = \mathbf{x}_{j+1}^i \oplus \mathbf{x}_j^i$.

Computing Z_{i+1} therefore requires approximately:

$$2n \cdot \sum_{j=1}^{q_i - 1} |\Delta_j^i| \text{ bit operations.}$$

Now, let us consider the Δ_j^i as integer number between 0 and $2^n - 1$. The x_{j+1}^i are the zeroes of a set of i random polynomials, and under the assumption that each point of $(\mathbb{F}_2)^n$ has one chance over 2^i to be such a zero, then the difference Δ_j^i between two such consecutive zeros follows a geometric distribution of parameter 2^{-i}, and thus has expectation 2^i. The hamming weight $|\Delta_j^i|$ is upper-bounded by $\lceil \log_2 \Delta_j^i \rceil$ (considered as an integer), and therefore $|\Delta_j^i|$ has expectation less than i.

Computing Z_{i+1} therefore requires on average $2n \cdot i \cdot 2^{n-i}$ bit op. Finally, computing Z from Z_k requires on average:

$$2n \cdot \sum_{i=k}^{n-1} i \cdot 2^{n-i} \leq 4n \cdot (k+1) \cdot 2^{n-k} \text{ bit operations}$$

An approximately optimal value of k would then satisfy

$$2k \cdot 2^n = 4n \cdot (k+1) \cdot 2^{n-k}$$

which is approximately $k = 1 + \log_2 n$. The complexity of the whole procedure is then $4 \log_2 n \cdot 2^n$. However, implementing this technique efficiently looks like a lot of work for at best a $2\times$ gain.

11.4.4 Practical Considerations

Choosing the "optimal" value of k is not only of theoretical interest, but may have a practical significance if a very *ad hoc* circuit were to be designed from scratch. Even in the software implementations we are concerned with in this paper, it provides a guideline. However, when implemented in software on processors with registers, the logical operation width of the hardware becomes a determinant argument in the actual choice of k. If operations are always performed on ω-bit registers, then it is likely that the best choice of k is precisely ω. In all our implementations, we used the "Early-abort + Naive Evaluation" strategy with $k = 32$. This enables us to make use of the full register width, while keeping the "naive evaluation" time negligible. However, this means that the enumeration process must store 32 times more data in fast memory, compared to the evaluation of only one polynomial.

11.5 Spatial and Temporal Proximity

The critical loop of Algorithm 11.7 is very short, since it performs only d logical operations. Implementing it *very* efficiently is nevertheless a bit tricky. For instance, it accesses the memory $d+1$ times, which suggests that memory bandwith will be an actual performance bottleneck.

The remaining of this chapter is devoted to the issue of efficiently implementing Algorithm 11.7. In § 11.6, we will show that the memory-bandwidth problem present itself again, yet differently, when we try to parallelize the algorithm on shared-memory architectures. In this section, we argue that the algorithm is *cache-efficient*, *i.e.*, that is uses the cache efficiently regardless of its size. This shows that the algorithm has good spatial and temporal proximity, and cannot be really improved in this respect.

Cache Model. We will study the behavior of the algorithm in the *Ideal Cache Model* [FLPR99]. This model considers a computer with a two-level memory hierarchy consisting of an ideal (data) cache of Z words and an arbitrarily large main memory. The cache is partitioned into cache lines, each consisting of L consecutive words that are always moved together between cache and main memory. The processor can only reference words that reside in the cache. If the referenced word belongs to a line already in cache, a *cache hit* occurs, and the word is delivered to the processor. Otherwise, a *cache miss* occurs, and the line is fetched into the cache. The ideal cache is *fully associative*: cache lines can be stored anywhere in the cache. If the cache is full, a cache line must be evicted. The ideal cache uses the optimal off-line strategy of replacing the cache line whose next access is farthest in the future, and thus it exploits temporal locality perfectly. An algorithm with an input of size n is measured in the ideal-cache model in terms of the usual number of operation performed by the processor, but also in terms of its *Cache Complexity* $Q(n, Z, L)$ – the number of cache misses it incurs as a function of Z and L. We now move on to evaluate the cache complexity of the enumeration algorithm, as show in fig. 11.7. We will assume that the "memory cells" accessed by the algorithm have the same size as the a word in the cache (if this were not the case, it would only incur a constant multiplicative loss, and we are mostly interested in an asymptotic result).

Cyclic Memory Accesses in Algorithm 11.7. The memory words accessed in the algorithm belong to arrays of various dimension, and are accessed with indices of variable length. It should be clear from the description of the algorithm that for all $k \leq d$, the memory location of index $[i_1, \ldots, i_k]$ is accessed at step s if $b_j(s) = i_j$, for all $j \leq k$. This memory access pattern is in fact very regular. We say that a memory word is accessed with period T if, when it is accessed at iteration i, it is also accessed at iteration $i + T$, but not in-between.

Lemma 11.9. *For all $k \leq d$, the memory location of index $[i_1, \ldots, i_k]$ is accessed with period 2^{i_k+1}.*

Proof. We associate with an index $[i_1, \ldots, i_k]$ the set $\Omega_{i_1,\ldots,i_k}$ of integers n such that $b_1(n) = i_1, \ldots, b_k(n) = i_k$. The problem amounts to show that the difference between two consecutive elements of this set is 2^{i_k+1}. But it is easily seen that if $n \in \Omega_{i_1,\ldots,i_k}$, then $n + j \cdot 2^{i_k+1} \in \Omega_{i_1,\ldots,i_k}$ for any positive integer j. This follows from the fact that $b_j(n) = b_j(n + 2^\ell)$ if $\ell > j$, and establishes the result. □

It should be clear that all the memory location accessed with period exactly T are accessed in the first T iteration of the main loop. Moreover, they are accessed in a certain order. For instance, memory words with period 8 are accessed in this order in the first 8 iterations: $[2], [0, 2], [1, 2]$. By definition of the period, this access pattern is reproduced without modifications in the next T iterations. Thus, memory words with period T are accessed in a *cyclic* fashion.

The algorithm easily defines a total order relation on the memory locations it accesses: $x \leq y$ if and only if the first access to x takes places *before* the first access to y. Let us assume that the actual memory addresses are compatible with this order relation. Then, more frequently accessed words are stored with the lowest addresses, and words with the same access frequency are stored contiguously in memory. There are $\sum_{i=0}^{\min(d-1,k)} \binom{k}{i}$ memory locations that are accessed with period 2^{k+1}.

Critical Period. This being said, we will focus on the case where all the $\sum_{i=0}^{d} \binom{n}{i}$ memory words accessed by the algorithm do not fit into the cache, to avoid studying the trivial case. Let us define the *critical period* 2^{T_c+1} to be the biggest integer such that all the memory words accessed with period 2^{T_c+1} fit in the cache:

$$\sum_{k=0}^{T_c} \sum_{i=0}^{\min(d-1,k)} \binom{k}{i} \leq Z - 1$$

Under the (mild) assumption that the cache contains $Z \geq 2^d$ words, and thus that T_c is greater than d, this condition becomes:

$$2^d - 1 + \sum_{k=d}^{T_c} \sum_{i=0}^{d-1} \binom{k}{i} \leq Z - 1$$

This is the summation in a rectangle inside Pascal's triangle, then by applying Pascal's rule recursively, we may simplify this expression, and find that it is equivalent to:

$$\sum_{i=0}^{d} \binom{T_c + 1}{i} \leq Z$$

T_c can be easily expressed as a function of Z when d is small:

$$d = 2 \quad \rightarrow \quad T_c + 1 = \frac{\sqrt{8Z - 7} - 1}{2}$$

$$d = 3 \quad \rightarrow \quad T_c + 1 = \frac{\left(3 \cdot (Z-1) + \sqrt{Z^2 - 2Z + 368/243}\right)^{\frac{2}{3}} - 15}{\left(3 \cdot (Z-1) + \sqrt{Z^2 - 2Z + 368/243}\right)^{\frac{1}{3}}}$$

The important point is that all memory words with period 2^{T_c+1} fit in the cache and *do not leave it*. This fact is almost true by definition of T_c: the optimal off-line cache strategy will not evict a cache line that will be accessed in T steps if it can evict a cache line that will only be accessed in $2T$ steps. And there will always be a cache line not containing a word accessed with period 2^{T_c+1} or less. Note that this shows that the cache complexity of the algorithm is essentially independent of L in the ideal cache model. This being said, we can state our result:

Theorem 11.10. *Under the assumption that $T_c \geq 2d$, the following two inequalities hold:*

i) $Q(n, d, Z, L) \leq 2^{n-2-T_c} \cdot (d+1) \cdot \binom{T_c + 1}{d - 1}$

ii) $Q(n, d, Z, L) \leq 2^{n-2-T_c} \cdot \dfrac{d \cdot (d+1)}{T_c + 2 - d} \cdot Z$

Proof. The second inequality is a nearly-direct consequence of the first one, and of the definition of T_c. Let us thus focus on the first one.

By definition of T_c, the algorithm may make a cache miss every time it accesses a memory location whose index contain a coordinate bigger than T_c. Such memory words have period greater than 2^{T_c+2}, so each of them is accessed at most 2^{n-T_c-2} times. Multiplying this by the number of such memory words yields the total number of cache misses:

$$\begin{aligned} Q(n, d, Z, L) &= \sum_{k=T_c+1}^{n} 2^{n-1-k} \cdot \sum_{i=0}^{d-1} \binom{k}{i} \\ &\leq 2^{n-1} \sum_{i=0}^{d-1} \sum_{k=T_c+1}^{+\infty} 2^{-k} \cdot \binom{k}{i} \end{aligned}$$

It is well-known that $\sum_k \binom{k}{i} x^k = x^i/(1-x)^{i+1}$. Thus, we find with $x = 1/2$:

$$\sum_{k=0}^{+\infty} 2^{-k} \cdot \binom{k}{i} = 2$$

We can therefore rewrite

$$Q(n, d, Z, L) \leq 2^{n-1} \cdot \sum_{i=0}^{d-1} \left(2 - \sum_{k=0}^{T_c} 2^{-k} \cdot \binom{k}{i}\right)$$

Now we claim that

$$2 - \sum_{k=0}^{T_c} 2^{-k} \cdot \binom{k}{i} = 2^{-T_c} \cdot \sum_{j=0}^{i} \binom{T_c + 1}{j}.$$

It is not particularly difficult to establish this by induction on i, by using Pascal's rule. Going back to the expression of $Q(n, d, Z, L)$, we find:

$$\begin{aligned} Q(n, d, Z, L) &\leq 2^{n-T_c-1} \cdot \sum_{i=0}^{d-1} \sum_{j=0}^{i} \binom{T_c + 1}{j} \\ &\leq 2^{n-1-T_c} \cdot \sum_{j=0}^{d-1} (d-j) \cdot \binom{T_c + 1}{j} \end{aligned}$$

And since $\binom{n}{k} = \frac{n}{k}\binom{n-1}{k-1}$, we obtain:

$$Q(n,d,Z,L) \quad \leq \quad 2^{n-1-T_c} \cdot \left(d \cdot \sum_{j=0}^{d-1} \binom{T_c+1}{j} - (T_c+1) \cdot \sum_{j=1}^{d-1} \binom{T_c}{j-1} \right)$$

We next claim that by applying Pascal's rule recursively, we obtain:

$$2 \cdot \sum_{j=1}^{d-1} \binom{T_c}{j-1} \quad = \quad \binom{T_c}{d-2} + \sum_{j=0}^{d-2} \binom{T_c+1}{j}$$

Substituting this into $Q(n,d,Z,L)$ yields:

$$\begin{aligned}
Q(n,d,Z,L) &\leq 2^{n-1-T_c} \cdot \left(d \cdot \sum_{j=0}^{d-1} \binom{T_c+1}{j} - \frac{T_c+1}{2} \cdot \left[\binom{T_c}{d-2} + \sum_{j=0}^{d-2} \binom{T_c+1}{j} \right] \right) \\
&\leq 2^{n-1-T_c} \cdot \left(d \cdot \binom{T_c+1}{d-1} + \frac{2d-T_c-1}{2} \cdot \sum_{j=0}^{d-2} \binom{T_c+1}{j} - \frac{T_c+1}{2} \cdot \binom{T_c}{d-2} \right) \\
&\leq 2^{n-1-T_c} \cdot \left(\frac{(d+1)(T_c+1)}{2(d-1)} \cdot \binom{T_c}{d-2} + \frac{2d-T_c-1}{2} \cdot \sum_{j=0}^{d-2} \binom{T_c+1}{j} \right) \\
&\leq 2^{n-1-T_c} \cdot \left(\frac{d+1}{2} \cdot \binom{T_c+1}{d-1} + \frac{2d-T_c-1}{2} \cdot \sum_{j=0}^{d-2} \binom{T_c+1}{j} \right)
\end{aligned}$$

And under the (again mild) assumption that $T_c \geq 2d$, we find:

$$Q(n,d,Z,L) \leq 2^{n-2-T_c} \cdot (d+1) \cdot \binom{T_c+1}{d-1}$$

□

Numerical Application. Let us consider a polynomial in 64 variables. If we assume an incredibly small cache of $Z = 2^{10}$ bits and that our polynomial is of degree 2, then $T_c = 44$ and the enumeration will make about 2^{25} cache misses, for a running time of at least 2^{65}. If we assume that our polynomial is of degree 4, and that the cache is 2^{14}-bit large, then $T_c = 24$, and there will be 2^{52} cache misses, for more than 2^{66} attempted memory accesses.

11.5.1 Transforming the Equations to do Less Work

We conclude this section by presenting a ruse allowing to cut off a fraction $1/(d \cdot 2^d)$ of the work in some cases. This is mostly interesting when $d = 2$, where it reduces the workload by 12.5%. The most frequently accessed constant is $D[0, 1, 2, \ldots, d-1]$. If this constant were equal to zero, then we could just remove the code that XORs it. This constant is accessed with period 2^d, and there are $d \cdot 2^d$ XOR between two accesses to this constant.

Now, how can we force $D[0, 1, 2, \ldots, d-1]$ to be zero? Suppose we want to find the solutions of a polynomial system (f_1, f_2), and that the first step is to find all the zeroes of f_1. The constant has a 50% chance to be zero by itself, but if it is not, we can always first enumerate f_2. If the constant is again non-zero when enumerating f_2, we can enumerate $f_1 + f_2$ instead without affecting the solution set, but this time the constant will be zero by linearity of the differential (*i.e.*, $\mathrm{D}(\mathbf{f} + \mathbf{g}) = \mathrm{D}\mathbf{f} + \mathrm{D}\mathbf{g}$). This easily generalizes to more than two polynomials (one must however be carefull to perform an invertible linear combination of the polynomials).

11.6 Parallelization-Related Issues

Parallelizing Algorithm 11.7 is awfully easy, as the problem is "embarrassingly parallel". It simply comes down to partitioning the search space into the number of available cores. For instance, the main loop of Algorithm 11.1 can be split in independent chunks, as illustrated by Algorithm 11.8. On distributed-memory

Algorithm 11.8 Parallel enumeration, assuming one processing unit capable of running 2^T threads.

```
procedure DUMB-PARALLEL-ZEROES(f, T)
    parallel-for t from 0 to 2^T − 1 do
        State[t] ← INIT(f, 0, GRAYCODE(t · 2^{n−t}))
        for i from 0 to 2^{n−t} − 1 do
            if State[t].y = 0 then State[t].x is a zero of f
            NEXT(State[t])
        end for
    end parallel-for
end procedure
```

architectures, this would be perfectly fine. However, on shared-memory architectures, the problem of any parallelization is that it will put even more pressure on the memory subsystem than the sequential version. Recall from §11.3.4 that the *State* is made of a read-write part (the "internal state") and of a read-ony part (the "constants"). That each thread needs its own internal state makes perfect sense. However, that each thread needs its own copy of the constants seems wasteful, because these constants are functions of the enumerated polynomial(s), and all threads enumerate the same. However, in Algorithm 11.8, the constants also depends on the "starting point" of the enumeration (the third argument of INIT), and therefore they are different in all threads.

This seems suboptimal, if only because it cannot exploit shared caches in an efficient way (most modern multi-core CPUs have a large Level-3 cache shared between all the cores). The memory bandwidth problem becomes even more pressing on some parallel architectures, such as GPUs, or the Cell, in which the "fast" memory available to each core is fairly restricted, and main memory is relatively slow and/or inaccessible. Even on a single-core of a desktop CPUs it is possible to run several parallel copies of Algorithm 11.7 using SIMD instructions, but if multiple instances compete for the caches and try to access the main memory simultaneously, then the pressure on the memory subsystem should increase.

In addition, in some parallel architecture there are no caches, but instead a small "scratch pad" of fast memory that must be *explicitly* managed by the program (as opposed to caches that are managed by the CPU itself). This means we will need to decide, given its size, what goes in the (fast) scratch pad, and what remains in the (slow) main memory. This can be made even more complicated by the fact that the scratch pad is shared between multiple threads). Note that the previous section gives reasonably good guidelines to deal with this problem.

Sharing Constants. While the increased memory consumption of running multiple instances of any enumeration algorithm cannot be completely avoided, because each copy needs its own internal state, the problem can be partially alleviated by sharing the constants between all the threads. This is essentially achieved by splitting the main loop of algorithm 11.7 in chunks of a given size (say, 2^L) that can be processed concurrently. This results in Algorithm 11.9. Enumerating a chunk of size 2^L is comparable to enumerating a polynomial in L variables (it makes the same number of calls to NEXT).

Internal State. The full sequential process stores and maintains an internal state of $\sum_{i=0}^{d-1} \binom{n}{i} = \mathcal{O}\left(n^{d-1}\right)$ bits. However, when processing a chunk of size 2^L, a smaller internal state is sufficient. Counting the number of reachable internal state bits amounts to counting the number of possible tuples $(k_1, k_2, \ldots, k_{d-1})$ that will be generated in the loop.

Our first observation is that as soon as the hamming weight of $i+1$ is greater than d, then all the k_j's will be smaller than L. When all the k indices are smaller than L, only $\sum_{i=0}^{d-1} \binom{L}{i}$ bits of internal state can be addressed. Now, let us assume that HAMMINGWHEIGHT$(i+1) = w < d$. In this case, we find

$$k_j = \begin{cases} b_j(i+1) & \text{when } j \leq w \\ L + b_{j-w}(i_0) & \text{otherwise} \end{cases}$$

So, when $w < d$, then the value of $k_j (j > w)$ is uniquely determined by i_0, and is greater than L. Thus, in a given thread, for a given value of w, the number of possible tuples $(k_1, k_2, \ldots, k_{d-1})$ is precisely the number of possible values of i with hamming weight w.

Proposition 11.11. *An internal state of* $2 \cdot \sum_{i=0}^{d-1} \binom{L}{i} = \mathcal{O}\left(L^{d-1}\right)$ *bits is sufficient for each thread.*

Algorithm 11.9 Parallel Iterative algorithm for all degrees.

procedure SMART-PARALLEL-ZEROES(f, L, T)
  initialize C (the constants shared by all the threads)
  for b **from** 0 **to** $2^{n-L-T}-1$ **do**
    parallel-for t **from** 0 **to** 2^T-1 **do**
      initialize S_t (the internal state of thread t)
      let $i_0 = \left(b + t \cdot 2^{n-T-L}\right) \cdot 2^L$ **in**
      for i **from** 0 **to** 2^L-1 **do**
        if $\mathbf{y}_t = 0$ **then** GRAYCODE(i_0+i) is a zero of f
        let $k_1 = b_1(i_0+i+1)$ **in**
        let $k_2 = b_2(i_0+i+1)$ **in**
        $\vdots$
        $k_d = b_d(i_0+i+1)$ **in**
        let if $k_d > -1$ **then** $S_t[k_1,\ldots,k_{d-1}] \leftarrow S_t[k_1,\ldots,k_{d-1}] \oplus C[k_1,\ldots,k_{d-1},k_d]$
        $\vdots$
        if $k_3 > -1$ **then** $S_t[k_1,k_2] \leftarrow S_t[k_1,k_2] \oplus S_t[k_1,k_2,k_3]$
        if $k_2 > -1$ **then** $S_t[k_1] \leftarrow S_t[k_1] \oplus S_t[k_1,k_2]$
        $\mathbf{y}_t \leftarrow \mathbf{y}_t \oplus S_t[k_1]$
      end for
    end parallel-for
  end for
end procedure

Synchronous Behavior. Let us consider for a moment than all the 2^T threads are executed synchronously (on an SIMD unit for instance). In this setting it is an advantage that all the threads behave similarly: if they access the same memory location, then often it can be read once and broadcast to all the threads. Conversely, if they access contiguous memory locations, then the queries can sometimes be coalesced and served with only one request from the main memory.

When the hamming weight of i is greater than d, then the tuple $(k_1,\ldots,k_d)$ depends only on i, and therefore is the same in all the thread. This places us in the favorable situation described above. In fact, this favorable case presents itself a bit more often. For instance, if HAMMINGWHEIGHT$(i+1) = w < d$ and if b has hamming weight at least $d-w$, then all the threads will end up having the same k_j's. In fact, this situation does not happen only when $b \cdot 2^L i + 1$ has hamming weight less than d, and it is easy to see that the number of times this happens is

$$\sum_{i=0}^{d-1} \binom{n-T}{i} = \mathcal{O}\left((n-T)^{d-1}\right). \tag{11.3}$$

This means that even though the algorithm performs an exponential number of steps, all the 2^T threads behave synchronously except in a polynomial number of steps. This feature is interesting, because on some architecture penalties apply when threads do not behave synchronously. Table 11.2 shows how the algorithm can be run with 4 threads and obtain the number of non-broadcasts advertised

11.7 Implementations

11.7.1 A Brief Description of the Hardware Platforms

Vector Units on x86-64. The most prevalent SIMD (single instruction, multiple data) instruction set today is SSE2, available on all current Intel-compatible CPUs. SSE2 instructions operate on 16 architectural `xmm` registers, each of which is 128-bit wide. We use integer operations, which treat `xmm` registers as vectors of 8-, 16-, 32- or 64-bit operands.

The SSE instruction set includes Loads and Stores (to/from `xmm` registers, memory — both aligned and unaligned, and traditional registers), Packing/Unpacking/Shuffling, Logical Operations (`AND, OR, NOT, XOR`, Shifts Left, Right Logical and Arithmetic — bit-wise on units and byte-wise on the entire `xmm` register), and Arithmetic (add, substract, multiply, max-min) with some or all of the arithmetic widths. The interested reader is referred to Intel and AMD's manuals for details on these instructions, and to references such as [Fog10] for throughput and latencies.

thread 0				thread 1				thread 2				thread 3				
$i_0 \gg L$	$i+1$	k_1	k_2	$i_0 \gg L$	$i+1$	k_1	k_2	$i_0 \gg L$	$i+1$	k_1	k_2	$i_0 \gg L$	$i+1$	k_1	k_2	
0000	001	0	-1	0100	001	0	5	1000	001	0	6	1100	001	0	5	✓
0000	010	1	-1	0100	010	1	5	1000	010	1	6	1100	010	1	5	✓
0000	011	0	1	0100	011	0	1	1000	011	0	1	1100	011	0	1	
0000	100	2	-1	0100	100	2	5	1000	100	2	6	1100	100	2	5	✓
0000	101	0	2	0100	101	0	2	1000	101	0	2	1100	101	0	2	
0000	110	1	2	0100	110	1	2	1000	110	1	2	1100	110	1	2	
0000	111	0	1	0100	111	0	1	1000	111	0	1	1100	111	0	1	
0000	1000	3	-1	0100	1000	3	5	1000	1000	3	6	1100	1000	3	5	✓
0001	001	0	3	0101	001	0	3	1001	001	0	3	1101	001	0	3	
0001	010	1	3	0101	010	1	3	1001	010	1	3	1101	010	1	3	
0001	011	0	1	0101	011	0	1	1001	011	0	1	1101	011	0	1	
0001	100	2	3	0101	100	2	3	1001	100	2	3	1101	100	2	3	
0001	101	0	2	0101	101	0	2	1001	101	0	2	1101	101	0	2	
0001	110	1	2	0101	110	1	2	1001	110	1	2	1101	110	1	2	
0001	111	0	1	0101	111	0	1	1001	111	0	1	1101	111	0	1	
0001	1000	4	-1	0101	1000	4	5	1001	1000	4	6	1101	1000	4	5	✓
0010	001	0	4	0110	001	0	4	1010	001	0	4	1110	001	0	4	
0010	010	1	4	0110	010	1	4	1010	010	1	4	1110	010	1	4	
0010	011	0	1	0110	011	0	1	1010	011	0	1	1110	011	0	1	
0010	100	2	4	0110	100	2	4	1010	100	2	4	1110	100	2	4	
0010	101	0	2	0110	101	0	2	1010	101	0	2	1110	101	0	2	
0010	110	1	2	0110	110	1	2	1010	110	1	2	1110	110	1	2	
0010	111	0	1	0110	111	0	1	1010	111	0	1	1110	111	0	1	
0010	1000	3	4	0110	1000	3	4	1010	1000	3	4	1110	1000	3	4	
0011	001	0	3	0111	001	0	3	1011	001	0	3	1111	001	0	3	
0011	010	1	3	0111	010	1	3	1011	010	1	3	1111	010	1	3	
0011	011	0	1	0111	011	0	1	1011	011	0	1	1111	011	0	1	
0011	100	2	3	0111	100	2	3	1011	100	2	3	1111	100	2	3	
0011	101	0	2	0111	101	0	2	1011	101	0	2	1111	101	0	2	
0011	110	1	2	0111	110	1	2	1011	110	1	2	1111	110	1	2	
0011	111	0	1	0111	111	0	1	1011	111	0	1	1111	111	0	1	
0011	1000	5	-1	0111	1000	6	-1	1011	1000	5	6	1111	1000	7	-1	✓

Table 11.2: Enumeration with $n = 7$, in chunks of $L = 2^3$ elements with 4 batches of $T = 4$ concurrent threads. The tick marks (✓) means that the 4 threads do not access the same constant. In conformance with (11.3), there are $1 + (7 - 2) = 6$ non-broadcast memory accesses.

G2xx-series Graphics Processing Units from NVIDIA. We choose NVIDIA's G2xx GPUs as they have the least hostile GPU parallel programming environment called CUDA (Compute Unified Device Architecture). In CUDA, we program in the familiar C/C++ programming language plus a small set of GPU extensions.

An NVIDIA GPU contains anywhere from 2–30 streaming multiprocessors (MPs). There are 8 ALUs (streaming processors or SPs in market-speak) and two super function units (SFUs) on each MP. A top-end "GTX 295" card has two GPUs, each with 30 MPs, hence the claimed "480 cores". The theoretical throughput of each SP per cycle is one 32-bit integer or floating-point instruction (including add/subtract, multiply, bitwise AND/OR/XOR, and fused multiply-add), and that of an SFU 2 floating-point multiplications or 1 special operation. The arithmetic units have 20+-stage pipelines.

Main memory is slow and forms a major bottleneck in many applications. The read bandwidth from memory on the card to the GPU is only one 32-bit read per cycle per MP and has a latency of > 200 cycles. To ease this problem, the MP has a register file of 64 KB (16,384 registers, max. of 128 per thread), a shared memory of 16 KB, and an 8-KB cache for read-only access to a declared "constant region" of at most 64 KB. Every cycle, each MP can achieve one read from the constant cache, *which can broadcast to many thread at once*.

Each MP contains a scheduling and dispatching unit that can handle a large number of lightweight threads. However, the decoding unit can only decode once every 4 cycles. *This is typically 1 instruction, but certain common instructions are "half-sized", so two such instructions can be issued together if independent.* Since there are 8 SPs in an MP, CUDA programming is always on a Single Program Multiple Data basis, where a "warp" of threads (32) should be executing the same instruction. If there is a branch which is taken by some thread in a warp but not others, we are said to have a "divergent" warp; from then on only part of the threads will execute until all threads in that warp are executing the same instruction again. Further, as the latency of a typical instruction is about 24 cycles, NVIDIA recommends a minimum of 6 warps on each MP, although it is sometimes possible to get acceptable performance with 4 warps.

11.7.2 Implementations Details

Preliminary implementations have been written in C (using the compiler's "intrinsics" to generate SSE instructions), and using the CUDA tools for GPUs. We used the fact that both GT2xx and x86 are essentially SIMD units processing several 32-bit registers at the same time to enumerate the first 32 polynomials simultaneously using algorithm 11.9. We then expect 2^{n-32} points of $(\mathbb{F}_2)^n$ satisfying the first 32 equations, that have to be checked against the remaining $n-32$ equations. This is done using a bitsliced version of the naive evaluation strategy.

The inner loop of algorithm 11.9 was unrolled 2^8 times to avoid explicitly computing the k_j indices. On GPUs it is useful to run more threads than the number of hardware cores, to amortize the latency of the arithmetic units and of the memory. There is however a delicate tradeoff: running more threads gives the scheduler an opportunity to run instructions from other threads while some threads are waiting for the (slow) main memory, yet more threads consumme more memory, causing more frequent access to the main memory, and more waiting time... We chose to run 2^7 threads on each MP, and 2^{16} threads in total. On CPUs, we used SIMD instructions to run 4 threads simultaneously. Each thread enumerates a chunk of $L \approx 2^{30}$ elements.

11.7.3 Practical Results

Tables 11.3 and 11.4 shows what our preliminaries implementations are capable of. They are currently the fastest software for solving (generic, dense) systems of multivariate quadratic equations over $\mathbb{F}_2$. Table 11.4 in particular shows that on modern CPUs equiped with efficient SIMD units, our implementation checks on average more than two vectors of $(\mathbb{F}_2)^n$ in one CPU cycle. We consider this to be a decent achievement in terms of implementation quality.

Performances degrade when the degree increases because of the increased pressure on the cache and memory subsystem. The impact on the GPUs (where fast memory is scarce) is clearly visible. The impact is also greater on Phenom CPUs, because they have less cache than the others. The benchmarked Intel CPUs are more "efficient" than their AMD counterpart because they can dispatch one more SSE instructions per cycle (3 instead of 2). Writing efficient implementations is challenging. It is very time-con summing, as it requires many experiments. It also requires a quite deep understanding of the underlying hardware. For instance, developing these implementations lead us to discover an *undocumented feature* of G2xx series GPUs: the CUDA compiler (`nvcc`) reliably generates conditional move instructions, that are executed by the SPs and the SFUs very efficiently.

Time (minutes)			Testing platform				#cores	$n = 64$ in 1 month
$d = 2$	$d = 3$	$d = 4$	GHz	Arch.	Name	cost	(#used)	budget
1217	2686	3191	2.2	K10	Phenom 9550	\$120	4(1)	\$54,000
1157	1992	2685	2.3	K10+	Opteron 2376	\$184	4(1)	\$113,316
142	240	336	2.3	K10+	Opteron 2376×2	\$368	8(8)	
780	1364	1819	2.4	C2	Xeon X3220	\$210	4(1)	\$60,720
671	1176	1560	2.83	C2+	Core2 Q9550	\$225	4(1)	\$55,575
179	294	390	2.83	C2+	Core2 Q9550	\$225	4(4)	
761	1279	1856	2.26	Ci7	Xeon E5520	\$385	4(1)	\$78,720
95	154	225	2.26	Ci7	Xeon E5520×2	770	8(8)	
41	73	271	1.3	G200	GTX 280	n/a	240	n/a
21	36	126	1.25	G200	GTX 295	\$500	480	\$15,500

Table 11.3: Performance results for $n = 48$ and projected budgets for solving $n = 64$ in one month (as of summer 2010)

$n = 32$			$n = 40$			$n = 48$			Testing platform		
$d = 2$	$d = 3$	$d = 4$	$d = 2$	$d = 3$	$d = 4$	$d = 2$	$d = 3$	$d = 4$	GHz	Arch.	Name
0.58	1.21	1.41	0.57	1.27	1.43	0.57	1.26	1.50	2.2	K10	Phenom9550
0.57	0.91	1.32	0.57	0.98	1.31	0.57	0.98	1.32	2.3	K10+	Opteron2376
0.40	0.65	0.95	0.40	0.70	0.94	0.40	0.70	0.93	2.4	C2	Xeon X3220
0.40	0.66	0.96	0.41	0.71	0.94	0.41	0.71	0.94	2.83	C2+	Core2 Q9550
0.50	0.66	1.00	0.38	0.65	0.91	0.37	0.62	0.89	2.26	Ci7	Xeon E5520
2.87	4.66	15.01	2.69	4.62	17.94	2.72	4.82	17.95	1.296	G200	GTX280
2.93	4.90	14.76	2.70	4.62	15.54	2.69	4.57	15.97	1.242	G200	GTX295

Table 11.4: Efficiency comparison: cycles per $(\mathbb{F}_2)^n$-vector tested on one core

Chapter 12

"Isomorphism of Polynomials" problems

This chapter introduces the polynomial linear equivalence problem(s) we are concerned with. We discuss its cryptographic relevance, and provide a classification of instances.

The equivalence of linear maps is a standard notion of linear algebra. Geometrically speaking, two linear maps are said to be equivalent if they describe the same transformation, but with input and output coordinates expressed in different bases. In algebraic terms, two matrices A and B represent equivalent linear maps if there exist two invertible matrices S and T such that $B = T \times A \times S$. Here, S changes the basis in which the input coordinates are expressed, while T changes the basis of the output space.

It is well-known that S and T exist if and only if A and B have the same rank. Therefore, there is an efficient algorithm for the *decision problem*, namely the problem of testing whether two given matrices are equivalent: it suffices to check whether the ranks of A and B are equal. The *search problem*, *i.e.*, actually finding S and T if they exist, is a bit more complicated, but efficient algorithms are nevertheless available.

This equivalence relation lends itself to a natural generalization, wich is the equivalence of *polynomial maps*. Two polynomial maps $\mathbf{a}$ and $\mathbf{b}$ are said to be (linearly) equivalent if there exist two bijective endomorphisms S and T such that $\mathbf{b} = T \circ \mathbf{a} \circ S$. The analogy with the usual matrix equivalence is blatant, and it carries the same geometrical meaning: two equivalent polynomial maps describe the same transformation, but with coordinates expressed in potentially different bases. We observe that it is equivalent, and much more convenient, to work with the following definition of S and T:

$$T \circ \mathbf{b} = \mathbf{a} \circ S \tag{12.1}$$

The equivalence of polynomial maps naturally raises algorithmic issues. There is again a *decision problem* ("Are $\mathbf{a}$ and $\mathbf{b}$ equivalent ?") and a *search problem* ("Given the promise that $\mathbf{a}$ and $\mathbf{b}$ are equivalent, what are S and T ?"). We will forget about the decision problem and focus on the search problem, which we call Polynomial Linear Equivalence (PLE), as it does not seem any easier, and is more relevant. Also, the degree-two case is special enough to give it a name of its own. The Quadratic Maps Linear Equivalence problem (QMLE) refers to the special subclass of PLE where the total degree of the polynomials defining the maps is two.

The equivalence of polynomial maps will look familiar to anyone who has been taught the usual theory of quadratic forms. Recall from §10.2 that two quadratic forms f and g are said to be equivalent if there exist a invertible matrix S such that $f = g \circ S$. When $n = 2$, the equivalence classes are formed of parabolas, hyperbolas, etc. Again, the equivalence notion has a geometic interpretation: equivalent quadratic forms define the same "kind of curve". We have seen that testing the equivalence of quadratic forms is easy, even in characteristic two, because of the existence of easily-computable canonical forms.

The equivalence of quadratic forms also lend itself to several natural generalizations. We are interested in particular in the equivalence of higher-degree forms, and most notably the Cubic Form Equivalence (CFE) problem. In an orthognal direction, an interesting problem is to test the *simultaneous* equivalence of more than one pair of quadratic forms: given two pairs of quadratic forms f, f', g and g', is there an invertible matrix S such that $f = g \circ S$ and $f' = g' \circ S$?

Example 12.1. Let us consider the curve of $\mathbb{R}^3$ formed by all the points over which the two quadratic polynomials $\mathbf{a}_1 = y^2 - x$ and $\mathbf{a}_2 = (x + z)^2 - y$ vanish. Now, let us pick an invertible random change of coordinates:

$$S = \begin{pmatrix} 7 & 2 & -4 \\ 6 & 7 & -3 \\ -5 & -9 & 1 \end{pmatrix}$$

By construction, the zeroes of $\mathbf{b} = \mathbf{a} \circ S$ describe the same curve as those of $\mathbf{a}$, but in a linearly transformed space (or, equivalently, with coordinates expressed in a different basis). We easily compute $\mathbf{b}$:

$$\begin{aligned} \mathbf{b}_1 &= 4x^2 + 28xy - 36xz + 49y^2 - 126yz + 81z^2 - 7x - 6y + 5z \\ \mathbf{b}_2 &= 9x^2 + 18xy - 24xz + 9y^2 - 24yz + 16z^2 - 2x - 7y + 9z \end{aligned}$$

By construction, $\mathbf{a}$ and $\mathbf{b}$ are made of two simultaneously equivalent quadratic polynomials.

Of course, we are not restricted to the equivalence of two pairs of quadratic forms, and we could check the equivalence of an arbitrarily high number. We call the corresponding search problem the Quadratic Forms Simultaneous Equivalence (QFSE) problem.

12.0.4 Related Problems

The most well-known related problem is probably Graph-Isomorphism (GI). Given two graphs G_1 and G_2, the problem is to decide whether G_1 is equal to G_2, up to a relabelling of the vertices. This particular problem is special in many aspects. After more than 40 years of research there is still no wost-case polynomial time algorithm for GI. On the other hand, GI is unlikely to be **NP**-complete, since this would make the polynomial hierarchy collapse.

A related problem is the following: suppose you have oracle access to two boolean functions $f, g : \{0,1\}^n \to \{0,1\}$. They are said to be *isomorphic* if they are equal up to a permutation of the inputs. How many queries to the oracles are required in the worst case to decide if f and g are isomorphic? Alon and Blais show in [AB10] that the query complexity of the problem is $\Theta\left(q^{n/2}\right)$. They also show that if f is known, then the complexity of the problem drops to $\Theta(n)$ for almost all functions f.

12.1 Cryptographic Usage

PLE has been (quite badly) called the "Isomorphism of Polynomials" (IP) problem by Patarin in 1996 [Pat96b] and has received some attention from the cryptographic community. The hardness of the problem indeed underlied the security of several quadratic multivariate trapdoor one-way function. In addition, Patarin proposed to build an identification scheme out of it.

12.1.1 Building Trapdoor-One-Way Function

During the blossom of multivariate cryptography, many Trapdoor One-Way Functions were proposed, based on the following idea: the public-key is a *quadratic map*, and evaluating the trapdoor one-way function essentially amounts to evaluate the quadratic map. The intuitive security argument is that inverting arbitrary quadratic maps (*i.e.*, solving an instance of MQ) is well-known to be **NP**-complete (cf. chapter 10). The trapdoor is that these quadratic maps are built with a special structure allowing the owner of the secret key to invert them. Usually, the idea was to choose an easily-invertible internal quadratic map $\mathbf{f}$ as well as two random invertible matrices S and T, and to "obfuscate" the easy-to-invert $\mathbf{f}$ by composing it with S and T. One would then hope that $\mathbf{PK} = T \circ \mathbf{f} \circ S$ looks random, and therefore should be as hard-to-invert as random quadratic maps. The public-key is then an "obscure representation" of some special, easily-invertible function $\mathbf{f}$.

After breaking C* [Pat95], and while introducing HFE [Pat96b], Patarin brought the attention of the crypto community to QMLE as a hardness assumption of its own, on the hardness of which other schemes could be based. The underlying reasoning was that while C* had succumbed to a *decryption* attack, it had so far resisted all attempts to mount a *key-recovery* attack. A successful key-recovery attack would implicitly find the solution of an instance of QMLE. This seemed hard, so why not build a new hardness assumption out of it? More generally, if the public-key is formed by the composition: $\mathbf{PK} = T \circ \mathbf{f} \circ S$ and if $\mathbf{f}$ is public, then recovering the secret key amounts to solving an instance of QMLE.

Subsequent multivariate public-key schemes relying on the hardness of QMLE include the Hidden Matrix (HM) encryption scheme [PGC98a], the SFLASH [PCG01a] signature scheme, the traceable block cipher of Billet and Gilbert [BG03], the Tractable Rational Map Signatures (TRMS) [WHL+05], the ℓ-IC signature scheme [DWY07], the Multivariate Quadratic Quasigroup encryption scheme (MQQ) [GMK08], as well as the SQUARE [CBD+09] and SQUARE-Vinegar [BCD08] encryption schemes. We do not wish to waste time and space by discussing the superabundant and mostly uninteresting variants of the Tame Transformation Signature scheme (TTS) or of the Stepwise Triangular System (STS).

It is worth remarking that *all* the schemes relying on the hardness of QLME have been broken: a key-recovery attack against C* can be found in [FMRS08], HM is broken in [FJPT10], a decryption attack against SFLASH is given in [DFS07, DFSS07], and a key-recovery attack can be found in [MR10]. The traceable block cipher is broken in [FMRS08], an early version of TRMS is broken in [JKJMR05], and a revised version is broken in [BFP08], the ℓ-IC signature are broken in [FMRPS08], MQQ is broken in [MDBW09], while SQUARE and SQUARE-Vinegar are broken in [BMR09].

We are led to believe that basing the security of a multivariate public-key primitive on the hardness of QMLE seems unsound *per se*. In fact, a more detailed study of the abundant literature mentioned above shows that in all these cases, the corresponding QMLE instances that were broken had a special structure:

indeed, the problem was to find S and T such that $\mathbf{PK} = T \circ \mathbf{f} \circ S$, where $\mathbf{f}$ had to be *easily invertible* by the legitimate user. For this reason, the internal map $\mathbf{f}$ could not be arbitrary, and the adversaries were facing special, easier, instances of QMLE.

12.1.2 Revisiting a Venerable Identification Scheme

In the same paper in which he introduced QMLE (we mean [Pat96b]), Patarin also suggested to base an identification scheme on the hardness of *arbitrarily chosen* instances of QMLE. This is interesting, because we may avoid the structured instances that turned out to be weak.

In an identification scheme, a *prover*, who has generated a pair of private/public keys (PK, SK), wants to prove her identity to a verifier who knows PK. In fact the provers wants to prove that she knows SK, but without revealing any information about SK to the verifier, or to anybody else. This is essentially a zero-knowledge proof of knowledge.

Zero-Knowledge proofs were introduced in 1985 by Goldwasser, Micali and Rackoff in [GMR85]. The next year, two notable and now-classical identification schemes appeared, making use of zero-knowledge proofs. Fiat and Shamir [FS86] proposed to use the hardness of Quadratic-Residuosity, while Goldreich, Micali and Wigderson [GMW86] built an elegant zero-knowledge proof system for GI and used it to build their identification scheme. The latter is illustrated by Algorithm 12.1. It is straightforwardly zero-knowledge: Peggy only reveals ρ (a random permutation), or $\rho \circ \sigma^{-1}$ (the composition of the secret σ with a random permutation, *i.e.*, a random permutation). If she actually knows σ, then Peggy is able to convince Victor in all cases. If Isabelle (the Impostor), who only knows G_1 and G_2 but not σ tries to play the game with Victor, she fails with probability 1/2. If she chooses a permutation π and set $G_3 = \pi(G_1)$, she succeeds in convincing Victor if he picks tails (because she can just reveal π), and she fails if Victor picks heads (because she does not know the isomorphism between G_2 and G_3). Symmetrically, if Isabelle sets $G_3 = \pi(G_2)$, she succeeds in convincing Victor if he picks heads (she reveals π), but she fails if he picks tails (because she does not know the isomorphism between G_1 and G_3). Victor is very playful and asks her partner to play with him again and again, so that the probablity that Isabelle succeeds in convinving him that she is in fact Peggy becomes exponentially small.

Algorithm 12.1 Identification scheme based on Graph Isomorphism

1. Peggy (the Prover) generates a graph G_1, chooses a random vertex-permutation σ, and computes $G_2 = \sigma(G_1)$. She publishes (G_1, G_2) and keeps σ secret.
2. Victor (the Verifier) knows G_1 and G_2 (he found them in a public directory), and he wants Peggy to prove him that she knows σ.
3. Peggy chooses a random vertex-permutation ρ and computes $G_3 = \rho(G_1)$. She sends G_3 to Victor.
4. Victor flips a coin. If heads (resp. tails) come out, he asks Peggy to reveal ρ (resp. $\rho \circ \sigma^{-1}$).
5. Peggy kindly complies.
6. Victor checks that $G_3 = \rho(G_1)$ (resp. $G_3 = \rho \circ \sigma^{-1}(G_2)$).

On the (slightly more) practical side, the difficulty in coming up with a concrete realization of the GI-based identification scheme is that most instances of GI are extremely easy to solve. For instance, there is an algorithm that solves random instances in expected linear time (see the survey [For96] for more details). So far, this is not so bad: random integers are easy to factor, but RSA is still there, because we know how to generate hard instances of the factorization problem. With GI the situation is much worse, because generating hard instances of GI is extremely delicate: many powerful heuristics have to be fooled simultaneously.

Patarin's idea was to simply replace GI by QMLE in Algorithm 12.1. The main reason one would want to do this is that it seems easier to generate hard instances of QMLE — choosing them uniformly at random seems essentially sufficient. Under the assumption that random QMLE is hard, then the resulting identification scheme could even reach decent key sizes.

In order to further improve performance and key size, QMLE could even be replaced by QFSE. The resulting scheme is then competitive with the state of the art in terms of key sizes, as discussed below. In terms of security, it seems that using QFSE instead of GI cannot worsen the (asymptotic) security level: Courtois, Patarin and Goubin indeed established that QFSE is GI-*hard*. They proved that any instance of GI could be transformed into a (polynomially bigger) instance of QFSE, the solution of which would reveal the solution of the original GI instance. *Ergo*, replacing GI with QFSE could in theory only lead to a security improvement, while making hard instance generation much easier. A drawback is that it has also been shown that QFSE was also unlikely to be **NP**-complete, as it would also make the polynomial hierarchy collapse [PGC98b, FP06].

Comparison With Other Identification Schemes. Many other identification schemes appeared after the aforementioned works of Fiat-Shamir and Goldreich-Micali-Wigderson. While some of them rely on the hardness of number-theoretic assumptions, some cryptographers took a different line of research, and tried to design identification scheme from different computational assumptions, not relying on number theory, but instead on the **NP**-hardness of some specific combinatorial problems. We highlight that Patarin's QFSE-based identification scheme does not look too shabby in terms of key size.

One of the very-first combinatorial identification scheme was proposed by Shamir [Sha89], and relied on the hardness of the Permuted Kernel Problem (PKP). Later on, Stern proposed in [Ste93] a scheme based on the intractability of Syndrome Decoding (SD), a hard problem from coding theory, and in [Ste94] a scheme based on the intractability of Constrained Linear Equations (CLE). Pointcheval [Poi95] proposed a scheme related to the hardness of the Perceptron problem, originating from the area of learning theory. Very recently, Sakumoto, Shirai and Hiwatari [SSH11] proposed a clever scheme based on the average-case hardness of MQ. All these problems are **NP**-complete (as opposed to QFSE). The designers proposed practical parameters, aiming for a security level of 2^{64} or more, which are summarized in table 12.1. It must be noted that several cryptanalytic results may apply to these parameters [BCCG92, Geo92, PC93, KM99].

In all these schemes, it is required that all users share a public common string, usually describing an instance of the hard problem at hand. For instance, in number-theoretic problems, the description of the curve, or of the group over which a discrete logarithm problem is considered is a common public information. While this information is not a "key" *stricto sensu*, it must nevertheless be stored by the prover and by the verifier, leading to higher memory requirements. However, in some case it can be chosen randomly, or generated online from a small seed using a PRNG.

Scheme	Common String	Public Key	Secret Key
PKP	2048	256	374
	7992	512	808
SD	131 072	256	512
	524 288	512	1024
CLE	3600	80	80
	3600	96	96
Perceptron	10807	144	117
MQ	265'680	80	80
QFSE	0	256	272

Table 12.1: Key sizes in bits corresponding to practical parameters proposed in [Sha89, Ste93, Ste94, Poi95, Pat96c, SSH11] in order to obtain a security level of at least 2^{64}.

On the contrary, the QFSE-based identification scheme proposed by Patarin in [Pat96b, Pat96c] does not need the prover and the verifier to share additional information (except maybe the description of the finite field, which is very small). The PKP and SD schemes lead to bigger keys than QFSE, while the Perceptron scheme leads to comparable key-sizes, and CLE yields smaller keys, if we neglect the additional memory requirement imposed by the common string shared between all the participants.

Additionnaly, the QFSE-based identification scheme does not make use of either hash functions or commitment schemes. This is in strong contrast with all the other proposals.

As a final note, let us mention that Lyubashevsky recently proposed in [Lyu08] to build an identification scheme using the hardness of lattice problems, but did not propose concrete parameters.

12.2 Taxonomy

A complete study of Polynomial Linear Equivalence problems is difficult, because many characteristics of the instances influence the performance of the various algorithms. We have already identified three relevant subproblems: QFSE, CFE and QMLE. In this section we try to refine this classification, and to isolate the relevant (sub-)subproblems. Ideally, this classification would help understand and express the performance of corresponding algorithms. Bits and pieces of this classification were done by the authors of the various existing PLE algorithms, but no complete picture was available up to date.

Presence of T. While it may sound kind of obvious, QMLE and QFSE are different problems. Of course, QMLE algorithms should apply "as-is" to QFSE, but this is rarely optimal.

Number of Polynomials. When we introduced QMLE at the beginning of this chapter, it was more or less implicit that we considered quadratic maps over $(\mathbb{F}_q)^n$, *i.e.*, vectors of n quadratic polynomials. However, when we introduced QFSE, we explicitly stated that we could deal with an arbitrary number of quadratic forms. Let us therefore denote by u the number of polynomials at hand. Key-recovery problems for encryption schemes often yield instances of QMLE with $u = n$ ($u < n$ is impossible, otherwise the ciphertext would contain less information than the plaintext, and $u > n$ is possible but bad in terms of bandwidth). For this reason (amongst others), QMLE has exclusively been studied with $u = n$. We also believe the other cases to be *much harder*. At the very least, it seems that most QMLE algorithms break down when $u \neq n$, and cannot be fixed in a simple way. Informally, the instance contains $\mathcal{O}\left(u \cdot n^2\right)$ units of information, and the solution that has to be recovered contains $\mathcal{O}\left(u^2 + n^2\right)$ units of information, so the problem is maximally overdetermined when $u = n$. The situation is essentially the same for QFSE, except that there is no cryptographic motivation pushing to study in particular the case $u = n$. In fact, the exact opposite is true: in the identification scheme, choosing u as small as possible reduces the key size and the communication complexity. It also apparently makes the scheme more secure: we will show in chapter 13 that the existing algorithms have an exponential running-time when $u \lll n$. Note that $u = 2$ is the lowest admissible value, because we already discussed that the equivalence of single quadratic forms is easy. If $\mathbf{a}$ and $\mathbf{b}$ are of higher degree, then it would make sense to consider $u = 1$ though.

Bijectivity of S and T. The definition of PLE problems makes it very clear that S and T should be invertible. If we drop this requirement, we face a completely different problem, also somewhat unfortunately termed the Morphism of Polynomial (MP) problem by [PGC98b]. This other problem is much harder, as it easy to show that it is **NP**-complete. Interestingly enough, finding the minimum number of field multiplications required to multiply two $n \times n$ matrices reduces to an instance of MP. It is known that this number is 7 for $n = 2$, but it is still unknown for $n \geq 3$. See [PGC98b] for more details regarding MP.

Degree. Certainly the most important parameter of PLE instances is the *degree* of the polynomial maps. Because of their cryptographic applications, most of our work is devoted to the quadratic case. However, since Patarin proposed to use cubic polynomials in his identification scheme, we investigate this particular matter in chapter 16.

The way in which the degree affects the complexity of solving instances depends of the algorithm used. Some algorithms, including the ones we present in chapters 17 and 18, only work on quadratic instances. The "to-n-fro" algorithm, presented in §13.1 is essentially degree-independent, while the complexity of the Gröbner-basis approach presented in §13.2 increases with the degree.

Homogeneous vs Inhomogeneous. The second most important characteristic of a PLE instance is whether the polynomials are *homogeneous* or not. This was (to the best of our knowledge) first observed in [dVP03], in which it was shown that (linear) inhomogeneous QFSE with $u = n$ was solvable in polynomial time. Homogeneous instances are *always* harder (sometimes exponentially). The following result (vastly plagiarized from [FP06]) expresses the fundamental reason behind this.

Lemma 12.1. *If $T \circ \mathbf{b} = \mathbf{a} \circ S$, with $S, T \in \mathrm{GL}_n(\mathbb{F}_q)$, then for all $i \in \mathbb{N}$, we have: $T \circ \mathbf{b}^{(i)} = \mathbf{a}^{(i)} \circ S$.*

In more concrete terms, lemma 12.1 states that when dealing with an *inhomogeneous* instance, S and T are simultaneously the solutions of homogeneous instances corresponding to each lower-degree homogeneous component. What becomes very interesting is then to look at the homogeneous components of degree zero and one. We find:

$$\begin{aligned} T \times \mathbf{b}^{(1)} &= \mathbf{a}^{(1)} \times S, \\ T \times \mathbf{b}^{(0)} &= \mathbf{a}^{(0)}. \end{aligned}$$

These two relations degenerate to $0 = 0$ in the homogeneous case, but otherwise reveal an important amount of information on S and T, and play a crucial role in the new algorithms we present in chapter 17. Prior to this work, Faugère and Perret proposed in [FP06] the first algorithm that (empirically) runs in polynomial time on inhomogeneous instances of any degree.

In general, we could summarize the difference between the homogeneous and inhomogeneous cases by saying that "there is no free lunch [1] in the homogeneous case". As shown above, there are various ways to obtain "free" information on S and T in the inhomogeneous case, but none of these are applicable to the homogeneous case. To emphasize the difference between the homogeneous and the inhomogeneous case of the QMLE problem, we distinguish between Inhomogeneous QMLE (IQMLE) and the standard (homogeneous) QMLE.

1. in reference to [WM97]

Linear vs Affine. We stress that it is only possible to take advantage of the inhomogeneity of an instance if S and T are *linear* (as opposed to *affine*) bijections. While bringing in affine transformations is not in the spirit of the generalization of matrix equivalence we started with, it is very natural if the purpose of S and T is to obfuscate the representation of some quadratic map (in an encryption/signature context). In the presence of affine transformation, equation (12.1) becomes:

$$T \circ \mathbf{b}(\mathbf{x}) + \mathbf{d} = \mathbf{a}(S \cdot \mathbf{x} + \mathbf{c}), \qquad \text{with} \quad S, T \in \mathrm{GL}_n(\mathbb{F}_q), \quad \text{and} \quad \mathbf{c}, \mathbf{d} \in (\mathbb{F}_q)^n \tag{12.2}$$

Affine instances were thought to be harder, and the strategy proposed in [PGC98b] was to guess $\mathbf{c}$ and $\mathbf{d}$ before doing anything else. It is in fact possible to do better, and the starting point is to adapt lemma 12.1 to the affine case. The result is unfortunately less appealing.

Lemma 12.2. *If $T \circ \mathbf{b}(\mathbf{x}) + \mathbf{d} = \mathbf{a}(S \cdot \mathbf{x} + \mathbf{c})$, with $S, T \in \mathrm{GL}_n(\mathbb{F}_q)$ and $\mathbf{c}, \mathbf{d} \in (\mathbb{F}_q)^n$, then for all $k \geq 1$ we have*

$$\mathbf{b}^{(k)} = T \circ \left(\mathbf{a} + \frac{\partial \mathbf{a}}{\partial \mathbf{c}}\right)^{(k)} \circ S.$$

In particular, if d is the degree of $\mathbf{a}$ and $\mathbf{b}$, then $\mathbf{b}^{(d)} = T \circ \mathbf{a}^{(d)} \circ S$.

Proof. Starting from equation (12.2) we have:

$$T \circ \mathbf{b}(\mathbf{x}) + \mathbf{d} = \mathbf{a}(S \cdot \mathbf{x} + \mathbf{c}) = \mathbf{a}(S \cdot \mathbf{x}) + \frac{\partial \mathbf{a}}{\partial \mathbf{c}}(S \cdot \mathbf{x})$$

The first statement of the lemma follows from applying lemma 12.1 to the leftmost and rightmost sides of this equation. The second statement follows from the fact that $\deg \frac{\partial \mathbf{a}}{\partial \mathbf{c}} = \deg \mathbf{a} - 1$. □

Lemma 12.2 has three important consequences.

1. When facing an affine instance, the linear part of the solution (S and T) is amongst the solutions of the *linear homogeneous* instance formed by considering only the homogeneous component of highest degree. Thus, $\mathbf{c}$ and $\mathbf{d}$ do not need to be guessed beforehand if the homogeneous problem can be cracked down.
2. If $\mathbf{a}$ and $\mathbf{b}$ are quadratic and homogeneous, then the value of $\mathbf{c}$ and $\mathbf{d}$ can be retrieved efficiently, as observed in [GSB01]. Because $\mathbf{b}$ is homogeneous, then $\mathbf{b}^{(1)}$ is identically zero. This means that $\left(\frac{\partial \mathbf{a}}{\partial \mathbf{c}}\right)^{(1)}$ is identically zero as well. Because $\mathbf{a}$ is quadratic, the function that maps $\mathbf{c}$ to $\frac{\partial \mathbf{a}}{\partial \mathbf{c}}$ is linear. Thus $\frac{\partial \mathbf{a}}{\partial \mathbf{c}} = 0$ in fact yields n^2 linear equations on the n coordinates of $\mathbf{c}$, and $\mathbf{c}$ can be retrieved in polynomial-time by straightforward linear algebra. Once $\mathbf{c}$ has been recovered, $\mathbf{d}$ comes for free.
3. The same reasoning applies to quadratic affine inhomogeneous instances, but S and T must be known before $\mathbf{c}$ and $\mathbf{d}$ can be recovered.

To summarize, when dealing with affine instances, the procedure is the following: solve the linear homogeneous instance obtained by looking only at the homogeneous component of highest degree. Once S and T have been retrieved, recover $\mathbf{c}$ and $\mathbf{d}$ in polynomial time.

This shows that the linear homogeneous case is *complete*, in the sense that if we know how to solve it efficiently, then we will know how to deal with all the other cases. This was already observed in the context of QFSE by Perret in [Per05]. So, the study of QMLE in fact boils down to two cases: the (easy) linear inhomogeneous case, and the (difficult) linear homogeneous case.

Parity of q and n. In some algorithms, the *parity* of the characteristic of the field plays a role. This happens for essentially two reasons: firstly, $\mathrm{D}_{\mathbf{x}}$ always vanish on $\mathbf{x}$ when q is even. Secondly, in characteristic two, the theory of quadratic forms comes in a specific "flavor". For instance, the sets of symmetric and skew-symmetric matrices nearly coincide, with funny consequences.

The algorithms often have to be analyed differently in characteristic two and in odd characteristic, and some of their properties are different (it is often trickier in characteristic two). In addition, in characteristic two, the parity of n sometimes also plays a role, notably because when n is odd, symmetric matrices with zeroes on the diagonal are always singular. This particular problem manifests itself in the Jacobian algorithm for QFSE presented below, and in the QFSE algorithm we present in chapter 15.

Bijectivity of $\mathbf{a}$ and $\mathbf{b}$, and Other Properties. There are some pre-existing algorithms where the fact that $\mathbf{a}$ and $\mathbf{b}$ are *bijective* quadratic maps makes the algorithm faster and the analysis simpler. In the same way, some specific algorithms may depend on particular properties of the quadratic maps. When it is the case, it is important that these properties are clearly stated.

Challenge	n	q	Public Key	Private Key	Broken ?
A_1	16	2	272 bits	256 bits	by us, in 1s
B_1	6	16	168 bits	144 bits	[GMS03, FP06]†

(a) QFSE challenges (2 quadratic polynomials, affine transform)

Challenge	n	q	Public Key	Private Key	Broken ?
C_1	16	2	816 bits	256 bits	by us, in 1 month
D_1	6	16	224 bits	144 bits	by us, in 411s

(b) CFE challenges (1 cubic form, affine transform)

Challenge	n	q	Public Key	Private Key
E_1	64	2	133 184 bits	8192 bits
F_1	16	16	8768 bits	2048 bits
G_1	8	256	2368 bits	1024 bits

(c) QMLE challenges (quadratic,inhomogeneous, affine, $u = n$)

Table 12.2: Patarin's challenges [Pat96c], along with the corresponding key-sizes for the identification scheme of section 12.1.2.

12.3 Challenges

We have identified three pre-existing algorithms for QFSE [Per05, dVP03, GMS03], and two algorithms for QMLE [PGC98b, FP06]. We survey all of them in chapter 13. Unfortunately though, comparing these algorithms based on their theoretical features is close to impossible. Some may fail with a given probability, while the others always succeed, but with uncertain complexity. Their inventors sometimes identified classes of instances that their algorithm solve efficiently (*i.e.*, in polynomial-time), but it is not always the case, and more often than not is just an empirical observation. Besides, these classes do not really intersect.

The authors of PLE algorithms thus measured the efficiency of their techniques *in practice*, by trying to break wider and wider ranges of parameters (except for the authors of [PGC98b] who did not measure the efficiency of their algorithms in practice). Patarin did the right thing by proposing concrete challenges in [Pat96c]. He did not provide actual instances, but he specified several parameter sets that he believed were more and more secure. These challenges can be used as milestones to measure the progress accomplished since their introduction. The corresponding parameter sets are shown in Table 12.2.

A few words of commentary. The QFSE/CFE challenges use the smallest number of polynomials that makes sense, which seems to make the problem as hard as is gets. The QMLE challenges have *affine* transformations S and T, which makes them belong to the hardest class of the problem. They were designed to require a workload of at least $\mathcal{O}(q^n)$ operations. To illustrate the difference in behavior between homogeneous and inhomogeneous instances, we will consider linear inhomogeneous variants of the challenges, and we will denote them by $\widetilde{A_1}$, $\widetilde{B_1}$, ...

These challenges were apparently designed, 15 years ago, to provide a security level of roughly 2^{64} elementary operations. According to Moore's law, the available computing power has been multiplied by about 1000 since that time. These challenge are therefore *legacy challenges* (let us denote them by A_1, B_1, etc). It would make sense to increase n by at least 25%, to form *updated challenges* $(A_2, B_2, \dots)$, and to even double n to form *safer challenges* $(A_3, B_3, \dots)$.

Some of these challenges could be broken by the algorithms published prior to this work, however, the authors of the corresponding papers did not realize it, or did not advertize it. These challenges are marked with a dagger (†) in the table.

Chapter 13

Revisiting Prior Algorithms for PLE

We describe and discuss the pre-existing algorithms for all variants of PLE.

In this chapter we describe and discuss the features of five pre-existing algorithms. Essentially two non-trivial algorithms have been proposed so far for QMLE: the "To-and-Fro" approach of Courtois, Goubin and Patarin [PGC98b] on the one hand, and the "Gröbner Basis" approach of Faugère and Perret [FP06] on the other hand. While the former is historically the first algorithm to improve on exhaustive search, its complexity is greater than q^n. The authors of the latter conjectured, without giving any supporting evidence, that their technique is subexponential, and observed that it empirically terminates in polynomial on random inhomogeneous instances, thereby providing the first concrete evidence that these instances were easy.

To our knowledge, the first algorithm dedicated to QFSE was given in 2003 by Geiselmann, Meier and Steinwandt [GMS03], and because it needs a name, we call it the "columnwise sieve". Soon after, Levy-dit-Vehel and Perret [dVP03] suggested to replace the "exhaustive search" component of the columnwise sieve by the computation of a Gröbner basis, resulting in the "algebraic columnwise sieve". Both algorithms are exponential by nature, and their complexity was not even vaguely known. A breakthrough took place in 2005, when Perret [Per05] proposed a new algorithm, that we call the "Jacobian" algorithm because its distinctive feature is the use of Jacobian matrices. This algorithm is polynomial when the number u of polynomials in $\mathbf{a}$ and $\mathbf{b}$ is equal to the number of variables n. Recall that in the context of QFSE and CFE, $\mathbf{a}$ and $\mathbf{b}$ have u polynomials, and that we do *not* enforce $u = n$ *a priori*.

While we originally intended to do a quick survey of the state of the art, doing it turned out to be surprisingly challenging and disappointing. The algorithms were often quite vaguely described, with room for interpretation and without pseudo-code. In addition, their success probability, and/or their running time, were often not discussed. We thus give our own complexity analysis for four of the five algorithms under review.

13.1 Going To-and-Fro

Informal Description. The first non-trivial algorithm for QMLE, given in 1998 by Courtois, Goubin and Patarin [PGC98b]. It has been independently rediscovered five years later by Biryukov, De Cannière, Braeken and Preneel [BCBP03], under the form of an algorithm to test the linear equivalence of S-boxes.

The to-n-fro algorithm is based on the following informal observation: if we know the value of S on a few linearly independent points, then we can efficiently "amplify"our knowlegde of S into an exponentially bigger knowledge of T. Let $F = \langle f_1, \ldots, f_r \rangle$ be a subspace of $(\mathbb{F}_q)^n$ over which S is known. Then for any $\mathbf{x} \in F$, it is no big surprise that:

$$T \cdot \mathbf{b}(\mathbf{x}) = \mathbf{a}(S \cdot \mathbf{x}).$$

This reveals T over the q^r points forming the image of F by $\mathbf{b}$. Since $\mathbf{b}$ is by definition non-linear, we may legitimately hope to discover T on a subspace of $(\mathbb{F}_q)^n$ of dimension greater than r.

The second idea underlying the algorithm is that knowledge of T can be "transferred" back into knowledge of S, yet doing so is very expensive. Following its inventors, we will assume that $\mathbf{a}$ and $\mathbf{b}$ are *bijective* quadratic maps (we will try to drop this assumption later on). The point is that for all $\mathbf{x} \in (\mathbb{F}_q)^n$ we have:

$$\mathbf{a}^{\langle -1 \rangle}(T \cdot \mathbf{x}) = S\left(\mathbf{b}^{\langle -1 \rangle}(\mathbf{x})\right) \tag{13.1}$$

So, from the equality $\mathbf{y} = T \cdot \mathbf{x}$, involving only T, we deduce the equality $\mathbf{a}^{\langle -1 \rangle}(\mathbf{y}) = S\left(\mathbf{b}^{\langle -1 \rangle}(\mathbf{x})\right)$, involving only S. Solving two instances of the MQ problem thus reveals the image of S on a point. On some points of G, the preimage will not belong to F, and our knowledge of S will then be increased, allowing us to re-iterate the procedure, until S and T have been completely determined.

More In-Depth Study. The inventors of this algorithm did not give any in-depth complexity analysis, and did not give pseudo-code. Instead, they illustrated their algorithm on a toy example. Algorithm 13.1 shows our attempt at writing a rigorous description.

Algorithm 13.1 Bijective to-n-fro algorithm for QMLE.

```
1:  function BIJECTIVE-TO-N-FRO(a, b, F)                ▷ S is initially known on F
2:      repeat
3:          for all x ∈ F do                                              ▷ Enlarge G
4:              G ← G + ⟨b(x)⟩
5:              remember that T · b(x) = a(S · x)
6:          end for
7:          if dim G = dim F then report failure
8:          for all x ∈ G − b(F) do                                       ▷ Enlarge F
9:              u ← b^⟨−1⟩(x)
10:             v ← a^⟨−1⟩(T · x)
11:             F ← F + ⟨u⟩
12:             remember that S · u = v
13:         end for
14:     until dim F = n
15:     Fully determine T
16:     return S and T
17: end function
```

Looking at the pseudo-code, we realize that the algorithm may fail. In that case, it must be restarted with an enlarged F (which means that the image of S on one more point must be found). The algorithm fails on line 7 if $\mathbf{b}(F)$ is a vector space. Note that this will always happen if $\dim F = 1$, $q = 2$ and the polynomials are homogeneous. Under the same conditions, if $F = \langle \mathbf{u}, \mathbf{v} \rangle$, $q = 2$ then failure would mean that $\mathbf{b}(\mathbf{u}+\mathbf{v}) = \mathbf{b}(\mathbf{u})+\mathbf{b}(\mathbf{v})$. If $\mathbf{b}$ were a random permutation we could assert that this happens with probability 2^{-n} (over the random choice of $\mathbf{b}$). We can in fact check that this is true even though $\mathbf{b}$ is a random bijective quadratic map, but we skip the details.

It therefore seems clear that the "Bijective to-n-fro" will find the solution with high probability given one (resp. two) known relation in F when q is greater than two (resp. equal two). In both cases, $2n$ instances of the MQ problem will have to be solved.

Giving Up on Bijectivity. It seems natural to try to drop the bijectivity requirement, as a random QMLE instance is not bijective with overwhelming probability. This is in fact surprisingly challenging, and there are many possible ways to adapt the bijective algorithm. The main issue is that there may now be zero, one, or more than one preimages. The simplest option is probably to ignore the cases when there is not a single preimage. This however creates a non-negligible possibility of failure: for instance, if $q = 2$, $F = \langle \mathbf{x}, \mathbf{y} \rangle$ and the polynomials are homogeneous, then if everything goes well we will have $G = \langle \mathbf{b}(\mathbf{x}), \mathbf{b}(\mathbf{y}), \mathbf{b}(\mathbf{x}+\mathbf{y}) \rangle$, and $G - \mathbf{b}(F)$ contains 4 elements. We know the distribution of the number of preimages of $\mathbf{b}$ from §10.2, and the probability that none of these 4 elements has a single preimage is $(1 - 1/e)^4 \approx 16\%$. Algorithm 13.2 shows our attempt at getting rid of the bijectivity constraint.

The case where there are more than one preimage could be dealt with by "guessing" the right permutation of the preimages and going on, but this then yields a more complicated backtracking algorithm, and it is unclear that the result will be more efficient. In any case, the number of MQ instances that we need to solve in order to find a point of G having a single preimage follows a geometric distribution of parameter $1/e$. Therefore, we expect, all-in-all to have to solve $3.7n$ instances of the MQ problem.

To conclude, we note that the method generally has problems with non-bijective instances of QMLE, but its failure probability goes down when the size of the field grows. Lastly, it is inherently exponential, as it assumes the possibility to invert the quadratic maps. Moreover, since the image of S must be known on at least one point (two when $q = 2$), then the full algorithm has a complexity of order $\mathcal{O}\left(n \cdot q^{2n}\right)$ when $q > 2$, and $\mathcal{O}\left(n \cdot q^{3n}\right)$ when $q = 2$ (in conformance with the claims of Courtois, Goubin and Patarin).

13.2 The Gröbner-Basis Algorithm

Let us take a look back at equation (12.1):

$$T \circ \mathbf{b} = \mathbf{a} \circ S$$

Algorithm 13.2 General (non-bijective) to-n-fro algorithm for QMLE.

```
1: function General-To-n-Fro(a, b, F)                    ▷ S is initially known on F
2:     repeat
3:         for all x ∈ F do                                ▷ Enlarge G
4:             G ← G + ⟨b(x)⟩
5:             remember that T · b(x) = a(S · x)
6:         end for
7:         if dim G = dim F then report failure
8:         for all x ∈ G − b(F) do                         ▷ Enlarge F
9:             U ← b^⟨−1⟩(x)
10:            if |U| = {u} then
11:                v ← a^⟨−1⟩(T · x)
12:                F ← F + ⟨u⟩
13:                remember that S · u = v
14:            end if
15:        end for
16:    until dim F = n
17:    Fully determine T
18:    return S and T
19: end function
```

This is an equation between two vectors of multivariate polynomials. If we pick a random vector $\mathbf{x} \in (\mathbb{F}_q)^n$, we observe that $T \cdot \mathbf{b}(\mathbf{x}) - \mathbf{a}(S \cdot \mathbf{x}) = 0$. In this equality the coefficients of S and T appear with degree two and one, respectively. We thus have n quadratic equations in $2n^2$ variables. But nothing prevents us to generate a different vector of n quadratic equations in these same variables by choosing a different random vector $\mathbf{x}$. In fact, repeating this process $2n^3$ times would yield $2n^4$ quadratic equations in $2n^2$ unknowns, and it would be possible to solve the resulting system by linearization in time $\mathcal{O}\left(n^6\right)$.

This approach, while very tempting and independently rediscovered many times, in fact fails miserably. The reason is that the number of linearly independent equations that can be generated that way is in fact upper-bounded by $n^2 \cdot (n-1)/2$ in the homogeneous case (this is n times the number of monomials in n variables of degree exactly two), and by $n \cdot (n+2) \cdot (n+1)/2$ in the inhomogeneous case (this is n times the number of monomials in n variables of degree at most two), as shown in [FP06].

Directly Obtaining The Interesting Equations. Faugère and Perret demonstrated that it is easy to directly acquire a basis of the vector space spanned by these equations. If we rewrite (12.1) into $T \circ \mathbf{b} - \mathbf{a} \circ S = 0$, then the multivariate polynomials on the left are identically zero, and therefore the coefficient of each monomial must be zero. These coefficients are themselves quadratic polynomials in the entries of S and T. So, this coefficient-wise identification yields one quadratic equation in the entries of S and T for each occurrence of each monomial in $\mathbf{a}$ or $\mathbf{b}$, and this makes precisely $n \cdot \binom{n+2}{2}$ equations. So, a possible QMLE algorithm consists in:

1. Acquiring the $n \cdot \binom{n+2}{2}$ quadratic equations in S and T resulting from (12.1).
2. Computing a Gröbner basis of the corresponding ideal.
3. Actually finding all the solutions of these equations using the Gröbner basis.

Properties of the Equations. First of all, the system of equations obtained that way is overdetermined. As a consequence, even without adding the field equations we find that the existence of solutions in the algebraic closure besides those in $\mathbb{F}_q$ is extremely unlikely, which suggests that the properties of this QMLE algorithm are relatively independent from the size of the field. This concurs to make this remarkable method unique.

Additionnaly, when $\mathbf{a}$ and $\mathbf{b}$ are homogeneous, then $\mathcal{I}$ is a *bi-homogeneous* ideal of positive dimension (most likely one). The coefficients of T only occur linearly, while the coefficients of S appear in monomials having the same degree as $\mathbf{a}$, thus the ideal is bi-homogeneous. Some specialized Gröbner basis algorithm could then possibly be applied [FDS11].

When there is a solution, there is in fact an infinite family of solutions in the algebraic closure of the field, because if (S, T) is a solution, then $(\lambda \cdot S, \lambda^2 \cdot T)$ is also a solution. This justifies in passing that the ideal is of positive dimension.

13.2.1 Theoretical and Practical Complexity

Faugère and Perrey empirically observed that when applied to quadratic random inhomogeneous instances, the degree of the polynomials handled by the Gröbner basis computation was always less than or equal to three. This means, according to the discussion in §10.4, that the complexity of computing the Gröbner basis is $\mathcal{O}\left(n^9\right)$. They also observed that in this particular case, the running time of the whole procedure was dominated by the cost of *constructing* the equations, and not by that of their resolution! This motivates a little more in-depth investigation of the matter.

Generating the Equations. A first observation is that if we have to generate $\mathcal{O}\left(n^3\right)$ quadratic equations in $\mathcal{O}\left(n^2\right)$ variables, then our system of equation can have up to $\mathcal{O}\left(n^7\right)$ non-zero coefficients in the worst case. Were we to use a dense representation to store our equations, the time needed to generate them would then obviously be $\Omega\left(n^7\right)$. The only way this bound could be beaten would be to exhibit some kind of pattern in the equations and to use a sparse representation, as shown below.

We will focus, for the sake of simplicity, on the homogeneous case. Suppose then that we are given $\mathbf{a}$ (resp $\mathbf{b}$) as a collection of n quadratic forms, each represented by an upper-triangular $n \times n$ matrix over $\mathbb{F}_q$ denoted by $A^{[i]}$ (resp. $B^{[i]}$). If we look at the k-th coordinate of equation (12.1), we see:

$$T_{k\bullet} \cdot \mathbf{b}(\mathbf{x}) = \mathbf{a}_k(S \cdot \mathbf{x})$$

It is not incredibly difficult to work out that this translates as:

$$\sum_{i=1}^{n}\sum_{j=i}^{n}\left(\sum_{\ell=1}^{n} T_{k\ell} \cdot B_{ij}^{[\ell]}\right) \cdot \mathbf{x}_i\mathbf{x}_j = \sum_{i=1}^{n}\sum_{j=1}^{n}\left(\sum_{\ell=1}^{n}\sum_{m=\ell}^{n} S_{\ell i} \cdot A_{\ell m}^{[k]} \cdot S_{mj}\right) \cdot \mathbf{x}_i\mathbf{x}_j$$

Identifying the coefficients of the $\mathbf{x}_i^2$ monomials yields:

$$\sum_{\ell=1}^{n} T_{k\ell} \cdot B_{ii}^{[\ell]} = \sum_{\ell=1}^{n}\sum_{m=\ell}^{n} S_{\ell i} \cdot A_{\ell m}^{[k]} \cdot S_{mi} \tag{13.2}$$

And doing the same with the $\mathbf{x}_i\mathbf{x}_j$ monomials gives:

$$\sum_{\ell=1}^{n} T_{k\ell} \cdot B_{ij}^{[\ell]} = \sum_{\ell=1}^{n}\sum_{m=\ell}^{n} (S_{\ell i} + S_{\ell j}) \cdot A_{\ell m}^{[k]} \cdot (S_{mi} + S_{mj}) \tag{13.3}$$

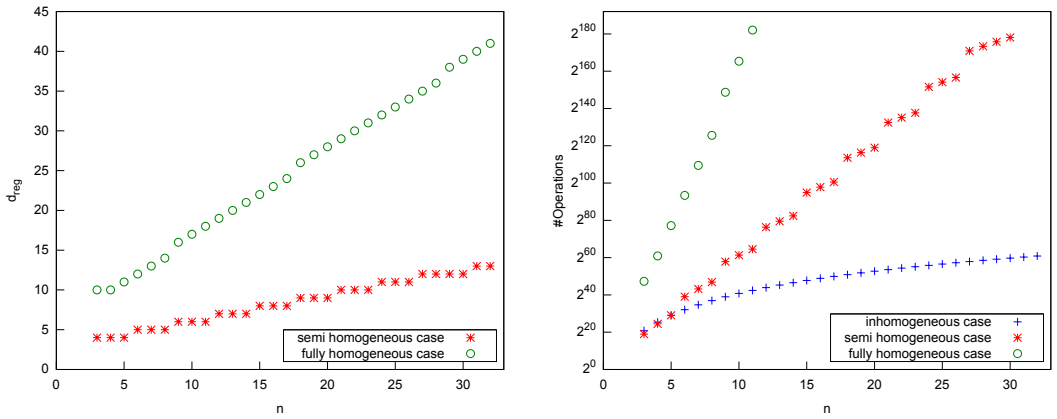
And our system of $n^2(1+n)$ quadratic equations $\mathcal{S}_{\text{quad}}$ is formed by assembling (13.2) and (13.3) for all $1 \leq k \leq n, 1 \leq i \leq n$ and $i < j \leq n$. More interestingly, if we look at an individual equation, we find that only the coefficients of $S_{\bullet i}, S_{\bullet j}$ and $T_{k\bullet}$ occur. Thus the generators of $\mathcal{S}_{\text{quad}}$ are indeed sparse, and the complete set of equations can be represented with $\mathcal{O}\left(n^5\right)$ non-zero coefficients. In addition, these coefficients can be obtained without any arithmetic operations, with just some shuffling around of the coefficients of $\mathbf{a}$ and $\mathbf{b}$.

Solving the Equations. The equations in $\mathcal{S}_{\text{quad}}$ are certainly not random-looking. This can be seen with an informal information-theoretic argument: a random system of $\mathcal{O}\left(n^3\right)$ equations in $\mathcal{O}\left(n^2\right)$ variables cannot be described by less that $\mathcal{O}\left(\log q \cdot n^7\right)$ bits, while $\mathcal{S}_{\text{quad}}$ is completely characterized by only $\mathcal{O}\left(\log q \cdot n^3\right)$ bits (this is the size of a complete description of $\mathbf{a}$ and $\mathbf{b}$).

The authors of [FP06] considered two distinct settings and evaluated their algorithm in both: the *inhomogeneous* case in which random homogeneous components of all degrees are present, and the *semi-homogeneous* case, where only the quadratic and linear homogeneous components are present, but not the constant one. It is also interesting to take a look at the (fully) *homogeneous* case.

Let us assume, for the sake of the argument, that $\mathcal{S}_{\text{quad}}$ is a *semi-regular system* (cf. §10.5). In the homogeneous case, we only have quadratic equations, so we directly compute the degree of regularity using the Hilbert series, assuming the presence of $2n^2$ variables and $n^2(n+1)/2$ equations. In the semi-homogeneous case, we note that there are n^2 linear equations in the system, which amounts to remove n^2 variables. We therefore compute the degree of regularity assuming the presence of n^2 variables and $n^2(n+1)/2$ quadratic equations. In the inhomogeneous case, we assume that the degree of regularity is 3, as observed by Faugère and Perret. These degrees of regularity are shown in Figure 13.1a and the corresponding complexities are shown in Figure 13.1b. In the inhomogeneous case, we find a polynomial complexity, as expected. In the semi-homogeneous case, we find a complexity of order $\mathcal{O}\left(2^{6n}\right)$, and in the (fully) homogeneous case, a complexity of order $\mathcal{O}\left(2^{18n}\right)$.

We note that this directly contradicts [FP06], in which it was conjectured that the Gröbner basis computation would be subexponential in the semi-homogeneous case. Our complexity estimates are established under the debatable assumption that quite structured systems of quadratic equations are semi-regular. Thus, determining the complexity of this algorithm is an essentially open problem.



(a) Degree of regularity (b) Estimated Complexity of GB computation

Figure 13.1: Properties of semi-regular systems with the same number of equations and variables as $\mathcal{S}_{\text{quad}}$.

13.2.2 Experimental Results

> log log log is known to grow to infinity, but it has never been observed doing so...
>
> Don Coppersmith

We could not obtain definitive results on the complexity of the algorithm from a theoretical point of view, but we still have a chance to study it in practice. It would be interesting to compare our estimates with the actual behavior of the algorithm. Getting a complete picture of the practical behavior of the algorithm is nevertheless not very simple, for two reasons: a) many parameters influence the behavior of the algorithm, and b) the time and space complexity quickly becomes so high that only very small values of the parameteres can be considered in practice. This makes it close to impossible to check whether the asymptotic estimates are meaningful. For instance, in the fully-homogeneous case over $\mathbb{F}_{2^8}$, the algorithm terminates in 30s when $n = 5$ but runs for more than 2 weeks when $n = 6$.

We could try to benchmark the performance of the algorithm, but we would in fact vastly be benchmarking a particular implementation of a particular Gröbner basis algorithm, specifically, the implementation of $\mathbf{F_4}$ in the MAGMA [BCP97] computer algebra system. We therefore quote the numbers from [FP06] in Table 13.1.

13.3 The "Columnwise Sieve" Algorithm

In [GMS03] Geiselmann, Meier and Steinwandt presented a nice and simple algorithm for QFSE. Their objective was to tackle the affine case, but for the sake of simplicity we will consider their ideas in the linear case.

The initial idea is quite simple, and consists in sieving the individual columns of S. Since we have $\mathbf{b} = \mathbf{a} \circ S$, then evaluating over a vector of the canonical basis of $(\mathbb{F}_q)^n$ yields: $\mathbf{b}(e_i) = \mathbf{a}(S_{\bullet i})$. Therefore, the set of possible candidates for the i-th column can be restricted to:

$$C_i = \{\mathbf{x} \in (\mathbb{F}_q)^n \mid \mathbf{b}(e_i) = \mathbf{a}(\mathbf{x})\}$$

Next, if we have sieved the possible candidate for columns i and j of S, we can sieve further, by noting that for all $\alpha, \beta \in \mathbb{F}_q$, we should have: $\mathbf{b}(\alpha \cdot e_i + \beta \cdot e_j) = \mathbf{a}(\alpha \cdot S_{\bullet i} + \beta \cdot S_{\bullet j})$. If the equation does not hold, then we know that the combination $(S_{\bullet i}, S_{\bullet j})$ is wrong. In other terms, we extend our candidate sets to pairs of columns:

$$C_{ij} = \{\mathbf{x}, \mathbf{y} \in (\mathbb{F}_q)^n \mid \forall \alpha, \beta \in \mathbb{F}_q,\ \mathbf{b}(\alpha \cdot e_i + \beta \cdot e_j) = \mathbf{a}(\alpha \cdot \mathbf{x} + \beta \cdot \mathbf{y})\}$$

We could extend our definition of candidate sets to more than two columns in the same way ; we illustrate this with the extension to three columns:

$$C_{ijk} = \{\mathbf{x}, \mathbf{y}, \mathbf{z} \in (\mathbb{F}_q)^n \mid \forall \alpha, \beta, \gamma \in \mathbb{F}_q,\ \mathbf{b}(\alpha \cdot e_i + \beta \cdot e_j + \gamma \cdot e_k) = \mathbf{a}(\alpha \cdot \mathbf{x} + \beta \cdot \mathbf{y} + \gamma \cdot \mathbf{z})\}$$

type	n	q	$\mathbf{F_5}$ running time (s)
inhomogeneous	8	2^{16}	0.14
	10		0.63
	12		2.16
	15		10.9
	17		27.95
	20		91.54
	10	65521	0.44
	15		8.08
	20		69.96
	23		235.91
semi-homogeneous	5	2^{16}	0.13
	6		1.03
	7		6.15
	8		54.34
	9		79.85
	10		532.33
homogeneous	5	2^8	30
	6		$\geq$ 14 days

Table 13.1: Practical performance of the Gröbner basis algorithm, using MAGMA's $\mathbf{F_4}$.

A Simplified Version. We are now ready to build a simplified version of the columnwise sieve. Algorithm 13.3 is the result of a compromise between faithfulness to the ideas of [GMS03] and simplicity. We stress that there are countless ways to improve our presentation of the algorithm, but we do not elaborate further.

Algorithm 13.3 A simplified version of the columnwise-sieve algorithm.

1: **function** Columnwise-Sieve($\mathbf{a}, \mathbf{b}$)
2: $C_1 \leftarrow \emptyset$ ▷ Compute C_1
3: **for all** $\mathbf{x} \in (\mathbb{F}_q)^n$ **do**
4: **if** $\mathbf{b}(e_1) = \mathbf{a}(\mathbf{x})$ **then** $C_1 \leftarrow C_1 \cup \{\mathbf{x}\}$
5: **for** i **from** 2 **to** n **do**
6: $C_i \leftarrow \emptyset$ ▷ Compute C_i
7: **for all** $\mathbf{x} \in (\mathbb{F}_q)^n$ **do**
8: **if** $\mathbf{b}(e_i) = \mathbf{a}(\mathbf{x})$ **then** $C_i \leftarrow C_i \cup \{\mathbf{x}\}$
9: $C_{1,2,\ldots,i} \leftarrow \emptyset$ ▷ Compute $C_{1,2,\ldots,i}$
10: **for all** $(\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_{i-1}) \in C_{1,2,\ldots,i-1}$ **and** $\mathbf{x}_i \in C_i$ **do**
11: **for all** $(k_1, \ldots, k_i) \in (\mathbb{F}_q)^i$ **do**
12: **if** $\mathbf{b}(k_1 \cdot e_1 + \cdots + k_i \cdot e_i) \neq \mathbf{a}(k_1 \cdot \mathbf{x}_1 + \cdots + k_i \cdot \mathbf{x}_i)$ **then** jump to line 14
13: $C_{1,2,\ldots,i} \leftarrow C_{1,2,\ldots,i} \cup \{(\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_i)\}$
14: **end for**
15: **end for**
16: **return** $C_{1,2,\ldots,n}$
17: **end function**

After regretting that the complexity analysis we are going to sketch is absent from [GMS03], we begin with a few remarks. First of all, Algorithm 13.3 always succeeds, and returns the set of all possible S. Secondly, just computing $C_1, C_2, \ldots, C_n$ makes its complexity $\Omega(n \cdot q^n)$. Taking into account the time (and space) complexity of what happens in line 9–14 is a bit more troublesome, as it cannot be done without (at least) estimating the sizes of the various sets handled by the algorithm.

It is reasonable to expect $|C_i| = q^{n-u}$. On the other hand, the usual complications appear as soon as we try to estimate the size of C_{ij}: it depends on whether $\mathbf{a}$ and $\mathbf{b}$ are homogeneous or not. Indeed, a given pair $(\mathbf{x}, \mathbf{y})$ must satisfy q^2 conditions to belong to C_i (one condition per pair (α, β) in the definition of C_i). The point is that some of these conditions are always satisfied, such as the condition with $\alpha = 0, \beta = 0$, which says nothing about $\mathbf{x}$ and $\mathbf{y}$.

– If $\mathbf{a}$ and $\mathbf{b}$ are inhomogeneous, this is the only always-satisfied condition, and therefore:

$$|C_{ij}| \approx q^{2\cdot n - u\cdot\left(q^2-1\right)}$$

– If $\mathbf{a}$ and $\mathbf{b}$ are homogeneous, then the conditions (α, β) and $(\alpha\cdot\gamma, \beta\cdot\gamma)$ are equivalent if $\gamma \neq 0$. To see why, observe that:

$$\mathbf{b}(\gamma\cdot\alpha\cdot e_i + \gamma\cdot\beta\cdot e_j) = \gamma^2\cdot\mathbf{b}(\alpha\cdot e_i + \beta\cdot e_j)$$

Something similar holds for $\mathbf{a}$. Thus, we may restrict α to be equal to either zero or one without loss of generality, and when $\alpha = 0$ we may safely restricts ourselves to $\beta = 1$. This leaves $q+1$ non-trivial conditions, and we find:

$$|C_{ij}| \approx q^{2\cdot n - u\cdot(q+1)}$$

So, as usual, things are worse in the homogeneous case. Having noted this fact, we switch back to the inhomogeneous case that was the prime concern of [GMS03]. By extending the above reasoning, we expect that:

$$|C_{i_1,\ldots,i_\ell}| \approx q^{\ell\cdot n - u\cdot\left(q^\ell-1\right)}$$

To measure the spatial complexity of the algorithm, we would need to measure the maximal size reached by $C_{1,\ldots,i}$ during the course of the computation. It is not particularly difficult to verify that the maximum cardinality is reached for $i_{\max} = \log_q(n/(u\ln q))$, and that the maximum cardinality itself is roughly:

$$q^u\cdot\left(\frac{n}{u\cdot\ln q}\right)^n\cdot e^{-n}$$

So, the number of times line 12 is executed when $i = i_{\max}$ is:

$$\begin{aligned} T_{12} &= |C_{i_1,\ldots,i_{\max}}|\cdot|C_{i_{\max}}|\cdot q^{i_{\max}} \\ &= q^u\cdot\left(\frac{n}{u\cdot\ln q}\right)^n\cdot e^{-n}\cdot q^{n-u}\cdot\frac{n}{u\ln q} \\ &= \frac{n}{u\cdot\ln q}\cdot\left(\frac{n\cdot q}{e\cdot u\cdot\ln q}\right)^n \end{aligned}$$

So all-in-all, the complexity of algorithm 13.3 is approximately:

$$T_{columnwise} = \max\left(n\cdot q^n,\ \frac{n}{u\cdot\ln q}\cdot\left(\frac{n\cdot q}{e\cdot u\cdot\ln q}\right)^n\right) \tag{13.4}$$

Figure 13.2 illustrates how the complexity of is influenced by q, n and u. It appears that increasing u makes the algorithm more efficient. Interestingly enough, $u = 1$ and $u = 2$ looks like the worst-cases, yet these settings yield the smallest keys in the identification scheme of §12.1.2.

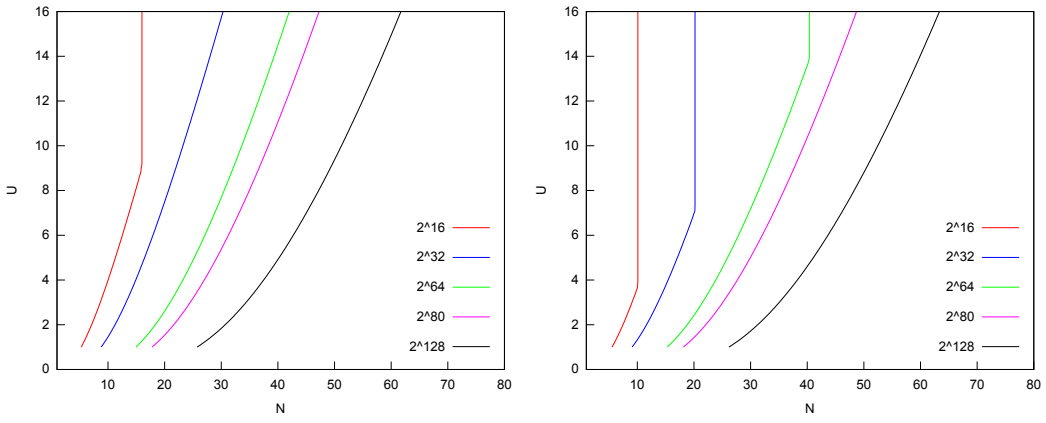
Possible Improvements Using Extension Fields. The authors of this algorithm proposed several non-trivial improvements to the simple version that we presented. The intuition behind our definition of C_i is that $\mathbf{b}(e_i) = \mathbf{a}(S_{\bullet i})$. One improvement is that this equality can be quantified over $\mathbb{F}_q$: for all $\lambda\in\mathbb{F}_q$, we have $\mathbf{b}(\lambda\cdot e_i) = \mathbf{a}(\lambda\cdot S_{\bullet i})$. We could therefore extend a bit our definition of C_i, by considering:

$$\left\{\mathbf{x}\in(\mathbb{F}_q)^n \ \mid \forall\lambda\in\mathbb{F}_q,\ \mathbf{b}(\lambda\cdot e_i) = \mathbf{a}\left(\lambda\cdot\mathbf{x}\right)\right\}$$

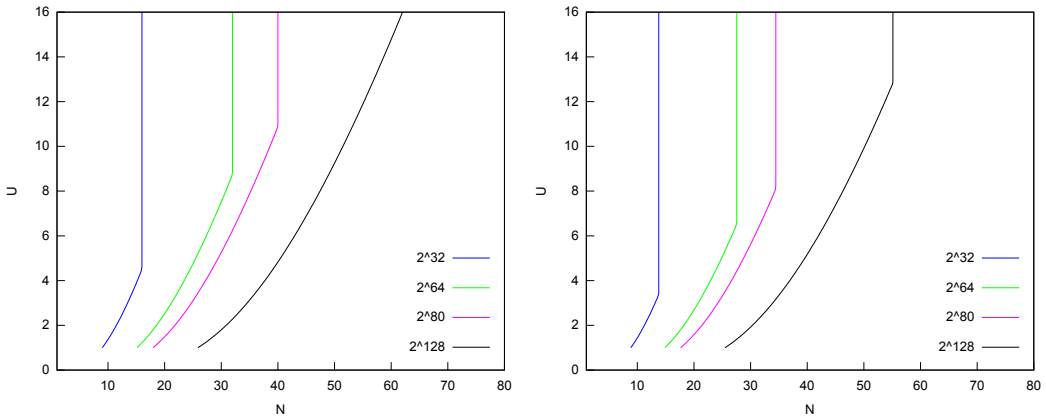
Note that when $q = 2$, then quantifying over all $\mathbb{F}_2$ does not improve anything. However, as the authors of [GMS03] pointed out, this problem can be circumvented by embedding $\mathbb{F}_q$ in a suitably large field extension (say, $\mathbb{F}_{q^k}$, with $q^k \geq d+1$), and setting:

$$C_i^{[k]} = \left\{\mathbf{x}\in(\mathbb{F}_q)^n \ \mid \forall\lambda\in\mathbb{F}_{q^k},\ \mathbf{b}(\lambda\cdot e_i) = \mathbf{a}\left(\lambda\cdot\mathbf{x}\right)\right\}$$

One may even hope that the cardinality of $C_i^{[k]}$ would be a decreasing function of k. Of course, we always have $C_i^{[k]}\subseteq C_i$. Introducing this new set of candidates would only be interesting if $C_i^{[k]}$ were quite smaller than C_i. Again, we must be very careful that *when* $\mathbf{a}$ *and* $\mathbf{b}$ *are homogeneous*, then $C_i = C_i^{[k]}$ (for reasons we in fact discussed above). However, in the inhomogeneous case this might work and sensibly reduce the size of the candidate sets. The following lemma will help us figure by how much, and generally clarifies things.



(a) $q = 2$ (b) $q = 3$



(c) $q = 4$ (d) $q = 5$

Figure 13.2: Iso-complexity lines of Algorithm 13.3 (in the inhomogeneous case) for various values of q. The complexity is smaller than indicated by each line on the left, and greater on the right. The lines become vertical when $n \cdot q^n$ dominates the complexity, and increasing u no longer helps in making $T_{columnwise}$ smaller in equation (13.4) smaller.

Lemma 13.1. *Let α be a primitive element in $\mathbb{F}_{q^k}$, and let $\beta_1, \ldots, \beta_d$ be d distinct non-zero elements of $\mathbb{F}_{q^k}$. Consider the set:*

$$\begin{aligned} \widetilde{C}_i^{[k]} &= \{\mathbf{x} \in (\mathbb{F}_q)^n \mid \mathbf{b}(\beta_1 \cdot e_i) = \mathbf{a}(\beta_1 \cdot \mathbf{x}), \ldots, \mathbf{b}(\beta_d \cdot e_i) = \mathbf{a}(\beta_d \cdot \mathbf{x})\} \\ \overline{C}_i^{[k]} &= \{\mathbf{x} \in (\mathbb{F}_q)^n \mid \mathbf{b}(\alpha \cdot e_i) = \mathbf{a}(\alpha \cdot \mathbf{x})\} \end{aligned}$$

When $k \geq d$, then we have: $C_i^{[k]} = \widetilde{C}_i^{[k]} = \overline{C}_i^{[k]}$.

Proof. Let $\mathbf{f}$ be an arbitrary polynomial of total degree d in n variables, and let $\mathbf{x}$ be a non-zero vector in $(\mathbb{F}_q)^n$. Let us define the degree-d polynomial over $\mathbb{F}_{q^k}[X]$:

$$P_{\mathbf{f},\mathbf{x}}(X) = \sum_{i=0}^{d} \mathbf{f}^{(i)}(\mathbf{x}) \cdot X^i$$

We obviously have: $\mathbf{f}(\lambda \cdot \mathbf{x}) = P_{\mathbf{f},\mathbf{x}}(\lambda)$. Therefore, we may reformulate the definition of $C_i^{[k]}$:

$$C_i^{[k]} = \bigcap_{j=1}^{u} \left\{\mathbf{x} \in (\mathbb{F}_q)^n \mid \forall \lambda \in \mathbb{F}_{q^k},\ P_{\mathbf{b}_j,e_i}(\lambda) = P_{\mathbf{a}_j,\mathbf{x}}(\lambda)\right\}$$

But it now becomes clear that if $P_{\mathbf{b}_j,e_i}$ and $P_{\mathbf{a}_j,\mathbf{x}}$ (which are are degree-d polynomials) coincide on $d+1$ points, then they are in fact equal coefficient-wise, and will coincide on the whole $\mathbb{F}_{q^k}$. And since we always have $P_{\mathbf{b}_j,e_i}(0) = P_{\mathbf{a}_j,\mathbf{x}}(0)$, we find that it is in fact sufficient to check d distinct values of λ (as opposed to

type	q	n	u	degree	predicted complexity
Linear inhomogeneous	7	7	1	cubic	2^{24}
	16	6	6	quadratic	$2^{27\star}$
	2	16	7		2^{22}
Affine inhomogeneous	16	6	3	quadratic	n.a.
	3	10	3		
	2	16	8		
	2	8	2		

Table 13.2: Parameters solved in practice in [GMS03]. Our reasoning does not cover the affine variant of the algorithm.

challenge	n	q	u	predicted complexity
A_1	16	2	2	$2^{52.9}$
A_2	20	2	2	2^{72}
A_3	32	2	2	2^{135}
B_1	6	16	2	$2^{26.6}$ $\star$
B_2	8	16	2	2^{35} $\star$
B_3	12	16	2	$2^{51.6}$ $\star$
C_1	16	2	1	$2^{69.9}$
C_2	20	2	1	2^{93}
C_3	32	2	1	$2^{168.3}$
D_1	6	16	1	$2^{26.6}$ $\star$
D_2	8	16	1	2^{35} $\star$
D_3	12	16	1	$2^{58.2}$

Table 13.3: Projected complexity of the Columnwise Sieve on the challenges.

q^k). This shows that $C_i^{[k]} = \widetilde{C}_i^{[k]}$. We observe that as soon as $q \geq d+1$, then there is no gain in using a field extension, since we may very well choose the β_i in $\mathbb{F}_q$.

It remains to show that if $k \geq d$, then $C_i^{[k]} = \overline{C}_i^{[k]}$. Let us thus assume that $k \geq d$, and that $P_{\mathbf{b}_j,e_i}(\alpha) = P_{\mathbf{a}_j,\mathbf{x}}(\alpha)$. Applying the Frobenius map to both sides of this equation yields:

$$\begin{aligned} P_{\mathbf{b}_j,e_i}(\alpha)^q &= P_{\mathbf{a}_j,\mathbf{x}}(\alpha)^q \\ \sum_{k=0}^{d} \alpha^{q\cdot k} \cdot \mathbf{b}_j^{(k)}(e_i) &= \sum_{k=0}^{d} \alpha^{q\cdot k} \cdot \mathbf{a}_j^{(k)}(\mathbf{x}) \\ P_{\mathbf{b}_j,e_i}(\alpha^q) &= P_{\mathbf{a}_j,\mathbf{x}}(\alpha^q) \end{aligned}$$

This argument repeats, and we find that for any integer $1 \leq k \leq d$, $P_{\mathbf{b}_j,e_i}\left(\alpha^{q^k}\right) = P_{\mathbf{a}_j,\mathbf{x}}\left(\alpha^{q^k}\right)$. This shows that $\widetilde{C}_i^{[k]} \subseteq \overline{C}_i^{[k]}$ (with $\beta_i = \alpha^{q^i}$), and establishes the result. □

In [GMS03], it was suggested to choose a somewhat large extension, with the underlying hope that the bigger the extension is, the smaller $C_i^{[k]}$ would get. Lemma 13.1 shreds this hope to pieces. Making use of an extension field is only useful if $q \leq d$ (typically, when $q = 2$). In the quadratic case, with $q = 2$, a degree-two extension is sufficient, and testing only *one* well-chosen value of λ is also sufficient. It also makes it reasonable to expect $|C_i^{[k]}| \approx q^{n-d\cdot u}$, as soon as $q^k \geq d+1$.

Achievements. To conclude, we report in Table 13.2 the parameter ranges that have been broken in practice in [GMS03]. The star ($\star$) indicates that the running time should be dominated by the computation of the C_i.

We have extrapolated how much time would the Columnwise Sieve require to break the challenges, and show the result in Table 13.3. We first conclude that it should clearly be able to break the linear inhomogeneous legacy challenges $\widetilde{\mathrm{B}_1}$ and $\widetilde{\mathrm{D}_1}$ with roughly 2^{27} operations. It seems plausible that it would also have broken the affine variants. The challenges with greater n are more difficult: we project that the algorithm would perform 2^{53} evaluations of $\mathbf{a}$ and store 2^{35} candidates to solve $\widetilde{\mathrm{A}_1}$. These figures grow to 2^{70} operations and 2^{50} candidates for challenge $\widetilde{\mathrm{C}_1}$. The affine cases should be even harder.

q	n	u	(projected) complexity of algorithm 13.3
2	15	15	2^{18}
2	15	13	2^{18}
2	10	11	2^{18}
11	10	10	2^{38}
11	10	9	2^{38}

Table 13.4: Parameters solved in practice in [dVP03]. In all these cases, the running time of algorithm 13.3 would be dominated by the computation of the C_i.

Moving on to the updated challenges, it seems plausible that the algorithm would break $\overline{\mathrm{B}_2}$ and $\overline{\mathrm{D}_2}$ (in 2^{35} operations). Breaking $\overline{\mathrm{B}_3}$ seems theoretically possible, but would be challenging (2^{52} operations are required), and breaking $\overline{\mathrm{D}_3}$ seems quite impractical (2^{58} operations required). All the other challenges are out of reach.

13.4 The "Algebraic Columnwise Sieve" Algorithm

In [dVP03], Perret and Levy-dit-Vehel proposed a reformulation of the columnwise sieve in more algebraic terms. It is indeed clear that:

$$C_i = \mathbf{a}^{\langle -1 \rangle}\left(\mathbf{b}(e_i)\right)$$

Let us try to move to a more algebraic framework. Let us consider the ring $R = \mathbb{F}_q[\mathbf{x}_1, \ldots, \mathbf{x}_n]$, and let $\mathcal{I}_q = \langle {x_i}^q - x_i \rangle_{1 \leq i \leq n}$ be the "field ideal" whose variety is precisely $(\mathbb{F}_q)^n$. It is not hard to express C_i as a variety:

$$C_i = \mathbf{V}\left(\mathcal{I}_q + \langle \mathbf{b}_j(e_i) - \mathbf{a}_j(\mathbf{x}) \rangle_{1 \leq j \leq u}\right)$$

The same goes for the more sophisticated version C'_i, which quantifies over $\mathbb{F}_q$:

$$C'_i = \mathbf{V}\left(\mathcal{I}_q + \left\langle \mathbf{b}_j(\alpha \cdot e_i) - \mathbf{a}_j(\alpha \cdot \mathbf{x}) \right\rangle_{\substack{1 \leq i \leq n \\ \alpha \in \mathbb{F}_q,\ \alpha \neq 0}}\right)$$

Interestingly, the discussion of lemma 13.1 has a counterpart here: in the inhomogeneous case, C'_i contains only $d \cdot u$ linearly independent equations (and not $q \cdot u$).

The first algorithm given in [dVP03] essentially amounts to compute $C'_1, C'_2, \ldots, C'_n$ trough a Gröbner basis computation, and then to exhaustively search their cartesian product. The point is that for big values of q, computing C'_i this way is going to be more efficient than by exhaustive search. However, if $n - u$ is big, searching the cartesian product will be prohibitivey expensive. As it seems that Levy-dit-Vehel and Perret mostly had the $u = n$ case in mind, we also stick to this case. In this setting, then computing C'_i amounts to solve a system of $d \cdot n$ equations of degree d in n variables. In the case where $d = 2$, we find that the degree of regularity of a system of $2n$ semi-regular quadratic equations in n variables is (thanks to (10.2)):

$$D_{\mathrm{reg}} = 0.0858n + 1.041n^{1/3} - 1.50$$

Using this, we may estimate, more or less precisely, the complexity determining C'_i. Figure 13.3 shows an asymptotic improvement over exhaustive search when $q \geq 3$. The practical signification of this asymptotic result is not completely clear though.

Achievements. We report in Table 13.4 the parameter ranges that were broken in practice in [dVP03]. Again, we note that the improvement over [GMS03] is not breathtaking. On the other hand, the difference would probably be more visible on bigger fields, where Gröbner bases have a greater edge over exhaustive search.

13.5 A Revolution in the QFSE World : the Jacobian Algorithm

> De l'audace, encore de l'audace,
> toujours de l'audace !
>
> Georges Jacques Danton
> (1759–1794)

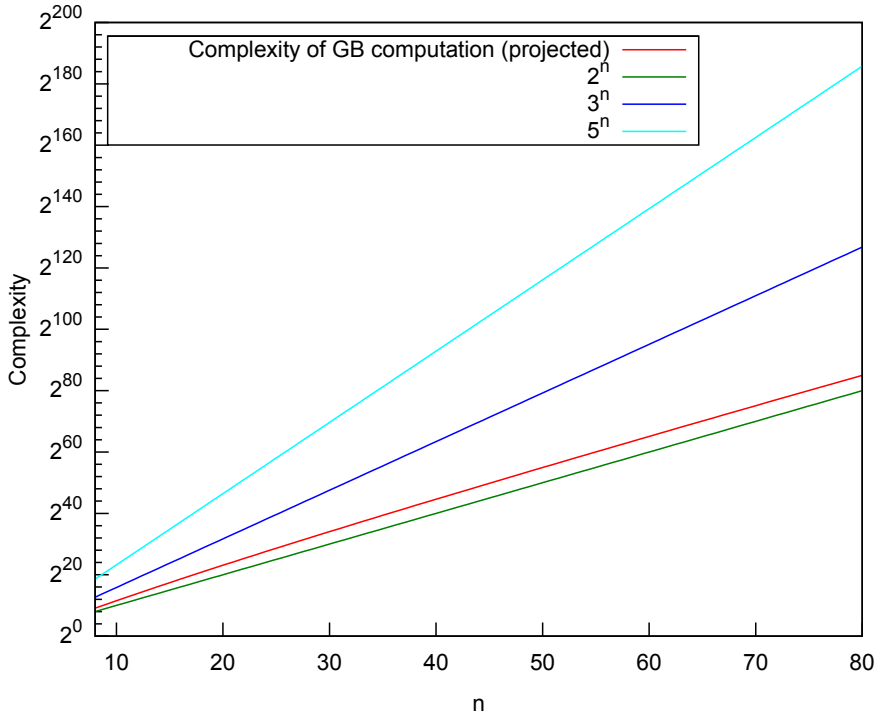


Figure 13.3: Expected improvement from replacing exhaustive search by Gröbner bases computations.

This section is devoted to the aglorithm described by Perret in [Per05]. We call it the "Jacobian Algorithm", since one of its distinctive features is that it makes use of Jacobian matrices. This algorithm clearly improves on its predecessors, since it terminates in time *polynomial* in n, under the condition that $u \approx n$. We first recall the definition of Jacobian matrices.

Definition 13.1. Let $\mathbf{f} : (\mathbb{F}_q)^n \to (\mathbb{F}_q)^u$ be an arbitrary function. The **Jacobian matrix** of $\mathbf{f}$ is the $u \times n$ matrix:

$$J_{\mathbf{f}} = \begin{bmatrix} \frac{\partial \mathbf{f}_1}{\partial x_1} & \cdots & \frac{\partial \mathbf{f}_1}{\partial x_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial \mathbf{f}_u}{\partial x_1} & \cdots & \frac{\partial \mathbf{f}_u}{\partial x_n} \end{bmatrix}$$

where $\frac{\partial \mathbf{f}}{\partial x_1}$ denotes the usual high-school "Calculus I" derivative.

If $g(x) = f(\alpha \cdot x)$, then a high-school student would find that $\frac{\mathrm{d}g}{\mathrm{d}x}(y) = \alpha \cdot \frac{\mathrm{d}f}{\mathrm{d}x}(\alpha \cdot y)$. The following result is a generalization to the multivariate setting.

Lemma 13.2 ([Per05], theorem 1)**.** *For all* $\mathbf{x} \in (\mathbb{F}_q)^n$*, we have:* $J_{\mathbf{b}}(\mathbf{x}) = J_{\mathbf{a}}(S \cdot \mathbf{x}) \times S$

A particularly interesting consequence is that evaluating the Jacobian matrix on the null vector does not require prior knowledge of S, and yet yields $u \cdot n$ linear equations on the entries of S:

$$J_{\mathbf{b}}(\mathbf{0}) = J_{\mathbf{a}}(\mathbf{0}) \times S \tag{13.5}$$

Intuitively, it is visible that if $u = n$ then this nearly completely reveals S. More precisely, it *does* reveal S if the Jacobian matrix is nonsingular. In fact, $J_{\mathbf{a}}(0)$ is exactly the matrix formed by the coefficients of the linear terms in $\mathbf{a}$, thus it is fully random if $\mathbf{a}$ is random. Therefore when $u = n$, the probability that equation (13.5) completely specifies S is $\lambda(n)$ (cf. §10.7) . Therefore, bigger values of q makes it more likely to be able to terminate in polynomial time when $u = n$.

We remark that, as usual, nothing comes for free when $\mathbf{a}$ is homogeneous: in that case, the entries of its Jacobian matrix are also homogeneous, and thus $J_{\mathbf{a}}(\mathbf{0})$ is... the null matrix.

Now, let us turn the idea of using the Jacobian matrix into a full algorithm. The approach described in [Per05] is to find ℓ relations $\mathbf{y}_i = S \cdot \mathbf{x}_i$ on S, and then use lemma 13.2 to obtain $\ell \cdot u \cdot n$ additional linear

equations in the coefficients of S. As soon as n^2 linearly independent such equations have been obtained, then S can be fully reconstructed with usual linear algebra, in time $\mathcal{O}\left(n^6\right)$.

The value of ℓ is therefore the critical parameter of the algorithm. Let us denote by $\ell^\perp$ the minimal possible value (assuming no unfortunate linear dependency occurs), as a function of n and u. In the inhomogeneous case, equation (13.5) brings in $u \cdot n$ equations, the ℓ relations bring $\ell \cdot n$ equations, and using the Jacobian matrix (by fixing $\mathbf{x}$ in lemma 13.2) the ℓ relations give birth to an additional $\ell \cdot u \cdot n$ equations. Therefore, if all the linear equations we obtain are independant, we will need to acquire at least $\ell^\perp = \lceil (n-u)/(u+1) \rceil$ relations. In the homogeneous case, a similar argument yields $\ell^\perp \geq \lceil n/(u+1) \rceil$.

13.5.1 Selective Exhaustive Search

The relations are found thanks to a so-called "selective exhaustive search" that exploits the idea that S transforms $\ker J_{\mathbf{a}}(0)$ into $\ker J_{\mathbf{b}}(0)$, as implied by equation (13.5). It also exploits the similar fact that S transforms the kernel of $\mathcal{P}\left(\mathbf{a}_i\right)$ into that of $\mathcal{P}\left(\mathbf{b}_i\right)$. The exhaustive search process can therefore be restricted to these subspaces of $\left(\mathbb{F}_q\right)^n$, assuming that they are not reduced to the zero vector, and hoping that they are not too big.

We will try to derive some useful results on the complexity of the Selective Exhaustive Search, assuming that instances are randomly chosen. This will enable us to derive complexity bounds on the full algorithm.

In the inhomogeneous case, we will assume that $u \leq n$. Indeed, if $u \geq n$, then $J_{\mathbf{a}}(0)$ has full rank with high probability and it should be possible to deduce S from equation (13.5) in nearly all cases. If $u = n$, we will assume that $\det J_{\mathbf{a}}(0) = 0$, for the same reason. We will focus on the inhomogeneous case, and we will distinguish between two situations: $u = n$ and $u \geq n/2$.

Case $u = n$. If $J_{\mathbf{a}}(0)$ is invertible, then S can be deduced from (13.5) in time $\mathcal{O}\left(n^3\right)$, and this happens with probability $\lambda(n)$. In the other case, we see that $\ell^\perp = 1$, so we will focus on obtaining one relation. For the sake of simplicity, we forget about the polar forms, since we are guaranteed to find our relation in the kernel of the jacobian matrix. We thus pick an arbitrary $\mathbf{x} \in \ker J_{\mathbf{a}}(0)$, and we try all the possible $\mathbf{y} \in J_{\mathbf{b}}(0)$. The expected number of tries is the expected cardinality of $\ker J_{\mathbf{a}}(0)$, which is a uniformly distributed random matrix as argued above. Lemma 10.20 then tells us that the expected number of tries is $q + \mathcal{O}\left(1/q\right)$. Therefore we conclude that finding the single needed relation is very efficient, and the algorithm should run in expected polynomial time, even if the Jacobian matrix is non-invertible.

Case $u \geq n/2$. It is easy to deduce from the expression of $\ell^\perp$ that $n/2$ is the smallest value of u such that a single relation will be neeed. However, the kernel of $J_{\mathbf{a}}(0)$ will then contains q^{n-u} elements, and searching it is expensive. We therefore turn our attention to the kernels of the polar forms, hoping that they will not all be trivial. We have determined the probablity that $\mathcal{P}\left(\mathbf{a}_i\right)$ is non-singular in §10.7.1. Since the original presentation of the algorithm assumed q to be odd, we will stick to this choice, and then the probability that will find at least one relation without searching $\ker J_{\mathbf{a}}$ is exactly:

$$1 - \left(\frac{\lambda(q,n)}{\lambda(q^2,n/2)} \right)^n \approx 1 - e^{-n/q}$$

We observe that this estimate is surprisingly accurate. All-in-all, we conclude that as long as $u \geq n/2$, the expected running time of the algorithm is:

$$\mathbb{E}\left[\text{running time}\right] = n^6 + e^{-n/q} \cdot q^{n-u} \tag{13.6}$$

Homogeneous Case. In the homogeneous, $\ker J_{\mathbf{a}}(0) = \left(\mathbb{F}_q\right)^n$, and searching it is no longer an option. The only way the selective exhaustive search could perform better than an actual exhaustive search is by exploiting the kernels of the $\mathcal{P}\left(\mathbf{a}_i\right)$'s. However, we have seen that with they will all be trivial with probability about $e^{-n/q}$. When $u = n$, in which case only one relation is required, the expected complexity is therefore $\mathcal{O}\left(n^6 + e^{-q/n} \cdot q^n\right)$.

Achievements. We report in Table 13.5 the parameter ranges that were broken in practice in [Per05].

Because of its ability to solve instance with arbitrarily large values of n, the Jacobian algorithm is an actual milestone in QFSE solving. This is made possible by the use of different techniques, such as the Jacobian matrix, and by the idea to make S appear as the solution of a system of linear equations that must be built during the course of the algorithm. Because the algorithm is plainly exponential when $u \lll n$, the Jacobian algorithm unfortunately cannot tackle the challenges.

Note that from a historical perspective, the *Jacobian* is the determinant of the Jacobian matrix, named after Carl Gustav Jacob Jacobi (1804–1851). It is not to be mistaken with the *Jacobins*, the most radical

q	n	u	resolution time (quadratic)	projected	resolution time (cubic)
2	70	70	10s	poly.	5min
2	70	65			
2	70	60			
2	70	55			
11	60	60	0.2s	poly.	10s
11	60	55	2min	$2^{9.5}$	1h
11	60	50		$2^{26.5}$	
257	50	50	0.2s	poly.	10s
257	50	45	3min	2^{40}	6h

Table 13.5: Parameters solved in practice in [Per05]. The instances are linear and inhomogeneous.

political fraction of the National Convention during the French Revolution. Along with their political Leader, Maximilien Robespierre, they are often associated with the Reign of Terror (September 1793–July 1794), the most "intense" and critical period of the Revolution. As for QFSE instances, they shall be terrorized some more in chapter 15.

Chapter 14

A General Method for Quadratic PLE: Matrix Pencils

In this chapter we introduce a generic method to solve some classes of polynomial equivalence problems efficiently. This general method naturally has to be specialized for each particular case.

Our method is based on *pencils* of matrices. The key idea is that the solutions of an instance of the polynomial equivalence problem at hand are also generally solutions of a certain *matrix pencil equivalence problem*. Before going into the details, let us quickly define what these objects are.

Definition 14.1. An $n \times n$ **matrix pencil** $\mathbf{P}_{\lambda,\mu}$ over $\mathbb{K}$ is an $n \times n$ matrix whose coefficients are homogeneous polynomials of degree one in two variables (usually denoted by λ and μ). It can also be written $\mathbf{P}_{\lambda,\mu} = \lambda \cdot A + \mu \cdot B$ where A and B are matrices over $\mathbb{K}$. The dependency in the two variables is often omitted.

Definition 14.2. Two matrix pencils $\mathbf{P}$ and $\mathbf{Q}$ are said to be **equivalent** if there exist two invertible matrices S and T such that:

$$T \times \mathbf{P} = \mathbf{Q} \times S, \qquad S, T \in \mathrm{GL}_n(\mathbb{K}) \tag{14.1}$$

This is an obvious generalization of the usual notion of matrix equivalence to matrix pencils. In addition, we remark that if $\mathbf{P} = \lambda \cdot A + \mu \cdot B$ and $\mathbf{Q} = \lambda \cdot C + \mu \cdot D$, then:

$$T \times \mathbf{P} = \mathbf{Q} \times S \iff \left\{ \begin{array}{rcl} T \times A & = & C \times S \\ T \times B & = & D \times S \end{array} \right. \tag{14.2}$$

Definition 14.3. A Matrix pencil $\mathbf{P}_{\lambda,\mu}$ is said to be **regular** if the determinant $\det \mathbf{P}_{\lambda,\mu}$, which is in fact a homogeneous bivariate polynomial in $\mathbb{K}[\lambda, \mu]$, is not identically zero. A non-regular pencil is said to be **singular**.

The interest of making this distinction is that when dealing with regular pencils, we may without loss of generality assume that one of the two matrices is invertible. Suppose that two pencils $\mathbf{P} = \lambda \cdot A + \mu \cdot B$ and $\mathbf{Q} = \lambda \cdot C + \mu \cdot D$ are equivalent, and let $\mathbb{V}(\mathbf{P}, \mathbf{Q})$ denote the set of pairs (S, T) such that $T \times \mathbf{P} = \mathbf{Q} \times S$. We introduce a "change of coordinates" in the parameters:

$$\begin{array}{rcl} \lambda & = & \alpha_1 \tilde{\lambda} + \alpha_2 \tilde{\mu} \\ \mu & = & \beta_1 \tilde{\lambda} + \beta_2 \tilde{\mu} \end{array} \qquad \alpha_1 \beta_2 - \alpha_2 \beta_1 \neq 0$$

And we express the two pencils using the new parameters:

$$\begin{array}{rcl} \tilde{\mathbf{P}} & = & \tilde{\lambda}(\alpha_1 A + \beta_1 B) + \tilde{\mu}(\alpha_2 A + \beta_2 B) \\ \tilde{\mathbf{Q}} & = & \tilde{\lambda}(\alpha_1 C + \beta_1 D) + \tilde{\mu}(\alpha_2 C + \beta_2 D) \end{array}$$

Because $\alpha_1 \beta_2 - \alpha_2 \beta_1 \neq 0$, the change of coordinates is reversible, and $(\mathbf{P}, \mathbf{Q})$ can be obtained back from $(\tilde{\mathbf{P}}, \tilde{\mathbf{Q}})$ after an other related change of coordinates. Under this condition, the change of coordinates preserves the equivalence, and $\mathbb{V}(\mathbf{P}, \mathbf{Q}) = \mathbb{V}(\tilde{\mathbf{P}}, \tilde{\mathbf{Q}})$. Since $\mathbf{P}$ is regular, there exists a pair (α_1, β_1) such that $\det \alpha_1 A + \beta_1 B \neq 0$, and α_2, β_2 can be chosen so that the constraint $\alpha_1 \beta_2 - \alpha_2 \beta_1 \neq 0$ is satisfied. However, $\alpha_1, \beta_1, \alpha_2$ and β_2 may only live in a suitable field extension of $\mathbb{F}_q$, and not in $\mathbb{F}_q$ itself. This requires embedding the whole problem in a larger field, but does not modifies the set of solutions.

The Pencil Equivalence Problem. The matrix pencil equivalence problem is *easy*, as it essentially amounts to solve a system of $2n^2$ homogeneous linear equations in $2n^2$ unknowns. It has been extensively studied in the second half of the 19th century: Weierstraß established necessary and sufficient conditions for two regular

pencils to be equivalent in 1867, and Kronecker extended these results to the singular case in 1890. They also demonstrated that these pencils, just like regular matrices, admit nearly unique canonical forms that can be computed in time $\mathcal{O}\left(n^3\right)$, over any field. The system of matrix equations on the right-hand-side of (14.2) is called a *coupled generalized Sylvester equation*, after James J. Sylvester (1814–1897), who incidentally also invented the term "matrix". These equations appear frequently in the resolution of *generalized eigenproblems*, where given A and B the point is to find vectors $\mathbf{x}$ such that $A \cdot \mathbf{x} = \lambda B \cdot \mathbf{x}$ (see [vL00] for more background).

Coming back to polynomial equivalence, our idea is that, given an instance $(\mathbf{a}, \mathbf{b})$ of a polynomial equivalence problem, it is sometimes possible to find two *matrix pencils* $\mathbf{P}$ and $\mathbf{Q}$ such that the solutions of the polynomial equivalence problem are also solutions of the pencil equivalence problem:

$$T \circ \mathbf{b} = \mathbf{a} \circ S \Longrightarrow T \cdot \mathbf{P} = \mathbf{Q} \cdot S$$

The converse will not be true in general, because pencil equivalence is an inherently linear problem while PLE problems are non-linear. Our strategy is to compute a basis of the vector space $\mathbb{V}(\mathbf{P}, \mathbf{Q})$ in which the solutions of the pencil equivalence problem are living. The point is that the dimension of this vector space will generally be much smaller than $2n^2$, and knowing that S and T live in $\mathbb{V}(\mathbf{P}, \mathbf{Q})$ *dramatically* reduces the quantity of information that remains to be found. To see why is this useful, let us consider an instance of an (unspecified) PLE problem:

$$T \circ \mathbf{b} = \mathbf{a} \circ S$$

and the (unspecified) associated matrix pencil equivalence:

$$T \times \mathbf{P} = \mathbf{Q} \times S$$

We build the pencils so that solutions of the PLE problem are also solutions of the pencil equivalence problem. Let $\ell = \dim \mathbb{V}(\mathbf{P}, \mathbf{Q})$ and $\left(S^{[1]}, T^{[1]}\right), \ldots, \left(S^{[\ell]}, T^{[\ell]}\right)$ be a basis thereof. Let us also consider the polynomial ring $\mathbb{F}_q[x_1, \ldots, x_\ell]$. Because all the solutions of the PLE instance belong to $\mathbb{V}(\mathbf{P}, \mathbf{Q})$, there is a one-to-one correspondance between them and the solutions in $x_1, \ldots, x_\ell$ of the following equation:

$$\left(\sum_{i=1}^{\ell} x_i \cdot T^{[i]}\right) \circ \mathbf{b} = \mathbf{a} \circ \left(\sum_{i=1}^{\ell} x_i \cdot S^{[i]}\right) \qquad (\star)$$

Following the ideas of Faugère and Perret, we obtain an equivalent system $\mathcal{S}_{\text{quad}}$ of quadratic equations in $x_1, \ldots, x_\ell$ by identifying coefficient-wise both side of $(\star)$. Compared to the "direct" method exposed exposed in §13.2, we have as many equations, but we hope to have much less variables. The implicit hope is that in most (non-completely degenerate) situations the dimension of $\mathbb{V}(\mathbf{P}, \mathbf{Q})$) is going to be approximately n, rather than $2n^2$. We will show in §14.3 below that this hope is quite reasonable.

If the equations composing $\mathcal{S}_{\text{quad}}$ were linearly independent, then it would be plausible to determine its solutions efficiently, either by plain linearization or by a Gröbner basis computation.

The rest of this chapter is organised as follows: in §14.2 we express the dimension of $\mathbb{V}(\mathbf{P}, \mathbf{Q})$ in terms of the invariants of the pencil, and we estimate this dimension when the pencils are random in §14.3. We then discuss how to determine a basis of the solution space in §14.4, and how to generate the quadratic equations of $\mathcal{S}_{\text{quad}}$ in §14.5.

14.1 More Linear Algebra Background

A λ-matrix is a matrix whose entries are polynomials in λ. Given any rank-r (eventually rectangular) λ-matrix $A(\lambda)$, we denote by $D_j(\lambda)$ the greatest common divisor of all the minors of order j of $A(\lambda)$ (we assume it is unitary). It is (said to be) easy to see that in the series

$$D_r(\lambda), D_{r-1}(\lambda), \ldots, D_1(\lambda), D_0(\lambda) := 1$$

each polynomial is divisible by the preceding one. The corresponding quotients will be denoted by $i_1(\lambda), \ldots, i_r(\lambda)$:

$$i_r(\lambda) = \frac{D_r(\lambda)}{D_{r-1}(\lambda)}, \ldots, i_1(\lambda) = \frac{D_1(\lambda)}{D_0(\lambda)} \qquad (14.3)$$

Definition 14.4. The polynomials $i_1(\lambda), \ldots, i_r(\lambda)$ defined by (14.3) are called the **invariant polynomials of** $A(\lambda)$. Given a "normal" matrix M with entries in $\mathbb{F}_q$, the **invariant polynomials of** M (also called **invariant factors** or **similarity invariants**) are the n invariant polynomial of the λ-matrix $M - \lambda \cdot I_n$.

The important result is that two square matrices are similar if and only if they have the same invariant polynomials. The equivalence classes for the similarity relation are thus formed of all matrices sharing the same the invariant polynomials.

In addition, if $i_n(\lambda), \ldots, i_1(\lambda)$ are the invariant polynomial of M, then i_n is the minimal polynomial of M, the product $i_n \times \cdots \times i_1$ is the characteristic of M, and i_k divides i_{k+1}.

14.1.1 Commutativity

It is well-known that matrix multiplication is not commutative. However, it sometimes happen that $A \times B = B \times A$. For instance, all powers of A commute with A.

Definition 14.5. Given an $n \times n$ matrix A, the set of all matrices B such that $A \times B = B \times A$ forms a vector space called the **commutant of** A.

The commutant is a classical object of linear algebra, and the equation $A \times X = X \times A$ has been the object of much study. For instance, when the minimal and characteristic polynomial of a matrix coincide, then all the powers of A are linearly independent, and the commutant is at least of dimension n. This fact is in fact more generally true.

Theorem 14.1. *The commutant of any matrix is of dimension greater than or equal to n.*

Another classical result relates the exact dimension of the commutant of a matrix A to its invariant polynomials. The following theorem in fact tells us, for instance, that in the case where the minimal and characteristic polynomial of a matrix coincide, then the commutant is of dimension exactly n (*i.e.*, as small as it gets).

Theorem 14.2 ([Gan59],chapter VIII, §2, theorem 2)**.** *The dimension of the commutant of a given matrix A is*

$$\sum_{k=1}^{r} (2k-1) \deg i_k(\lambda),$$

where $i_k(\lambda)$ is the k-th invariant factor of A.

14.2 Dimension of Matrix Pencils Solution Spaces.

We now move on to discuss the dimension of $\mathbb{V}(\mathbf{P}, \mathbf{Q})$ under the assumption that the pencils $\mathbf{P}$ and $\mathbf{Q}$ are *regular*. If $\mathbf{P}$ and $\mathbf{Q}$ are not equivalent, then the problem is not very interesting, and we will therefore assume that they are equivalent. The following theorem seems to be a standard result, altough it is apparently not very well-known.

Theorem 14.3. *Let $\mathbf{P} = \lambda A + \mu B$ and $\mathbf{Q} = \lambda C + \mu D$ be two $n \times n$ regular and equivalent pencils, where in addition A and C are (without loss of generality) non-singular. Let $i_s(\lambda), \ldots, i_1(\lambda)$ be the invariants polynomials of $\mathcal{C} = B \times A^{-1}$. Then we have*

$$n \leq \dim \mathbb{V} = \dim \text{Commutant}(\mathcal{C}) = \sum_{k=1}^{s} (2s - 2k + 1) \cdot \deg i_k(\lambda)$$

Proof. Exploiting the non-singularity of A and C, it is possible to express S as a function of T in a unique way:

$$S = C^{-1} \cdot T \cdot A$$

Substituting this new expression of S in $T \times \mathbf{P} = \mathbf{Q} \times S$ yields the equivalent equation:

$$T \times B \times A^{-1} = D \times C^{-1} \times T \tag{14.4}$$

Next, we make use of our particular solution (S_0, T_0). Because T_0 is a solution of (14.4), then this very equation is equivalent to:

$$\left(B \times A^{-1}\right) \times \left({T_0}^{-1} \times T\right) = \left({T_0}^{-1} \times T\right) \times \left(B \times A^{-1}\right)$$

From there it is easy to see that the solution space of equation (14.1) is isomorphic to the commutant of $\mathcal{C} = B \times A^{-1}$. The theorem then follows from theorems 14.1 and 14.2. □

14.3 Expected Dimension of $\mathbb{V}$.

As discussed above, the interest of considering a matrix pencil is to discover, without expensive computation, as "small" subspace in which the solutions of a PLE problem live. But how small does it gets on average? If we pick up a random regular pencil, then $\mathcal{C} = B \times A^{-1}$ is a random matrix (we implicitly assume that A is invertible, as we may make it invertible when the pencils are regular).

As seen in §14.1, the solution space of the pencil equivalence problem is as small as possible as possible when the minimal polynomial of $\mathcal{C}$ is in fact also its characteristic polynomial.

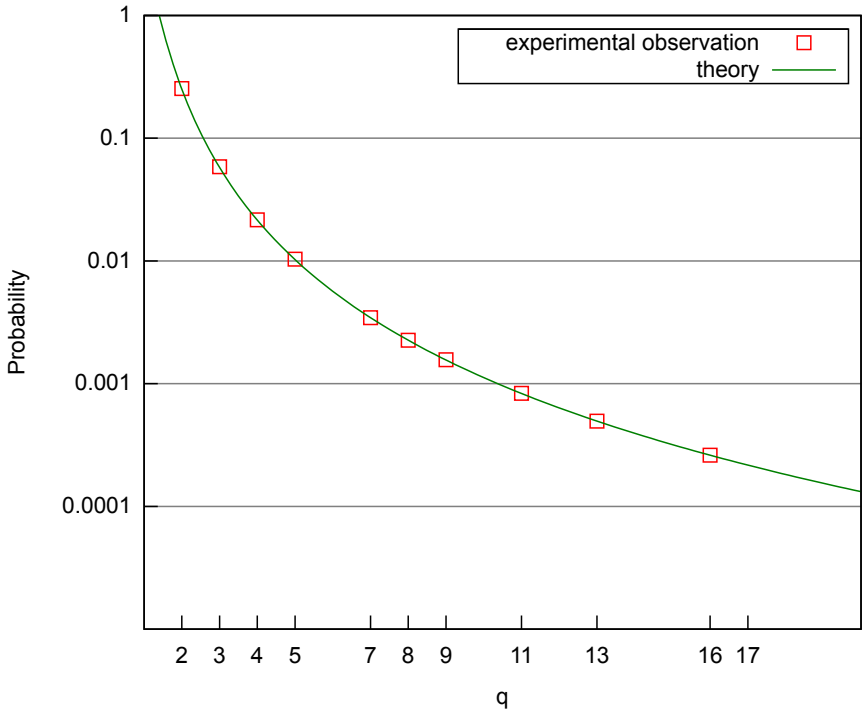


Figure 14.1: Probability to *not* be cyclic, observed with $n = 32$ for various values of q and a large number of samples.

Definition 14.6. A matrix whose minimal and characteristic polynomials coincide is said to be a **cyclic matrix**.

This name comes from the observation that given a cyclic matrix A, there is a vector $\mathbf{x}$ such that, $A^i \cdot \mathbf{x}$ for $i = 1, \ldots, n$, is a basis of $(\mathbb{F}_q)^n$. In fact, most matrices are cyclic.

Theorem 14.4 ([Ful01], theorem 1)**.** *Let $c(n, q)$ be the proportion of cyclic $n \times n$ matrices. We have:*

$$\frac{1}{q^2(q+1)} < 1 - c(n,q) < \frac{1}{(q^2-1)(q-1)}$$

And asymptotically, we have:

$$\lim_{n\to\infty} c(n,q) = \frac{q^5-1}{q^2(q-1)(q^2-1)} \cdot \prod_{i=1}^{\infty}\left(1 - \frac{1}{q^i}\right)$$

This is particularly interesting, because when $\mathcal{C}$ is cyclic, then theorem 14.3 tells us that the dimension of the solution space of the pencil equivalence problem reaches its smallest possible value (namely, n). Combining theorem 14.4 with equation (10.4) yields:

$$\lim_{n\to\infty} \mathbb{P}\left[\text{a random } n \times n \text{ matrix is cyclic}\right] = 1 - q^{-3} - q^{-4} - 2q^{-5} - q^{-6} + \mathcal{O}\left(q^{-7}\right) \tag{14.5}$$

It follows that $\mathcal{C}$ is cyclic with a not-at-all negligible probability of order $1 - 1/q^3$. We deduce therefrom that things only need to be " random enough" for this approach to yield its full potential. As suggested by the non-asymptotic statement of the theorem, $c(n, q)$ seems to only weakly depend on n, and quickly converges to its limit. Figure 14.1 shows that the actual probability coincides quite well with its limit.

In general, if the instance of the PLE problem is randomly chosen, then $\dim \mathbb{V}$ may be seen as a random variable, and its distribution —which is specific to the way $\mathbf{P}$ and $\mathbf{Q}$ are built— depends on the parameters of the problem. If $\mathcal{C}$ is random though, then as argued above $\dim \mathbb{V}$ will often be minimal. This makes this approach quite promising.

14.4 Computational Cost of Determining $\mathbb{V}$

Let M_ϕ denote the $2n^2 \times 2n^2$ matrix representing the following linear operator:

$$\begin{array}{rccc} \phi: & (\mathbb{F}_q)^{2n^2} & \to & (\mathbb{F}_q)^{2n^2} \\ & (X,Y) & \to & \begin{pmatrix} Y \times A - B \times X \\ Y \times C - D \times X \end{pmatrix} \end{array}$$

The Kernel of M_ϕ is thus the vector space $\mathbb{V}(\mathbf{P},\mathbf{Q})$ above. Since M_ϕ has dimensions $2n^2 \times 2n^2$, then it seems that constructing it and determining its kernel should take time $\mathcal{O}\left(n^6\right)$. The matrix M_ϕ is however quite sparse, and quite structured, so that there are shortcuts allowing most operations to be faster than in the generic case. The matrix in fact has a nice expression in terms of tensor products (also called Kronecker products in the context of matrices).

Lemma 14.5. *M_ϕ has the following 2×2 block structure:*

$$M_\phi = \begin{pmatrix} I_n \otimes {}^tA & -B \otimes I_n \\ I_n \otimes {}^tC & -D \otimes I_n \end{pmatrix}$$

We omit the proof of this result, as it is both elementary and tedious. Here is an "expanded view" of what the matrix looks like:

$$\phi(X,Y) = \left(\begin{array}{ccc|ccc} {}^tA & & & -B_{11}\cdot I_n & \cdots & -B_{1n}\cdot I_n \\ & \ddots & & \vdots & \ddots & \vdots \\ & & {}^tA & -B_{1n}\cdot I_n & \cdots & -B_{nn}\cdot I_n \\ \hline {}^tC & & & -D_{11}\cdot I_n & \cdots & -D_{1n}\cdot I_n \\ & \ddots & & \vdots & \ddots & \vdots \\ & & {}^tC & -D_{1n}\cdot I_n & \cdots & -D_{nn}\cdot I_n \end{array}\right) \cdot \left(\begin{array}{c} X_{1\bullet} \\ \vdots \\ X_{n\bullet} \\ \hline Y_{1\bullet} \\ \vdots \\ Y_{n\bullet} \end{array}\right)$$

We will not dig too deeply into the issue of computing $\mathbb{V}$ as fast as possible, for several reasons:

a) It is rarely the most expensive step in practice.

b) Intelligent linear algebra codes will exploit the fact that ϕ_M has only $\mathcal{O}\left(n^3\right)$ non-zero entries instead of $\mathcal{O}\left(n^4\right)$.

c) One could always reduce the pencils $\mathbf{P}$ and $\mathbf{Q}$ to Kronecker Canonical Form in time $\mathcal{O}\left(n^3\right)$. We would then be dealing with a simplified case where A, B, C and D are (block-)diagonal, and ultra-sparse, with typically $\mathcal{O}(1)$ entries in each line (note that we would find conjugates of the solution, but this is not a problem).

14.5 Complexity of Generating The Resulting Polynomial Equations

We now discuss how to turn the following equation into an equivalent system $\mathcal{S}_{\text{quad}}$ of quadratic equations.

$$\left(\sum_{i=1}^{\ell} x_i \cdot T^{[i]}\right) \circ \mathbf{b} = \mathbf{a} \circ \left(\sum_{i=1}^{\ell} x_i \cdot S^{[i]}\right)$$

We will assume that $\mathbf{a}$ and $\mathbf{b}$ are homogeneous (and in any case, the linear and constant components are much easier to deal with). We will also assume that the quadratic forms $\mathbf{a}_i$ and $\mathbf{b}_i$ are represented by upper-triangular matrices. Our starting point is:

$$\sum_{i=1}^{\ell} x_i \cdot T^{[i]}_m \cdot \mathbf{b}_i(\mathbf{y}) = \mathbf{a}_m\left(\sum_{i=1}^{\ell} x_i \cdot S^{[i]} \cdot \mathbf{y}\right)$$

We will denote by $A^{[m]}$ (resp. $B^{[m]}$) the (upper-triangular) matrix representation of $\mathbf{a}_m$ (resp. $\mathbf{b}_m$). Expanding the definitions of $\mathbf{a}, \mathbf{b}$ and the matrix-vector product with $\mathbf{y}$ yields:

$$\sum_{u=1}^{n}\sum_{v=u}^{n}\left(\sum_{r=1}^{\ell}\left[\sum_{i=1}^{n} T^{[r]}_{mi}\cdot B^{[i]}_{uv}\right]x_r\right)\cdot \mathbf{y}_u\mathbf{y}_v = \sum_{u=1}^{n}\sum_{v=1}^{n}\left(\sum_{r=1}^{\ell}\sum_{t=1}^{\ell}\left[\sum_{i=1}^{n}\sum_{j=i}^{n} A^{[m]}_{ij}\cdot S^{[r]}_{iu}\cdot S^{[t]}_{jv}\right]\cdot x_r x_t\right)\cdot \mathbf{y}_u\mathbf{y}_v$$

We observe that the nested structure of $\mathbb{F}_q[x_1, \ldots, x_\ell][\mathbf{y}_1, \ldots, \mathbf{y}_n]$ is clearly visible. In any case, identifying the coefficients of the $\mathbf{y}_u\mathbf{y}_v$ monomials yields two sets of equations:

$$\begin{aligned}
\mathcal{S}_{\text{quad}} &= \left\langle \sum_{r=1}^{\ell} \left(\sum_{i=1}^{n} T_{mi}^{[r]} \cdot B_{uu}^{[i]} \right) x_r - \sum_{r=1}^{\ell} \sum_{t=1}^{\ell} \left(\sum_{i=1}^{n} \sum_{j=i}^{n} S_{iu}^{[r]} \cdot A_{ij}^{[m]} \cdot S_{ju}^{[t]} \right) \cdot x_r x_t \right\rangle_{\substack{1 \leq u \leq n \\ 1 \leq m \leq n}} \\
\mathcal{S}'_{\text{quad}} &= \left\langle \sum_{r=1}^{\ell} \left(\sum_{i=1}^{n} T_{mi}^{[r]} \cdot B_{uv}^{[i]} \right) x_r - \sum_{r=1}^{\ell} \sum_{t=1}^{\ell} \left(\sum_{i=1}^{n} \sum_{j=i}^{n} A_{ij}^{[m]} \cdot \left(S_{iu}^{[r]} \cdot S_{jv}^{[t]} + S_{iv}^{[r]} \cdot S_{ju}^{[t]} \right) \right) \cdot x_r x_t \right\rangle_{\substack{1 \leq u < v \leq n \\ 1 \leq m \leq n}}
\end{aligned}$$

It seems that determining $\mathcal{S}'_{\text{quad}}$ would take time $\Omega\left(n^7\right)$ if done naively, which seems suboptimal. However, there is a lot of structure in these expressions, that begs to be exploited. We may indeed factor some computation out by observing that the innermost double-sum is in fact the expression of a given coefficient in a matrix-matrix-matrix triple product. To take advantage of this, we define $M^{[m,r,t]} = {}^t S^{[r]} \times A^{[m]} \times S^{[t]}$, and we find out:

$$\begin{aligned}
\mathcal{S}_{\text{quad}} &= \left\langle \sum_{r=1}^{\ell} \left(\sum_{i=1}^{n} T_{mi}^{[r]} \cdot B_{uu}^{[i]} \right) x_r - \sum_{r=1}^{\ell} \sum_{t=1}^{\ell} M_{uu}^{[m,r,t]} \cdot x_r x_t \right\rangle_{\substack{1 \leq u \leq n \\ 1 \leq m \leq n}} \\
\mathcal{S}'_{\text{quad}} &= \left\langle \sum_{r=1}^{\ell} \left(\sum_{i=1}^{n} T_{mi}^{[r]} \cdot B_{uv}^{[i]} \right) x_r - \sum_{r=1}^{\ell} \sum_{t=1}^{\ell} \left(M_{uv}^{[m,r,t]} + M_{vu}^{[m,r,t]} \right) \cdot x_r x_t \right\rangle_{\substack{1 \leq u < v \leq n \\ 1 \leq m \leq n}}
\end{aligned}$$

At this point, the equations can just be obtained by dispatching the coefficients of the $M^{[m,r,t]}$ matrices. So, $\mathcal{S}_{\text{quad}}$ can be computed in time $\mathcal{O}\left(\ell^2 \cdot n^4\right)$, and we really hope that this is $\mathcal{O}\left(n^6\right)$.

Chapter 15

Simultaneous Equivalence of Quadratic Forms

In this chapter we discuss the algorithm we have designed for the QFSE problem. The results of this chapter have been the object of an article presented at PKC'11 [BFFP11], along with those of the next chapter.

Recall that an instance of the QFSE problem is formed by two vectors $\mathbf{a}$ and $\mathbf{b}$ of u quadratic forms over $\mathbb{F}_q[x_1, \ldots, x_n]$, along with the promise that there exist $S \in \mathrm{GL}_n(\mathbb{F}_q)$ such that:

$$\mathbf{b} = \mathbf{a} \circ S, \tag{15.1}$$

Solving the QFSE instance means solving the above equation in S, *i.e.*, finding all the possible bijective changes of coordinates transforming $\mathbf{a}$ in $\mathbf{b}$.

In this chapter, we present a pencil-based algorithm for QFSE, and we try to understand its behavior. This algorithm is deterministic and returns all the possible solutions. On the practical side, it is quite efficient, and empirically solves classes of random instance of the problem in time polynomial in n. Its most distinctive feature compared to the prior state of the art is that this polynomial behavior persists when $u = 2$ in some cases. For instance, we are able to solve the case $q = 2, u = 2, n = 128$ in about one hour. We have seen in chapter 13 that all the other algorithms have an exponential complexity in that case.

The cryptographic implication of these result is that using the QFSE-based identification scheme with random instances is no longer secure, as the secret-key would be recovered in very practical time from the public-key. Unfortunately, the hope that random instances of the problem were hard was the main motivation for suggesting to replace GI by QFSE in the identification scheme of Goldreich, Micali and Wigderson [GMW86]. However, it might remain possible to choose hard instances of the problem, while still getting an advantage over GI.

On the theoretical side, we try to justify this empirically good behavior of the algorithm. We study its complexity under the assumption that the input instance has been uniformly chosen at random amongst the set of instances that have a solution. We are aware that there must be hard instances that the algorithm is not capable of solving in polynomial time (otherwise, we would have $\mathsf{GI} \in \mathbf{P}$). We nevertheless indentify a (large) class of instances over which the algorithm should perform well, and then estimate the probablity that a random instance belongs to this class. In these favorable cases the problem reduces to that of solving a system of quadratic equations. This in turn appears to be surprisingly tractable for wide and interesting ranges of parameters.

15.1 Specializing the Pencil Strategy

We wish to recover the change of coordinates S between $\mathbf{a} = (\mathbf{a}_1, \ldots, \mathbf{a}_u)$ and $\mathbf{b} = (\mathbf{b}_1, \ldots, \mathbf{b}_u)$. Let $A_1, \ldots, A_u, B_1, \ldots, B_u$ be matrix representations of the respective quadratic forms, and let us define $M_i = B_i - {}^tS \times A_i \times S$. Then by definition, (15.1) is equivalent to saying that M_i is skew-symmetric for all $1 \leq i \leq u$. This can be reformulated in an equivalent way in saying that for all i, $M_i + {}^tM_i = 0$ (and in addition the diagonal coefficients must be zero). And, by our definition of the polar form, we find that (15.1) is equivalent, regardless of the characteristic of the field, to:

$$\begin{cases} \mathcal{P}(\mathbf{b}_1) &=& {}^tS \times \mathcal{P}(\mathbf{a}_1) \times S \\ \quad\vdots & \vdots & \vdots \\ \mathcal{P}(\mathbf{b}_u) &=& {}^tS \times \mathcal{P}(\mathbf{a}_u) \times S \end{cases} \tag{15.2}$$

At this point, let us observe that if $u \geq 3$, then (15.2), is easily converted (by left-multipling by ${}^tS^{-1}$) into a very overdetermined system of $u \cdot n^2$ linear equations in the $2n^2$ coefficients of S, S^{-1}, and these equations are extremely likely to have only one solution. Thus, roughly 15 years after its introduction, we find out with about 10 lines of elementary algebraic manipulations that QFSE with $u \geq 3$ can be solved in time $\mathcal{O}\left(n^6\right)$.

We will therefore focus our attention on the case where $u = 2$, which seem to be the hardest one, and which is the most cryptographically relevant one. When $u = 2$, then for any value of q, equation (15.2) becomes:

$$\begin{cases} \mathcal{P}(\mathbf{b}_1) &= {}^tS \times \mathcal{P}(\mathbf{a}_1) \times S \\ \mathcal{P}(\mathbf{b}_2) &= {}^tS \times \mathcal{P}(\mathbf{a}_2) \times S \end{cases}$$

This looks very much like a pencil equivalence problem, and it can be reformulated as such:

$${}^tS^{-1} \times \left(\underbrace{\lambda \cdot \mathcal{P}(\mathbf{b}_1) + \mu \cdot \mathcal{P}(\mathbf{b}_2)}_{\mathbf{Q}} \right) = \left(\underbrace{\lambda \cdot \mathcal{P}(\mathbf{a}_1) + \mu \cdot \mathcal{P}(\mathbf{a}_2)}_{\mathbf{P}} \right) \times S \tag{15.3}$$

Theorem 14.3 tells us that if the pencils are regular, then equation (15.3) admits at least q^n solutions, which makes it hopeless to try them one-by-one. We note in passing that this contradicts the idea more-or-less clearly expressed in [PGC98b, section 9].

Let us denote by $\mathbb{V}$ the vector space spanned by the solutions of (15.3), and let $\ell = \dim \mathbb{V}$ and let us consider $\left(S^{[1]}, U^{[1]}\right), \ldots, \left(S^{[\ell]}, U^{[\ell]}\right)$, a basis of $\mathbb{V}$. Let us also consider the ring $\mathbb{F}_q[x_1, \ldots, x_\ell]$. We set:

$$\widetilde{S} = \sum_{i=1}^{\ell} x_i \cdot S^{[i]}$$

There is then a one-to-one correspondance between the solutions of (15.1) and those of:

$$\mathbf{b} = \mathbf{a} \circ \widetilde{S}$$

This last equation is a set of equality between polynomials and following the idea of Faugère and Perret, we identify the two sides of the equation coefficient-wise, and this gives a set $\mathcal{S}_{\text{quad}}$ of of $n(n+1)$ quadratic equations in $x_1, \ldots, x_\ell$. To recover all the solutions of (15.1), we find all the solutions of $\mathcal{S}_{\text{quad}}$ by computing a Gröbner basis of the ideal it spans, and we determine the corresponding variety. Algorithm 15.1 shows a pseudo-code of the method we outlined.

Algorithm 15.1 Pseudo-code of the pencil-based QFSE algorithm.

1: **function** Pencil-QFSE($\mathbf{a}, \mathbf{b}$)
2: **let** M_ϕ denote the $2n^2 \times 2n^2$ matrix representing:

$$\begin{array}{rccc} \phi: & (\mathbb{F}_q)^{2n^2} & \to & (\mathbb{F}_q)^{2n^2} \\ & (X, Y) & \to & \begin{pmatrix} Y \times B_1 - A_1 \times X \\ Y \times B_2 - A_2 \times X \end{pmatrix} \end{array}$$

3: **let** $\mathbb{V} = \ker M_\phi$
4: **let** $\ell = \dim \mathbb{V}$ **and** $\left(S^{[i]}, U^{[i]}\right)_{1 \le i \le \ell}$ be a basis of $\mathbb{V}$
5: **consider** the two rings $R_\perp \leftarrow \mathbb{F}_q[x_1, \ldots, x_\ell]$ **and** $R^\top \leftarrow \mathbb{F}_q[x_1, \ldots, x_\ell][y_1, \ldots, y_n]$
6: **let** $\widetilde{S} = \sum_{i=1}^{\ell} x_i \cdot S^{[i]}$
7: **let** $\mathbf{y} = (y_1, \ldots, y_n)$
8: **let** $\mathbf{z}(\mathbf{y}) = \mathbf{b}(\mathbf{y}) - \mathbf{a}\left(\widetilde{S} \cdot \mathbf{y}\right)$ ▷ $\mathbf{z}$ belongs to $\left(R^\top\right)^2$
9: **let** $\mathcal{S}_{\text{quad}}$ be the set of coefficients of $\mathbf{z}_1$ and $\mathbf{z}_2$ ▷ $\mathcal{S}_{\text{quad}} \subseteq R_\perp$
10: **let** $\mathcal{I}$ be the ideal of $R_\perp$ spanned by $\mathcal{S}_{\text{quad}}$
11: **return** the image of $\mathbf{V}(\mathcal{I})$ by $\psi : (u_i)_{1 \le i \le \ell} \mapsto \sum_{i=1}^{\ell} u_i \cdot S^{[i]}$
12: **end function**

Sketch of a Complexity Analysis. The only step of the algorithm whose complexity is not easily predictible is the Gröbner basis computation involved in determining the solutions of $\mathcal{S}_{\text{quad}}$. An additional problem is that while there is always $n(n+1)$ equations in $\mathcal{S}_{\text{quad}}$, the number of variables depends on the choice of the instance.

To analyze our algorithm, we will consider $\dim \mathbb{V}$ as a random variable, taken over the random choice of the input. We will try to obtain enough information about its distribution to argue that the Gröbner basis computation should terminate quickly with high probability. The key tool to obtain information on $\dim \mathbb{V}$ is theorem 14.3, but it only applies if the pencils are regular. We therefore begin by investigating the probability that it is the case. However, several steps of the analysis differ according to the parities of q and n.

15.2 Complexity Analysis for Odd q

De la musique avant toute chose,
Et pour cela préfère l'Impair
Plus vague et plus soluble dans l'air,
Sans rien en lui qui pèse ou qui pose.

Art Poétique, Paul Verlaine

It now remains to estimate the complexity of the Gröbner basis computation. For this purpose, we need to investigate the distribution of $\dim \mathbb{V}$ for our special case.

Regularity of the Pencils. We note that as soon as one of the polar form has full rank, then the pencil is regular. As seen is section 10.7.1, a (random) polar form is non-singular with probability $\lambda(q,n)/\lambda(q^2,n/2)$. We conclude that the probability that at least one of the polar form is non-singular is:

$$\mathbb{P}\left[\mathbf{P} \text{ is regular}\right] \geq 1 - \left(1 - \frac{\lambda(q,n)}{\lambda(q^2,n/2)}\right)^2$$

And asymptotically (using equation (10.4)) we find:

$$\lim_{n\to\infty} \mathbb{P}\left[\mathbf{P} \text{ is regular}\right] \geq 1 - q^{-2} - 2q^{-4} + 2q^{-5} + \mathcal{O}\left(q^{-6}\right) \tag{15.4}$$

Distribution of $\dim \mathbb{V}$. Under the assumption that the pencil is regular, we may make use of theorem 14.3. This will boil down to analyzing the matrix $\mathcal{C}$ occuring in the statement of the theorem. We will assume w.l.o.g. that $\mathcal{P}(\mathbf{a}_1)$ is invertible, so that $\mathcal{C} = \mathcal{P}(\mathbf{a}_2) \times \mathcal{P}(\mathbf{a}_1)^{-1}$. Since the inverse of a symmetric matrix is also a symmetric matrice, then $\mathcal{C}$ is the product of two random symmetric matrices (one of them being invertible). Unfortunately, $\mathcal{C}$ is provably not a uniformly distributed matrix: it has the rank distribution of symmetric matrices (given by theorem 10.28), while if it were random it would have the "normal" rank distribution (given by equation (10.5)). So, unfortunately, the nice results regarding the invariant factors of random matrices cannot be used.

One could hope that the distribution of $\dim \mathbb{V}$ we obtain is close to what it is when $\mathcal{C}$ is uniformly chosen at random. We compared the two distributions experimentally, and Figure 15.1 shows the outcome of our little experience. While the probability that $\dim \mathbb{V} = n$ seems close in the two cases, things are visibly not as nice for the other values of $\dim \mathbb{V}$. We will therefore focus our attention where we might expect a nice behavior, and we consider the two sequences:

$$\begin{aligned} u_q &= \mathbb{P}\left[\dim \mathbb{V} = n \;\middle|\; \mathbf{P} \text{ from algorithm (15.2), and regular}\right] \\ v_q &= \mathbb{P}\left[\dim \mathbb{V} = n \;\middle|\; \mathbf{P} \text{ chosen at random over } \mathbb{F}_q \text{ (amongst regular pencils)}\right] \end{aligned}$$

Fig 15.2a and 15.2b show that u_q and v_q are quite close. What is more surprising is the great precision with which we could verify that:

$$v_q - u_q \approx \frac{1}{q^2} - \frac{1}{q^3} - \frac{1}{q^4} + \frac{1}{q^5}$$

This, in conjunction with lemma 14.4 (giving the expression of v_q) lead us to conjecture an expression for the probability of $\dim \mathbb{V} = n$ in the algorithm.

Conjecture 15.1. In algorithm 15.1, when the pencils are regular, then the probability that $\mathcal{C}$ is cyclic (which is also the probability that $\dim \mathbb{V} = n$) is:

$$\mathbb{P}\left[\dim \mathbb{V} = n \;\middle|\; \mathbf{P} \text{ is regular}\right] = 1 - \frac{1}{q^2} - \frac{3}{q^5} + \mathcal{O}\left(q^{-6}\right)$$

We checked that this conjecture holds for $q \geq 3$ (q odd) up to a precision of about 10^{-4} (which is in fact smaller than 3^{-6}).

Putting Things Together. So, we may now estimate the probability that $\dim \mathbb{V} = n$ over the random choice of the instance. we find:

$$\begin{aligned} \mathbb{P}\left[\dim \mathbb{V} = n\right] &= \mathbb{P}\left[\dim \mathbb{V} = n \;\middle|\; \mathbf{P} \text{ is regular}\right] \cdot \mathbb{P}\left[\mathbf{P} \text{ is regular}\right] \\ &\geq 1 - \frac{2}{q^2} - \frac{1}{q^4} - \frac{1}{q^5} + \mathcal{O}\left(\frac{1}{q^6}\right) \end{aligned}$$

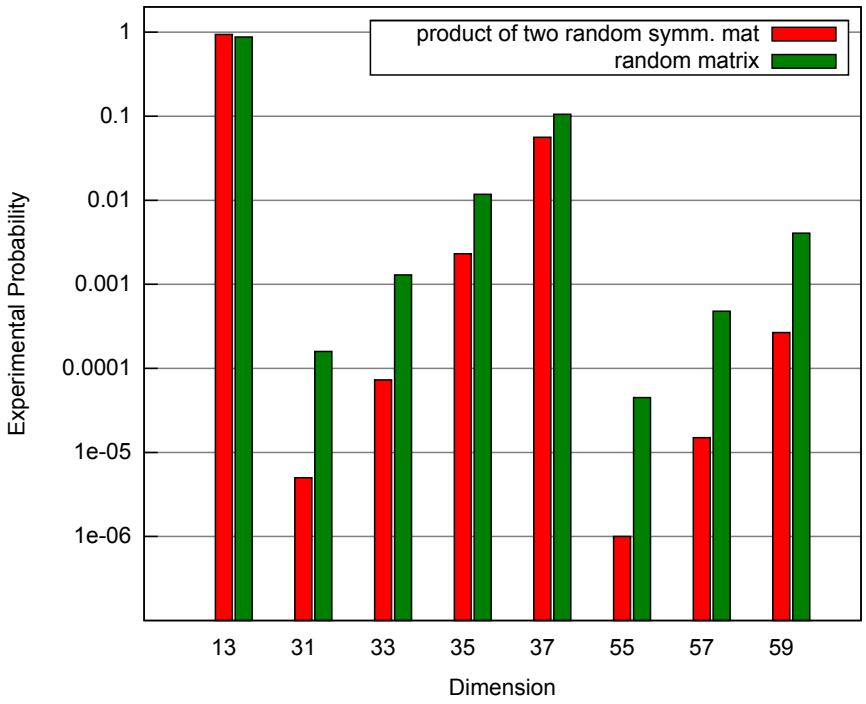
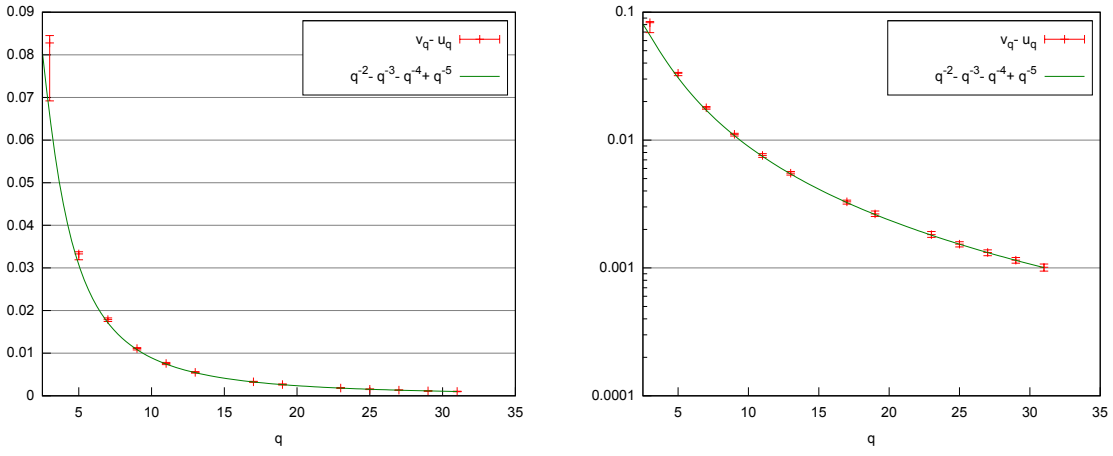


Figure 15.1: Distribution of $\dim \mathbb{V}$ when $\mathcal{C}$ is generated at random vs. when $\mathcal{C}$ is the product of a non-singular random symmetric matrix and a random symmetric matrix. The parameters were $n = 13$ and $q = 3$. A discrepancy is visible, but the log-scale may make it look worse than it really is.



(a) Normal scale (b) Log scale

Figure 15.2: Comparison between the probability of $\mathcal{C}$ being cyclic in algorithm 15.1 (u_q) and the probability that a random matrix is cyclic (v_q). All the values of $2 \leq n \leq 32$ have been considered; the graphs shows the minimum, the average, and the maximum over the range of n.

q	2	4	8	16	32
exp. prob.	0.9577	0.9989	0.99997	0.9999988	0.9999999
$1-q^{-4}$	0.9375	.9961	0.9998	0.99998	0.999999

Table 15.1: Experimentally observed probability that $\dim \mathbb{V} = 2n$ under the condition that the pencils are regular. The experiment involves one million random trials.

15.3 Analysis for even q

When q is even, the situation is quite different, and overally more complicated. Things start to go bad when n is odd. In that case, the prencil is *singular*, since $\mathbf{P}_{\lambda,\mu}$ is a skew-symmetric matrix (and thus cannot have rank n, when n is odd, in virtue of lemma 10.1).

We will therefore focus on the nicer case where n is even, and try to replay the analysis of the previous section. We have seen in §10.7.1 that the probability that a random polar form is invertible is the same as when q is odd. Therefore, the result of equation (15.4) still holds when n and q are even.

The next problem we have to deal with is the distribution of $\dim \mathbb{V}$. Earlier we empirically studied the probability that $\mathcal{C}$ is cyclic to obtain an interesting result. This time, when q is even, $\mathcal{C}$ is the product of two *skew-symmetric* matrices. Things get from bad to worse, because the product of two skew-symmetric matrices is *very far* from being random, and $\mathcal{C}$ is in fact cyclic with probability zero. We also observed that $\dim \mathbb{V}$ was always greater than $2n$ (and that this also was the most frequent value). This phenomenon puzzled us quite a bit, but it turns out to be a consequence of a property of the product of skew-symmetric matrices. The following theorem was stated (in a particular case) in 1919 by Bennet [Ben19], and then in a more general setting by Stenzen in 1922 [Ste22]. According to [IF09], from where we quote it, it has been rediscovered in 1952, in 1974, in 1984, in 1991 and in 1997.

Theorem 15.1 (Bennet, Stenzen *et al.*)**.** *An $n \times n$ matrix M is the product of two skew-symmetric matrices if and only if the following conditions are fulfilled:*

a) Every elementary divisor[1] *corresponding to a non-zero eigenvalue of M occurs an even number of times.*

b) Let $\lambda^{m_1}, \lambda^{m_2}, \ldots, \lambda^{m_s}$ be the elementary divisors of M corresponding to its zero eigenvalue and let:

$$m_1 \geq m_2 \geq \cdots \geq m_s$$

Then:

$$m_{2i-1} - m_{2i} \leq 1, \quad i = 1, 2, \ldots$$

(if s is odd, we set $m_{s+1} = 0$).

While it is not completely obvious, this theorem implies that the minimal polynomial of the product of two skew-symmetric matrices has degree at most $n/2$, and that the characteristic polynomial is a square. Here is the relevant consequence in our setting.

Corollary 15.2. *If q and n are even, then* $\dim \mathbb{V} \geq 2n$.

Proof. Since $\mathcal{C}$ is the product of two skew-symmetric matrices, we know that $\mathcal{C}$ meets the condition of theorem 15.1. We now argue that $\mathcal{C}$ has at least two invariant factors. If the minimal polynomial of $\mathcal{C}$ is not x^k for some k, then by condition a), it occurs at least twice. If the minimal polynomial is x^k, then it cannot be the only invariant factor, because of condition b).

So, the minimal polynomial appears twice in the invariant factors of $\mathcal{C}$. If there are no other invariant factors, then it is of degree $n/2$, and theorem 14.3 tells us that $\dim \mathbb{V} = 2n$. If $\mathcal{C}$ has more invariants, $\dim \mathbb{V}$ can only get higher. □

What we would in fact need to know is the probability that $\mathbb{V}$ is exactly of dimension $2n$. According to the proof of corollary 15.2, this would mean that $\mathcal{C}$ has only two similarity invariants, which implies that it has a minimal polynomial of maximal degree (a property reminiscent of cyclic matrices). It seems plausible that $\mathcal{C}$ is unlikely to have a very high number of similarity invariants. We could not compute explicitly this probability that $\dim \mathbb{V} = 2n$.

We measured it experimentally, under the assumption that the pencils are regular. The results are shown in table 15.1, and lead us to state the following conjecture:

1. A quick reminder on elementary divisors is available in §10.1

Conjecture 15.2. When q and n are even, then:

$$\mathbb{P}\big[\dim \mathbb{V} = 2n \;\big|\; \mathbf{P} \text{ is regular}\big] \geq 1 - \frac{1}{q^4}$$

To conclude, we assemble the pieces:

$$\begin{aligned}\mathbb{P}\left[\dim \mathbb{V} = 2n\right] &= \mathbb{P}\big[\dim \mathbb{V} = 2n \;\big|\; \mathbf{P} \text{ is regular}\big] \cdot \mathbb{P}\left[\mathbf{P} \text{ is regular}\right] \\ &\geq 1 - \frac{1}{q^2} - \frac{3}{q^4} + \mathcal{O}\left(\frac{1}{q^5}\right)\end{aligned}$$

So, the best we can hope for, $\dim \mathbb{V} = 2n$, hopefully happens frequently.

15.4 Solving the Quadratic Equations

> L'ignorance qui se sait, qui se juge et qui se condamne, ce n'est pas une entière ignorance : pour l'être, il faut qu'elle s'ignore soi-même.
>
> *Essais*, Montaigne

If $\mathbf{a}$ and $\mathbf{b}$ are represented by upper-triangular matrices $A^{[1]}, A^{[2]}, B^{[1]}$ and $B^{[2]}$, then following what we did in §14.5, we let $M^{[i,r,t]} = {}^tS^{[r]} \times A^{[i]} \times S^{[t]}$, and we find out:

$$\begin{aligned}\mathcal{S}_{\text{quad}} &= \left\langle B^{[i]}_{uu} - \sum_{r=1}^{\ell}\sum_{t=1}^{\ell} M^{[i,r,t]}_{uu} \cdot x_r x_t \right\rangle_{\substack{1\leq u\leq n \\ i=1,2}} \\ \mathcal{S}'_{\text{quad}} &= \left\langle B^{[i]}_{uv} - \sum_{r=1}^{\ell}\sum_{t=1}^{\ell} \left(M^{[i,r,t]}_{uv} + M^{[i,r,t]}_{vu}\right) \cdot x_r x_t \right\rangle_{\substack{1\leq u<v\leq n \\ i=1,2}}\end{aligned}$$

Observe that the equations can be obtained by dispatching the coefficients of the $M^{[i,r,t]}$ matrices, so that these equations be computed in time $\mathcal{O}\left(\ell^2 \cdot n^3\right)$, and we argued that this is $\mathcal{O}\left(n^5\right)$ with high probability. It follows from the previous reasonings that there are essentially three different scenarios to discuss:

a) q odd, $\ell \approx n$

b) q even, n even, $\ell \approx 2n$

c) q even, n odd, $\ell \approx n + 1$

In all these cases, $\mathcal{S}_{\text{quad}}$ should be composed of $n(n+1)$ quadratic equations in $\mathcal{O}(n)$ variables, and we would expect to be able to solve this very overdetermined system of quadratic equations either by plain linearization, or via a very quick Gröbner basis computation. This implicitly assumes that the quadratic equations in $\mathcal{S}_{\text{quad}}$ and $\mathcal{S}'_{\text{quad}}$ are *linearly independent*, but it makes sense to challenge this crucial assumption.

And this assumption turns out to be plainly false. In all cases, there are in fact only $\mathcal{O}(n)$ linearly independent equations in $\mathcal{S}_{\text{quad}} + \mathcal{S}'_{\text{quad}}$, as Table 15.2 shows. This means that we cannot argue that solving these equations is doable in polynomial time. An explanation of this phenomenon has eluded us so far. In particular, we have not been able to directly determine a basis of the vector space spanned by the equations.

However, when q is odd, it seems that in most cases, the linear spans of $\mathcal{S}_{\text{quad}}$ and $\mathcal{S}'_{\text{quad}}$ are equal. When q is even, these are two distinct linear spaces. Fully understanding this phenomenon is a fascinating subject for future work.

If the quadratic equations in $\mathcal{S}_{\text{quad}} + \mathcal{S}'_{\text{quad}}$ were semi-regular, then solving them would be plainly exponential. We note that even in this (pessimistic) setting, then our algorithm would still outperform all its predecessors.

Fortunately, the equations in $\mathcal{S}_{\text{quad}} + \mathcal{S}'_{\text{quad}}$ are *much easier* to solve than semi-regular equations. When n is small enough, then the Gröbner basis computation would be feasible even with semi-regular equations. However, when q is small, the maximal degree reached by polynomials during the Gröbner basis computation is unexpectedely small. Table 15.3 shows the corresponding complexities. For instance, when $q = 2$ and $n = 128$, we are solving a system of 256 quadratic equations in 256 variables over $\mathbb{F}_2$. When the equations are random, this is completely infeasible. In our case, it just takes 3 minutes! We have no clear explanation of this phenomenon. It appears that many linear equations appear in the first steps of the Gröbner basis computation, thus reducing the complexity of the whole process dramatically.

scenario	n	q	ℓ	# independent equations in $\mathcal{S}_{\text{quad}} + \mathcal{S}'_{\text{quad}}$	comment
a	8	17	8	8	
a	16	17	16	16	
a	27	3	27	27	
a	27	3	29	28	$n + \frac{1}{2}(\ell - n)$
a	27	3	31	29	
a	27	3	33	30	
a	27	3	35	31	
a	27	3	39	33	
b	8	256	16	16	
b	16	256	32	32	
b	32	2	64	80	
b	32	2	72	84	$2n + \frac{1}{2}(\ell - n)$
b	32	2	80	88	
b	32	2	88	92	

Table 15.2: Number of linearly independent equations in $\mathcal{S}_{\text{quad}}$, in the most favorable cases (*i.e.*, it can sometimes be less).

q	N	field equations	Time	Memory	Degree
2	80	yes	20s	2.8GB	4
2	128	yes	160s	15.8GB	4
3	80	yes	220s	2.GB	6
4	32	yes	19 595s	8GB	4
4	64	yes	out of memory	$\geq$ 100GB	?
5	20	yes	43s	151Mb	8
5	32	yes	31 714s	9GB	8
256	12	no	31 480s	374MB	10
256	12	no	420s	68M	4 $\star$
256	16	no	3898s	272MB	4 $\star$
65536	8	no	2.8s	11.5Mb	8
65537	8	no	8s	11Mb	10

Table 15.3: Complexity of computing a Gröbner basis of the ideal spanned by $\mathcal{S}_{\text{quad}}$ and $\mathcal{S}'_{\text{quad}}$.

For $q = 256$ and $n = 12, 16$, computing the Gröbner basis is feasible, but converting it to a suitable monomial order is *much* longer. We therefore decided to resort to hybrid solving, by enumerating all the values of a given variable and computing q Gröbner bases. This turn out to be much faster. In table 15.3, we marked with a star ($\star$) the lines where we used this technique. The table shows that for $n = 12, q = 256$ this yields an important speedup.

To conclude, the case where $q = 2, 3$ seems practical for any (reasonable) value of n. The cases where $n \leq 20$ also seems tractable, but in some cases the complexity of the whole process will be proportional to q.

15.5 Implementation and Practical Results

We implemented the algorithm described in this chapter using the computer algebra system MAGMA [BCP97]. The code (which is in the public domain) is available at:

`http://www.di.ens.fr/~bouillaguet/implem/qfse.magma`

This implementation is only 50 lines long. It breaks random instances of QFSE in very practical time when q or n are "small". For instance, all the challenges presented in §12.3 are solved in a few seconds. The (legacy) challenge A_1, in particular, is broken in 1.1s on a laptop. If we double n ("safer challenge" A_3), the running time of the algorithm goes up to 20s. MAGMA is apparently not particularly good at manipulating low-degree dense multivariate polynomials. When $q = 2$ and n is large, the running time of the implementation is dominated by the formation of $\mathcal{S}_{\text{quad}}$. Actually solving the resulting quadratic equations turns out to be easier than generating them.

In our implementation, the Gröbner basis is computed using the off-the-shelf implementation $\mathbf{F_4}$ [Fau99] algorithm present in MAGMA. Converting the Gröbner-basis to the lexicographic order is done using either the FGLM algorithm [FGLM93] or using the Gröbner Walk [CKM97], at MAGMA's own discretion.

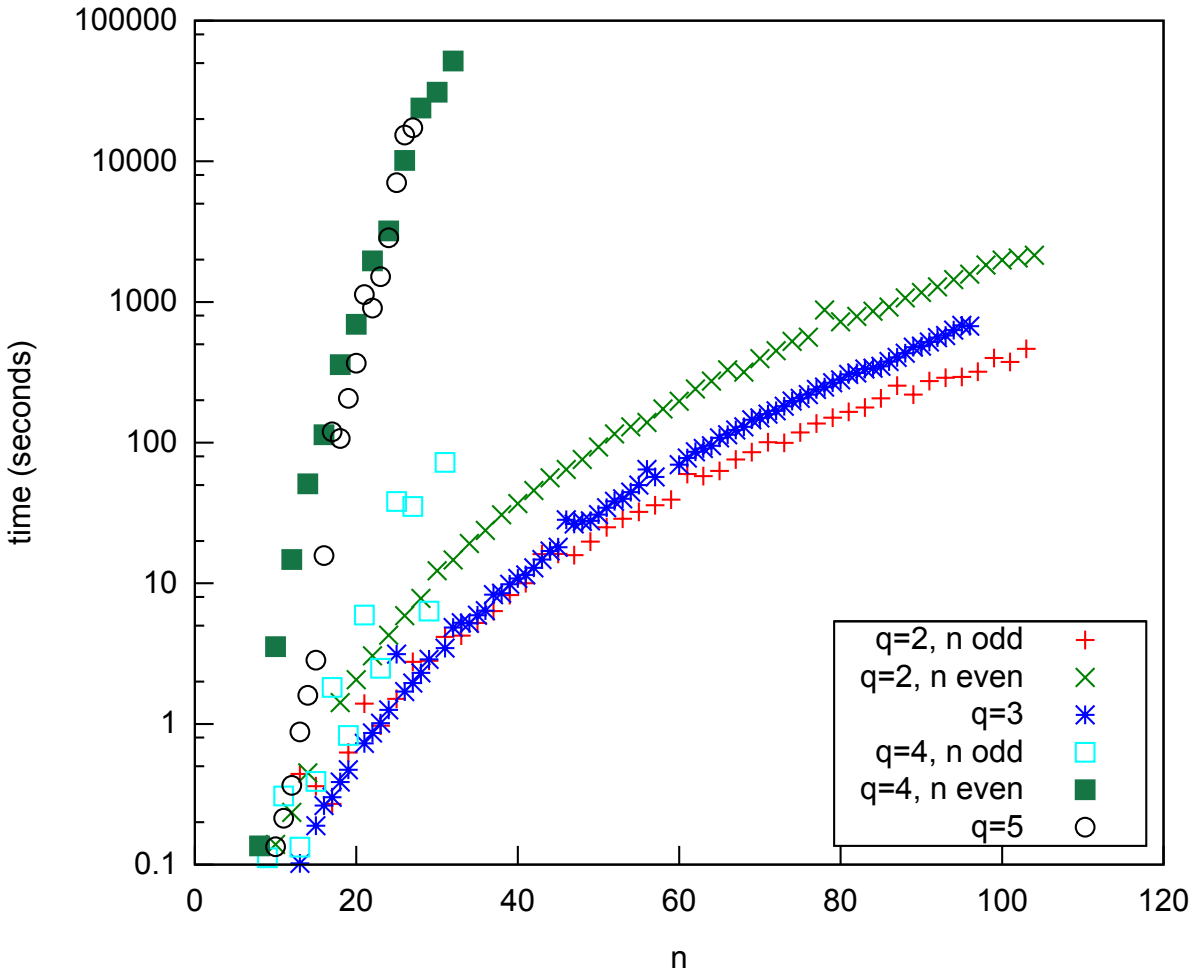


Figure 15.3: Time complexity of solving the QFSE problem for pairs of quadratic forms.

Chapter 16

Equivalence of Cubic Forms

This chapter presents an algorithm for the equivalent cubic forms problem. It is inspired by the method of the previous chapter, that cannot apply when polynomials are cubic. The original idea was found by Ludovic Perret while discussing the algorithm of the previous chapter. Later on, Ludovic Perret and Jean-Charles Faugère proved the main theorem. The results of this chapter, along with those of the previous one, were published at PKC'11 [BFFP11].

In this chapter, we focus on the case where $\mathbf{a}$ and $\mathbf{b}$ are composed of a single cubic polynomial:

$$\mathbf{a} = \sum_{i=1}^{n}\sum_{j=i}^{n}\sum_{k=j}^{n} A_{i,j,k} \cdot x_i x_j x_k, \qquad \mathbf{b} = \sum_{i=1}^{n}\sum_{j=i}^{n}\sum_{k=j}^{n} B_{i,j,k} \cdot x_i x_j x_k.$$

and we are promised that there exist an (unknown) invertible $n \times n$ matrix S such that:

$$\mathbf{b} = \mathbf{a} \circ S \tag{16.1}$$

The problem is to recover S given $\mathbf{a}$ and $\mathbf{b}$. The techniques developed previously for the quadratic case unfortunately cannot be directly applied in this setting. Indeed, cubic forms are no longer represented by matrices. However, it is still possible to define an equivalent of the polar form:

$$\phi_{\mathbf{f}}(\mathbf{x}, \mathbf{y}) = \mathbf{f}(\mathbf{x} + \mathbf{y}) - \mathbf{f}(\mathbf{x}) - \mathbf{f}(\mathbf{y})$$

This yields a symmetric "biquadratic" form (of total degree 3). There are no easily available matrix involving S here. However it is fairly obvious that for all vectors $\mathbf{x}$ and $\mathbf{y}$:

$$\phi_{\mathbf{b}}(\mathbf{x}, \mathbf{y}) = \phi_{\mathbf{a}}(S \cdot \mathbf{x}, S \cdot \mathbf{y})$$

Applying the Gröbner-based algorithm to this equation does not result in anything better than applying it directly to (16.1). However, if we set $\mathbf{y} = S^{-1} \cdot \mathbf{z}$, then we find:

$$\phi_{\mathbf{b}}(\mathbf{x}, S^{-1} \cdot \mathbf{z}) = \phi_{\mathbf{a}}(S \cdot \mathbf{x}, \mathbf{z}) \tag{16.2}$$

This equation is more interesting than the previous one, because the coefficients of S and S^{-1} occur with degree 2. Identifying coefficient-wise both sides of (16.2) results in $n^3 + n^2$ quadratic equations in $2n^2$ variables, to which we could add $2n^2$ quadratic equations expressing the fact that S^{-1} is both the left- and and right-inverse of S.

More formally, let R be the ring $\mathbb{F}_q[s_{1,1}, \ldots, s_{n,n}, u_{1,1}, \ldots, u_{n,n}]$. We consider the algebra $\mathcal{A}^s$ of all $n \times n$ matrices over R. Let $S = (s_{i,j})$ and $U = (u_{i,j})$ in $\mathcal{A}^s$ be symbolic matrices. We denote by $\mathcal{I}_{\mathbf{a},\mathbf{b}}$ the ideal generated by all the coefficients in R of the equations:

$$\begin{cases} \mathrm{D}\mathbf{a}(S \cdot \mathbf{x}, \mathbf{y}) - \mathrm{D}\mathbf{b}(\mathbf{x}, U \cdot \mathbf{y}) &= 0, \\ U \cdot S - 1_n &= 0_n, \\ S \cdot U - 1_n &= 0_n. \end{cases}$$

A possible solution is to directly compute a Gröbner basis of $\mathcal{I}_{\mathbf{a},\mathbf{b}}$. Under the (debatable) assumption that the equations spanning $\mathcal{I}_{\mathbf{a},\mathbf{b}}$ form a semi-regular sequence, the degree of regularity (and the projected complexity) can be derived from the results of §10.5. From fig. 16.1, it seems that the complexity of this Gröbner basis computation is of order $\mathcal{O}(8^n)$, which might be feasible for small values of n. Indeed, for $q = 16, n = 5$, computing a Gröbner basis of $\mathcal{I}_{\mathbf{a},\mathbf{b}}$ terminates in degree 3, after 6s of computation. When $n = 6$, the computation also terminate in degree 3, and takes 411s. When $n = 7$, it requires 17630s, uses 3GB of RAM, and still terminates in degree 3. However, when $n = 8$, after echelonizing a $555'829 \times 355'546$ matrix for a couple days, the Gröbner basis computation started crunching degree-4 polynomials, and died after using the 74Gbytes of available RAM.

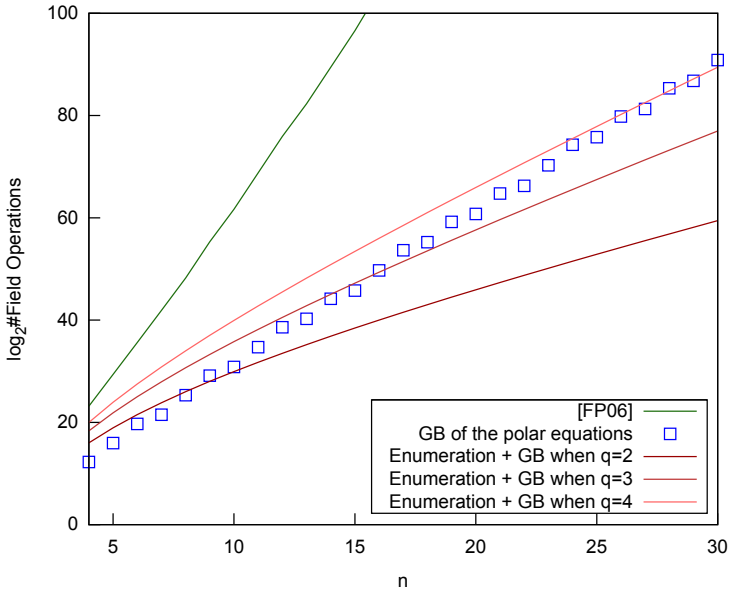


Figure 16.1: Comparison between the various possible methods, including: running the Gröbner-based algorithm on (16.1), computing a Gröbner-basis of $\mathcal{I}_{\mathbf{a},\mathbf{b}}$, and enumerating all the possible first rows of S, then computing a Gröbner basis of $\mathcal{I}_{\mathbf{a},\mathbf{b}} + \mathcal{J}$. The complexity of computing a Gröbner Basis of $\mathcal{I}_{\mathbf{a},\mathbf{b}}$ seems to be $\mathcal{O}(8^n)$.

It turns out that if a little bit of information about the final solution is available, then computing a Gröbner basis of $\mathcal{I}_{\mathbf{a},\mathbf{b}}$ becomes much easier. Let the $n \times n$ matrix $\widetilde{S} = (\widetilde{s}_{ij})$ over $\mathbb{F}_q$ be a particular solution of (16.1), and let $\mathcal{J}$ denote the following ideal:

$$\mathcal{J} \left\langle s_{1j} - \widetilde{s}_{1j} \mid j = 1, \ldots, n \right\rangle.$$

If the first row of the particular solution $\widetilde{S}$ is known, then the ideal $\mathcal{J}$ is known, and we may try to compute a Gröbner basis of $\mathcal{I}_{\mathbf{a},\mathbf{b}} + \mathcal{J}$. In fact, this Gröbner basis computation is extremely efficient.

Theorem 16.1. *The degree of regularity of $\mathcal{I}_{\mathbf{a},\mathbf{b}} + \mathcal{J}$ is 2. Therefore, computing a Gröbner basis of this ideal takes time $\mathcal{O}\left(n^6\right)$.*

Before giving the proof of this theorem, we discuss its implications. We could enumerate all the possible first rows of $\widetilde{S}$, and then for each of them compute a Gröbner basis of $\mathcal{I}_{\mathbf{a},\mathbf{b}}$. If the resulting ideal is non-empty, then we have found an actual solution of the problem. This requires $\mathcal{O}\left(q^n \cdot n^6\right)$ field operations to solve the whole instance. For small values of q, this may be asymptotically faster than computing a Gröbner basis of $\mathcal{I}_{\mathbf{a},\mathbf{b}}$, as Figure 16.1 suggests. Moreover, it is easily parallelizable, and requires only a modest amount of memory, which makes it more practical.

The biggest proposed cubic CFE challenge (Challenge C in Table 12.2) has $u = 1$, $n = 16$ and $q = 2$. Given that $\mathcal{J}$ is known, the computation of a Gröbner basis of $\mathcal{I}_{\mathbf{a},\mathbf{b}} + \mathcal{J}$ takes 90 seconds on a 2.8Ghz Xeon CPU using the publicly available implementation of F_4 in MAGMA. Since this has to be repeated 2^{15} times, we conclude that the whole process should take about one CPU-month, so that it can be carried out in practice.

16.1 Proof of the Theorem

We use the fact, established by Lazard in 1983 [Laz83], that the degree of regularity of an ideal is generically left invariant by any linear change of the variables or generators. In particular, we consider the ideal $\mathcal{I}'_{\mathbf{a},\mathbf{b}}$ generated by all the coefficients in $\mathbb{K}[x_1, \ldots, x_n, y_1, \ldots, y_n]$ of the equations:

$$\mathrm{D}\mathbf{a}(S_0(S + I_n)\mathbf{x}, \mathbf{y}) - \mathrm{D}\mathbf{b}(\mathbf{x}, (U + I_n)S_0^{-1}\mathbf{y}) = 0, \qquad U \cdot S = 0_n, \qquad S \cdot U = 0_n.$$

It is clear that $\mathcal{I}'_{\mathbf{a},\mathbf{b}}$ is obtained from $\mathcal{I}_{\mathbf{a},\mathbf{b}}$ by replacing S (resp. U) by $S_0(I_n + S)$ (resp. $(U + I_n)S_0^{-1}$). Thus, the degree of regularity of $\mathcal{I}'_{\mathbf{a},\mathbf{b}}$ and $\mathcal{I}_{\mathbf{a},\mathbf{b}}$ are equal. Using the same transformation, the ideal $\mathcal{J}$ becomes

$$\mathcal{J}' = \left\langle s_{1,j} \mid j = 1, \ldots, n \right\rangle.$$

We now estimate the degree of regularity of the ideal $\mathcal{I}'_{\mathbf{a},\mathbf{b}} + \mathcal{J}'$. For a reason which will become clear in the sequel, it is more convenient to work with $\mathcal{I}'_{\mathbf{a},\mathbf{b}} + \mathcal{J}'$. In what follows, F will denote the set of the generators of $\mathcal{I}'_{\mathbf{a},\mathbf{b}} + \mathcal{J}'$. We will show that many new linear equations appear when considering equations of degree 2. To formalize this, we introduce some definitions related to the $\mathbf{F_4}$ algorithm [Fau99]. In particular, we will denote by $I_{d,k}$ the linear space generated during the k-th step of $\mathbf{F_4}$ when considering polynomials of degree d.

Definition 16.1. We have the following recursive definition of $I_{d,k}$:

$$\begin{aligned} I_{d,0}(F) &= \langle F \rangle \\ I_{d,1}(F) &= \langle s_{i,j} \cdot f \mid 1 \leqslant i,j \leqslant n \text{ and } f \in I_{d,0}(F) \rangle + \langle u_{i,j} \cdot f \mid 1 \leqslant i,j \leqslant n \text{ and } f \in I_{d,0}(F) \rangle \\ I_{d,k}(F) &= \langle s_{i,j} \cdot f, u_{i,j} \cdot f \mid 1 \leqslant i,j \leqslant n \text{ and } f \in I_{d,k-1}(F) \text{ and } \deg(f) \leq d-1 \rangle \end{aligned}$$

Roughly speaking, the index k is the number of steps in the $\mathbf{F_4}$/ Ffive algorithm to compute an element $f \in I_{d,k}(F)$. We show that $I_{2,1}(F)$ contains exactly $n^2 + 2n$ linear equations. This means that we have already many linear equations generated during the first step of a Gröbner basis computation of F.

Lemma 16.2. *$I_{2,1}(F)$ contains the following linear equations:*

$$\{u_{1,j} \mid j = 1, \ldots, n\}. \tag{16.3}$$

Proof. From the first row of the following zero matrix $S \cdot U$ we obtain the following equations:

$$\begin{cases} s_{1,1}\, u_{1,1} + s_{1,2}\, u_{2,1} + s_{1,3}\, u_{3,1} + \cdots + s_{1,n}\, u_{n,1} = 0, \\ s_{1,1}\, u_{1,2} + s_{1,2}\, u_{2,2} + s_{1,3}\, u_{3,2} + \cdots + s_{1,n}\, u_{n,2} = 0, \\ s_{1,1}\, u_{1,3} + s_{1,2}\, u_{2,3} + s_{1,3}\, u_{3,3} + \cdots + s_{1,n}\, u_{n,3} = 0, \\ \quad \cdots \\ s_{1,1}\, u_{1,n} + s_{1,2}\, u_{2,n} + s_{1,3}\, u_{3,n} + \cdots + s_{1,n}\, u_{n,n} = 0 \end{cases}$$

Using the equations $s_{1,j} = 0$ from the ideal $\mathcal{J}'$, we obtain $u_{1,1} = 0, u_{1,2} = 0, \ldots, u_{1,n} = 0$. □

We can also predict the existence of other linear equations in $I_{2,1}(F)$.

Lemma 16.3. *For all $(i,j) \in \{1, \ldots, n\}^2$ the coefficient of $y_1 y_i x_j$ in*

$$\mathrm{D}\mathbf{a}(S_0(S + I_n)\mathbf{x}, \mathbf{y}) - \mathrm{D}\mathbf{b}(\mathbf{x}, (U + I_n)S_0^{-1}\mathbf{y})$$

is a non zero[1] *linear equation modulo the equations of the ideal $\mathcal{J}'$ and modulo equation* (16.3)*. Among these equations, there are n which depend only of the variables $\{s_{k,\ell} \mid 1 \leq k, \ell \leq n\}$.*

Proof. We consider the coefficient of the monomial $m = y_1 y_i x_j$ in the expression

$$\Delta = \underbrace{\mathrm{D}\mathbf{a}(S_0(S + I_n)\mathbf{x}, \mathbf{y})}_{\Delta_a} - \underbrace{\mathrm{D}\mathbf{b}(\mathbf{x}, (U + I_n)S_0^{-1}\mathbf{y})}_{\Delta_b}.$$

Since the monomial m is linear in x_j it is clear that the corresponding coefficient in Δ_a is also linear in the variables $s_{i,j}$; moreover this coefficient is non zero. We have now to consider the coefficient of m in Δ_b. Since $\mathrm{D}\mathbf{b}(\mathbf{x}, \mathbf{y})$ is the differential of an homogeneous polynomial of degree 3 we can always write:

$$\mathrm{D}\mathbf{b}(\mathbf{x}, \mathbf{y}) = \sum_{i=1}^{n} \sum_{j=i}^{n} \ell_{i,j}(y_1, \ldots, y_n)\, x_i x_j + \sum_{i=1}^{n} q_i(y_1, \ldots, y_n) x_i \tag{16.4}$$

where $\ell_{i,j}$ (resp. q_i) is a polynomial of degree 1 (resp. 2). Consequently, the coefficient of m in $\mathrm{D}\mathbf{b}$ is also the coefficient of $y_1 y_i$ in $q_j((U + I_n)S_0^{-1}\mathbf{y})$. That is to say, in $q_j(\mathbf{y})$ we now have to replace $\mathbf{y} = (y_1, \ldots, y_n)$ by $((U + I_n)S_0^{-1}\mathbf{y})$. Thus, modulo the equations of the ideal $\mathcal{J}'$ and modulo equation (16.3), we can write

1. more precisely, generically non zero.

the product $((U+I_n)S_0^{-1}\mathbf{y})$ as:

$$
\begin{aligned}
&= \begin{pmatrix} y_1 \\ \vdots \\ \vdots \\ y_n \end{pmatrix} \times \begin{pmatrix} 1 & 0 & 0 & 0 \\ u_{2,1} & \cdots & \cdots & u_{2,n} \\ \vdots & \cdots & \cdots & \vdots \\ u_{n,1} & \cdots & \cdots & u_{n,n} \end{pmatrix} \times \begin{pmatrix} * & * & * & * \\ * & * & * & * \\ * & * & * & * \\ * & * & * & * \end{pmatrix} \\
&= \begin{pmatrix} y_1 \\ \vdots \\ \vdots \\ y_n \end{pmatrix} \times \begin{pmatrix} * & * & * \\ (*u_{2,1}+\cdots+*u_{2,n}) & \cdots & \cdots & (*u_{2,1}+\cdots+*u_{2,n}) \\ \vdots & \cdots & \cdots & \vdots \\ (*u_{n,1}+\cdots+*u_{n,n}) & \cdots & \cdots & (*u_{2,1}+\cdots+*u_{n,n}) \end{pmatrix} \\
&= \begin{pmatrix} *y_1+(*u_{2,1}+\cdots+*u_{2,n})y_2+\cdots+(*u_{n,1}+\cdots+*u_{n,n})y_n \\ *y_1+(*u_{2,1}+\cdots+*u_{2,n})y_2+\cdots+(*u_{n,1}+\cdots+*u_{n,n})y_n \\ \vdots \\ *y_1+(*u_{2,1}+\cdots+*u_{2,n})y_2+\cdots+(*u_{n,1}+\cdots+*u_{n,n})y_n \end{pmatrix}
\end{aligned}
$$

Hence the coefficient of y_1y_i in $q_j((U+I_n)S_0^{-1}\mathbf{y})$ is linear in the variables $u_{k,l}$ when $i \neq 1$ and the coefficient of y_1^2 is a constant. □

To summarize:

Lemma 16.4. *$I_{2,1}(F)$ contains exactly n^2+2n linear equations.*

Proof. In $I_{2,1}(F)$, we have n linear equations from lemma 16.3, n linear equations from the very definition of $\mathcal{J}'$, and n^2 linear equations from lemma 16.3. □

As explained before, we obtain n^2+2n linear equations for $I_{2,1}(F)$. However, we have $2n^2$ variables. So, we have to consider $I_{2,2}(F)$, *i.e.*, the equations generated at degree 2 during the second step. Thanks to lemma 16.4, we can reduce the original system to a quadratic system in $2n^2-(2n+n^2)=(n-1)^2$. variables. Without loss of generality we can assume that we keep only the variable $u_{i,j}$ where $2 \leq i,j \leq n$. Let F' be the system obtained from F after substituting the $2n+n^2$ linear equations of lemma 16.4. All the monomials in $\mathbb{K}[x_1,\ldots,x_n,y_1,\ldots,y_n]$ of $\mathrm{D}\mathbf{a}(S_0(S+I_n)\mathbf{x},\mathbf{y})-\mathrm{D}\mathbf{b}(\mathbf{x},(U+I_n)S_0^{-1}\mathbf{y})$ have the following shape:

$$x_iy_jy_k \text{ or } y_ix_jx_k \text{ with } 1 \leq i,j,k \leq n.$$

Hence the number of such monomials is $2n\frac{n(n+1)}{2} = n^2(n+1) \approx n^3$, which implies that the number of equations in F' is also n^3.

Thanks to this remark, we will now prove that we can linearize F'. Let $T(F')$ be the set of all monomials occurring in F'. We can assume that $T(G')=[t_1<t_2<\cdots<t_N]$. It is important to remark that $t_1=u_{2,2}$ up to $t_{(n-1)^2}=u_{n,n}$ are in fact variables. Now, let M be the matrix representation of G' w.r.t. $T(G')$. Since we know precisely the shape of the equations from the proof of lemma 16.3, it is possible to establish that:

1. most of the equations are very sparse, namely each equation contains about n^2 non-zero terms.
2. all the variables $t_1,\ldots,t_{(n-1)^2}$ occur in *all* the equations

After a Gaussian elimination of the matrix M, we obtain the following shape:

$$\widetilde{M} = \begin{bmatrix} 1_{(n-1)^2} & 0 & 0 & 0 \\ 0 & \times & \cdots & \cdots \\ 0 & \times & \ddots & \vdots \\ 0 & \times & \cdots & \ddots \end{bmatrix}$$

Hence, we obtain after a second step of computation in degree 2 the equations:

$$u_{2,2} = \cdots = u_{n,n} = 0.$$

This means that after 2 steps of computation at degree 2, we obtain $(n-1)^2+2n+n^2=2n^2$ linear equations in $2n^2$ unknowns. This explains why the maximum degree reached during the Gröbner basis computation of $\mathcal{I}'_{\mathbf{a},\mathbf{b}}+\mathcal{J}'$ is bounded by 2, and concludes the proof of theorem 16.1.

16.1.1 An Interesting Failure

We conclude this section with a simple idea that could have lead to an improvement, by efficiently giving the image of S on one point, but which fails in an interesting manner. Let us denote by $Z_{\mathbf{a}}$ (resp. $Z_{\mathbf{b}}$) the set of zeroes of $\mathbf{a}$ (resp. $\mathbf{b}$). We have:

$$S\left(\sum_{\mathbf{x}\in Z_{\mathbf{a}}} \mathbf{x}\right) = \sum_{\mathbf{y}\in Z_{\mathbf{b}}} \mathbf{y}$$

This yields a relation on S, which is enough to use theorem 16.1. $\mathbf{a}$ and $\mathbf{b}$ may be assumed to have about q^{n-1} zeroes. Finding them requires time $\mathcal{O}(q^n)$. The complexity of the attack could thus be improved to $\mathcal{O}\left(n^6 + q^n\right)$. Surprisingly, this trick fails systematically, and this happen to be consequence of the Chevalley-Warning theorem .

Theorem 16.5 (Chevalley-Warning [Che35, War35])**.** *Consider a system of polynomial equations $P_j(x_1, \ldots, x_n) = 0,\ j = 1, \ldots, r$, where the P_j are polynomials with coefficients in a finite field $\mathbb{K}$ and such that the number of variables satisfies $n > d_1 + \cdots + d_r$, where d_j is the total degree of P_j.*

The characteristic of $\mathbb{K}$ divides the number of solutions in $\mathbb{K}$ of the polynomial system (which is always finite because $\mathbb{K}$ is finite).

Corollary 16.6. *The sum of the zeroes of a cubic form on 5 variables or more over $\mathbb{F}_q$ is always zero.*

Proof (found by Mehdi Tibouchi). Let us consider the elements of $Z_{\mathbf{a}}$ having α as their first coordinate, and let us denote by n_α their number. These are in fact the common zeroes of $(\mathbf{a}, x_1 - \alpha)$. By the Chevalley-Warning theorem, if $\mathbf{a}$ has at least 5 variables, then the characteristic of the field divides n_α. Therefore, their sum has zero on the first coordinate. Applying this result for all values of α shows that the sum of zeroes of $\mathbf{a}$ has a null first coordinate. We then just consider all coordinates successively. □

Chapter 17

Linear Equivalence of Inhomogeneous Quadratic Maps

In this chapter we present two algorithms for the IQMLE. This problem was empirically known to be solvable in polynomial time, however these new algorithms are much faster than their predecessors. We give strong arguments to justify that they terminate in polynomial time with high probability.

In this chapter we study the IQMLE problem (equivalence of inhomogeneous quadratic maps). Given two inhomogeneous quadratic maps $\mathbf{a}$ and $\mathbf{b}$, along with the promise that there exist two matrices $S, T \in \mathrm{GL}_n(\mathbb{F}_q)$ such that:

$$T \circ \mathbf{b} = \mathbf{a} \circ S, \tag{17.1}$$

the problem is to find S and T (given just $\mathbf{a}$ and $\mathbf{b}$). The maps $\mathbf{a}$ and $\mathbf{b}$ are inhomogeneous if they contain linear and constant terms.

As we discussed in §13.2, random instances of this problem were empirically found to be solvable in polynomial time by the Gröbner-based technique of Faugère and Perret. After running series of experiments, they conjectured that the running time of their algorithm was at most $\mathcal{O}(n^9)$.

We present two improved algorithms. The first one is in fact a variation of the "to-n-fro" algorithm, that uses the presence of a linear component to circumvent the need to invert the quadratic maps. It provably runs in time $\mathcal{O}(n^3)$ and heuristically succeed with high probability probability as soon as the homogeneous components of degree one of $\mathbf{a}$ and $\mathbf{b}$ are invertible.

The second algorithm we present is an instanciation of the matrix-pencil strategy outlined in chapter 14. It is more robust than our first algorithm, as it may succeed on classes of instances where the other one would fail deterministically. We show that it runs in time $\mathcal{O}(n^6)$ on a constant fraction of the instances, and that it succeeds on another constant fraction $1 - 1/q$ of the instances.

Both algorithm have been implemented, and we report on their practicality at the end of the chapter.

17.1 To-and-Fro Without Exponentially-Expensive Inversions

There are two sources of exponentially high costs in the to-n-fro algorithm of Courtois, Goubin and Patarin, that we presented in §13.1. On one hand, we need to know the image of S on one (and sometimes two) points, and it was originally suggested to exhaustively try all the possibilities (cost: q^n or q^{2n}). On the other hand, the quadratic maps have to be inverted at least n times, and each inversion has an exponential cost, be it done by exhaustive search or through a Gröbner basis computation.

It turns out that in the inhomogeneous case both problems can be circumvented. Ironically, the solution was almost entirely contained in some remarks of Faugère and Perret [FP06], the authors of the only other algorithm for QMLE prior to this work. Indeed, the observation at the heart of the new technique is lemma 12.1, which is directly taken from [FP06]. This lemma yields in particular:

$$T \times \mathbf{b}^{(1)} = \mathbf{a}^{(1)} \times S \tag{17.2}$$

$$T \times \mathbf{b}^{(0)} = \mathbf{a}^{(0)} \tag{17.3}$$

It follows immediately that in the linear inhomogeneous case we have *a priori* knowledge [1] on T, because of equation (17.3). In some cases this knowledge can be "transferred" to S without inverting $\mathbf{b}$, but instead using equation (17.2): assume that $\mathbf{b}(0) = \mathbf{b}^{(1)} \cdot \mathbf{x}$ and $\mathbf{a}(0) = \mathbf{a}^{(1)} \cdot \mathbf{y}$, it follows that $\mathbf{y} = S \cdot \mathbf{x}$. This removes the need to know time image of S on one point prior to running the algorithm.

If $\mathbf{a}^{(1)}$ and $\mathbf{b}^{(1)}$ are invertible and if $\mathbf{a}(0) \neq 0$, then this much less expensive knowledge transfer always succeeds. It can even be used in all the subsequent iterations of the to-n-fro algorithm, thus removing

1. Patarin, Goubin and Courtois actually pointed out that this is cheating.

the need for exponentially costly inversions of the quadratic maps. The complete pseudo-code is shown in Algorithm 17.1.

Complexity. The complexity of this procedure is $\mathcal{O}\left(n^3\right)$: inverting both $\mathbf{a}^{(1)}$ and $\mathbf{b}^{(1)}$ can be done once for all. The matrix-vector products take $\mathcal{O}\left(n^2\right)$ and there are n of them. Lastly, reconstructing S and T takes only $\mathcal{O}\left(n^3\right)$, because in the basis $(x_i)_{i\leq n}$, S is made of the y_i's. Changing the basis amounts to performing one matrix inversion and two matrix-matrix products. For all realistic ranges of parameters, the algorithm either fails or terminates in less than a second, even for parameters that were taking several minutes to the Gröbner-based algorithm.

Algorithm 17.1 A variant of the "To-and-Fro" algorithm without exponential inversions

1: **function** Inhomogeneous-To-n-Fro($\mathbf{a}, \mathbf{b}$)
2: **if** $\mathbf{a}^{(1)}$ is non-invertible **or** $\mathbf{a}^{(0)} = 0$ **then abort**
3: $\mathbf{x}_1 \leftarrow \mathbf{a}(0)$
4: $\mathbf{y}_1 \leftarrow \mathbf{b}(0)$
5: **for** $i = 1$ to n **do** ▷ At this point one has $\mathbf{y}_i = T \cdot \mathbf{x}_i$
6: $\mathbf{y}'_i \leftarrow \left(\mathbf{a}^{(1)}\right)^{-1} \cdot \mathbf{x}_i$
7: $\mathbf{x}'_i \leftarrow \left(\mathbf{b}^{(1)}\right)^{-1} \cdot \mathbf{y}_i$ ▷ And we obtain $\mathbf{y}'_i = S \cdot \mathbf{x}'_i$
8: **if** $\mathbf{x}'_i \in \langle \mathbf{x}'_1, \ldots, \mathbf{x}'_{i-1} \rangle$ **then abort**
9: $\mathbf{y}_{i+1} \leftarrow \mathbf{b}\left(\mathbf{x}'_i\right)$
10: $\mathbf{x}_{i+1} \leftarrow \mathbf{a}\left(\mathbf{y}'_i\right)$
11: **end for**
12: Reconstruct S from the pairs $(\mathbf{x}'_i, \mathbf{y}'_i)$ and T from the pairs $(\mathbf{x}_i, \mathbf{y}_i)$.
13: **return** (S, T)
14: **end function**

Success Probability. It is easy to evaluate the probability that the algorithm fails because its prerequisite are not met on line 2. We have seen in §10.7 that a random $n \times n$ matrix is invertible with probability greater than $\lambda(n)$. This tells us that for a random inhomogeneous instance, the probability that $\mathbf{a}^{(1)}$ is invertible is about 0.288 when $q = 2$ (this quickly grows to one for higher values of q) and the probability that $\mathbf{a}(0) \neq 0$ is $1 - 1/q^n$. Again, $q = 2$ looks like a worst case.

Deterministic Failure in Degenerate Cases. To highlight the danger of relying on heuristic assumptions and unfinished analysis, we point out that it is quite easy to cook up a class of instances *satisfying the two prerequisites* but on which the algorithms fails deterministically, since these conditions are necessary but not sufficient. In particular, it may fail unexpectedly if $\mathbf{a}$ and $\mathbf{b}$ are not "random" enough. Indeed, the non-linearity of $\mathbf{a}$ and $\mathbf{b}$ plays a crucial role in making each new relation on S and T linearly independent from the previous ones, as is it required for the algorithm to work.

For example, this algorithm cannot be used to determine the *automorphism group* of a quadratic map $\mathbf{a}$, namely the set of pairs (S, T) such that $T \circ \mathbf{a} = \mathbf{a} \circ S$. In this special case, the initial "bootstraping" relation $T \cdot \mathbf{a}(0) = \mathbf{a}(0)$ describes a fixpoint of T, and we have $\mathbf{x}_1 = \mathbf{y}_1$. It is not difficult to check that we will always have $\mathbf{x}_i = \mathbf{y}_i$ and $\mathbf{x}'_i = \mathbf{y}'_i$. The algorithm in fact only discovers the images of S and T over their respective invariant subspaces. Thus, it cannot fully discover S and T, unless both were... the identity matrix!

This particular class of instances is not a contrived example. We will see in chapter 19 that problems of this kind arise naturally when studying a variant of HFE.

17.2 A Pencil-Based Approach

We now present an instanciation of the general matrix pencil strategy outlined in chapter 14 in the case of IQMLE. The problem is to find a matrix pencil equivalence problem that is satisfied by the solutions of the IQMLE equation (17.1).

17.2.1 Obtaining a Related Pencil Equivalence Problem

Equation (17.2) already gives us a matrix equation, between S and T, but we would need another one to have a full pencil equivalence problem. It is possible to obtain this second matrix equation by *differentiating* both sides of (17.1). However, in order to carry out this operation, the image of S must be known on one point. We have seen in the previous section that with some luck, this much knowledge about S can in fact

be deduced from equations (17.2) and (17.3): if there exist $\mathbf{y}$ be such that $\mathbf{a}^{(0)} = \mathbf{a}^{(1)} \cdot \mathbf{y}$, then there also exist $\mathbf{x}$ such that $\mathbf{b}^{(0)} = \mathbf{b}^{(1)} \cdot \mathbf{x}$ and we find that $\mathbf{y} = S \cdot \mathbf{x}$.

The two vectors $\mathbf{x}$ and $\mathbf{y}$ can only be identified uniquely if $\mathbf{a}^{(1)}$ (and therefore $\mathbf{b}^{(1)}$) is invertible. Otherwise, if $\dim \ker \mathbf{a}^{(1)} = r > 1$, a possible solution is to choose arbitrarily a particular solution for $\mathbf{x}$, and then try the q^r possible values of $\mathbf{y}$ until a match is found — the possible $\mathbf{x}$ and $\mathbf{y}$ can be computed by straightforward linear algebra.

Given a "right pair" $\mathbf{y} = S \cdot \mathbf{x}$, we obtain as announced above a new matrix equation by *differentiating* equation (17.1). It follows from the definition of the differential that if $\mathbf{y} = S \cdot \mathbf{x}$:

$$T \times \mathrm{D}_{\mathbf{x}}\mathbf{b} = \mathrm{D}_{\mathbf{y}}\mathbf{a} \times S.$$

Putting things together, we have found that S and T are the solutions of a matrix pencil equivalence problem:

$$T \cdot \left(\underbrace{\lambda \cdot \mathbf{b}^{(1)} + \mu \cdot \mathrm{D}_{\mathbf{x}}\mathbf{b}}_{\mathbf{P}} \right) = \left(\underbrace{\lambda \cdot \mathbf{a}^{(1)} + \mu \cdot \mathrm{D}_{\mathbf{y}}\mathbf{a}}_{\mathbf{Q}} \right) \cdot S \tag{17.4}$$

17.2.2 Analysis of the Corresponding Algorithm.

Algorithm 17.2 Pencil-based algorithm for IQMLE.

1: **function** INHOMOGENEOUS-PENCIL-QMLE($\mathbf{a}, \mathbf{b}$)
2: **let** $\mathbf{x}$ be particular solution of $\mathbf{b}^{(1)} \cdot \mathbf{x} = \mathbf{b}^{(0)}$. **If** no solution exist **then abort.**
3: **let** $\mathbf{y}_0$ be a particular solution of $\mathbf{a}^{(1)} \cdot \mathbf{y} = \mathbf{a}^{(0)}$
4: **let** $Sols = \emptyset$
5: **for all** $\mathbf{y} \in \mathbf{y}_0 + \ker \mathbf{a}^{(1)}$ **do**
6: **let** M_ϕ denote the $2n^2 \times 2n^2$ matrix representing:

$$\begin{array}{rccl} \phi: & \mathcal{M}_n(\mathbb{F}_q) \times \mathcal{M}_n(\mathbb{F}_q) & \to & \mathcal{M}_n(\mathbb{F}_q) \times \mathcal{M}_n(\mathbb{F}_q) \\ & (X, Y) & \to & \begin{pmatrix} Y \times \mathbf{b}^{(1)} - \mathbf{a}^{(1)} \times X \\ Y \times \mathrm{D}\mathbf{x}\mathbf{b} - \mathrm{D}_{\mathbf{y}}\mathbf{a} \times X \end{pmatrix} \end{array}$$

7: **let** $\mathbb{V} = \ker M_\phi$
8: **let** $\ell = \dim \mathbb{V}$ **and** $\left(S^{[1]}, T^{[1]}\right), \ldots, \left(S^{[\ell]}, T^{[\ell]}\right)$ be a basis of $\mathbb{V}$
9: **if** $\ell \geq n$ **then**
10: **consider** the two rings $R_\perp \leftarrow \mathbb{F}_q[x_1, \ldots, x_\ell]$ **and** $R^\top \leftarrow \mathbb{F}_q[x_1, \ldots, x_\ell][y_1, \ldots, y_n]$
11: **let** $\widetilde{S} = \sum_{i=1}^{\ell} x_i \cdot S^{[i]}$ **and** $\widetilde{T} = \sum_{i=1}^{\ell} x_i \cdot T^{[i]}$
12: **let** $\mathbf{y} = (y_1, \ldots, y_n)$
13: **let** $\mathbf{z}(\mathbf{y}) = \widetilde{T} \cdot \mathbf{b}(\mathbf{y}) - \mathbf{a}\left(\widetilde{S} \cdot \mathbf{y}\right)$ ▷ $\mathbf{z}$ belongs to $\left(R^\top\right)^n$
14: **let** $\mathcal{S}_{\text{quad}}$ be the set of coefficients of $\mathbf{z}_1, \ldots, \mathbf{z}_n$ ▷ $\mathcal{S}_{\text{quad}} \subseteq R_\perp$
15: **let** $\mathcal{I}$ be the ideal of $R_\perp$ spanned by $\mathcal{S}_{\text{quad}}$
16: **compute** $U = \mathbf{V}(\mathcal{I}) \cap (\mathbb{F}_q)^\ell$ by first computing a Gröbner basis of $\mathcal{I}$.
17: $Sols \leftarrow Sols \cup \left\{ \left(\sum_{i=1}^{\ell} u_i \cdot S^{[i]}, \sum_{i=1}^{\ell} u_i \cdot T^{[i]} \right) \middle| (u_1, \ldots, u_\ell) \in U \right\}$
18: **end if**
19: **end for**
20: **return** Sols
21: **end function**

Unfolding the general pencil technique outlined in chapter 14 and adapting the details yields Algorithm 17.2. We will say that an instance of the problem over which the algorithm does not fail on line 2 is *nice*. On a nice instance, then in at least one iteration of the loop we will actually have $\mathbf{y} = S \cdot \mathbf{x}$, and thus the matrix pencil equivalence problem will actually have solutions that include the actual solutions of the IQMLE problem. In this case, these solutions will be found and the algorithm will eventually return them. Note that in this iteration, the dimension of $\mathbb{V}$ is necessarily greater than or equal to n (as per theorem 14.3). This allows the "if" on line 9 to discard bad choices of $\mathbf{y}$ very efficiently. It remains to compute the fraction of nice instances, the expected number of iterations of the loop, the expected dimension of $\mathbb{V}$, and the expected running time of the Gröbner basis computation.

q	2	3	4	5	6	7	8	16	256	65536
P_{nice}	0.610	0.703	0.765	0.808	0.838	0.860	0.877	0.937	0.996	0.999985

Table 17.1: Probability that the pencil-based algorithm succeeds on random instances.

Failure Probability. If an instance is random, then $\mathbf{b}^{(1)}$ is a random linear map, while $\mathbf{b}^{(0)}$ is a random vector. The probability that the latter is non-zero is simply $1-q^{-n}$. The probability that a specified non-zero vector belongs to the image of a random linear map can be derived from equation (10.6), and is shown in Table 17.1. The probability that a random instance is nice is essentially $P_{\text{nice}} = 1 - 1/q + 1/q^3 + \mathcal{O}\left(1/q^6\right)$. When $q = 2$, where this probability is minimal, the algorithm is capable to deal with only 61% of all the instances.

Expected Number of Iterations of the Loop. The number of iterations of the loop is the expected cardinality of $\ker \mathbf{b}^{(1)}$, which is a uniformly random matrix. Therefore, lemma 10.20 tells us that this expectation is $2 + \mathcal{O}\left(1/q^n\right)$. In addition, we can easily obtain an interesting deviation bound using lemma 10.19 and equation (10.4), by observing that the probability that there will be at most q iterations is:

$$p' = \frac{D(n,0,0) + D(n,1,0)}{q^{n^2}} \leq \lambda(n) \cdot \left(1 + \frac{q}{(q-1)^2}\right) = 1 - \frac{1}{q^4} + \mathcal{O}\left(\frac{1}{q^5}\right)$$

As soon as we try to figure out anything about $\dim \mathbb{V}$, the usual complications pop up.

17.2.3 Analysis In Odd Characteristic

Regularity of the Pencil. Exactly as in chapter 15, we observe that the pencil is regular as soon as one of the two matrices it is made from is invertible. The linear component $\mathbf{b}^{(1)}$ are easy to deal with: since it is uniformly random, it is invertible with probability $\lambda(n)$. The other matrix, $\mathrm{D}_{\mathbf{x}}\mathbf{b}$ is *a priori* a bit tougher to handle, since we do not know what it looks like. Fortunately, it is in fact plainly random, as the next lemma shows.

Lemma 17.1. *Let $\mathbb{F}_q$ be a field of odd characteristic. Given a matrix $M \in \mathcal{M}_n(\mathbb{F}_q)$ and a non-zero vector $\mathbf{y} \in (\mathbb{F}_q)^n$, then the number of quadratic maps $\mathbf{f}$ such that $\mathrm{D}_{\mathbf{y}}\mathbf{f} = M$ is independent from L and $\mathbf{y}$. Therefore, the differential of a random quadratic map at a given point is a random matrix.*

Proof. We first note that $\mathrm{D}_{\mathbf{y}}\mathbf{f}$ depends only on $\mathbf{y}$ and on the quadratic homogeneous component of $\mathbf{f}$. The linear and constant terms of $\mathbf{f}$ play no role here, thus we will assume $\mathbf{f}$ to be homogeneous. Let us therefore represent $\mathbf{f}$ as:

$$\mathbf{f} = \sum_{i=1}^{n}\sum_{j=i}^{n} f_{ij} \cdot x_i x_j, \qquad f_{ij} \in (\mathbb{F}_q)^n.$$

Let us also be given a matrix M represented as:

$$M \cdot \mathbf{x} = \sum_{i=1}^{n} M_{\bullet i} \cdot \mathbf{x}_i$$

We want to find all the possible assignments of f_{ij} such that $M = \mathrm{D}_{\mathbf{y}}\mathbf{f}$. It follows from the bilinearity of the differential that this is equivalent to:

$$\sum_{i=1}^{n} M_{\bullet i} \cdot \mathbf{x}_i = \sum_{i=1}^{n} \left(\sum_{j=1}^{i-1} f_{ji} \cdot \mathbf{y}_j + 2 f_{ii} \cdot \mathbf{y}_i + \sum_{j=i+1}^{n} f_{ij} \cdot \mathbf{y}_j \right) \cdot \mathbf{x}_i$$

Since this equation holds for all values of $\mathbf{x}$, we deduce that:

$$M_{\bullet i} = \sum_{j=1}^{i-1} f_{ji} \cdot \mathbf{y}_j + 2 f_{ii} \cdot \mathbf{y}_i + \sum_{j=i+1}^{n} f_{ij} \cdot \mathbf{y}_j, \qquad i = 1, \ldots, n$$

This can be represented as a nice "generalized" linear system, where the matrix of the "polar form" of $\mathbf{f}$ appears:

$$\begin{pmatrix} M_{\bullet 1} \\ M_{\bullet 2} \\ \vdots \\ M_{\bullet n} \end{pmatrix} = \begin{pmatrix} 2f_{11} & f_{12} & \cdots & f_{1n} \\ f_{12} & 2f_{22} & \cdots & f_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ f_{1n} & f_{2n} & \cdots & 2f_{nn} \end{pmatrix} \cdot \begin{pmatrix} \mathbf{y}_1 \\ \mathbf{y}_2 \\ \vdots \\ \mathbf{y}_n \end{pmatrix}$$

Anyway, since $\mathbf{y} \neq 0$, we can assume w.l.o.g. (up to renumbering) that $\mathbf{y}_n \neq 0$. We then find two necessary and sufficient conditions for $\mathrm{D}_{\mathbf{y}}\mathbf{f} = M$:

$$\begin{aligned} f_{in} &= \frac{1}{\mathbf{y}_n}\left(M_{\bullet i} - \sum_{j=1}^{i-1} f_{ji}\cdot\mathbf{y}_j - 2f_{ii}\cdot\mathbf{y}_i - \sum_{j=i+1}^{n-1} f_{ij}\cdot\mathbf{y}_j\right), \qquad 1 \leq i < n \\ f_{nn} &= \frac{1}{2\mathbf{y}_n}\left(M_{\bullet n} - \sum_{j=1}^{n-1} f_{jn}\cdot\mathbf{y}_j\right), \end{aligned}$$

To summarize, the f_{ij} can be chosen arbitrarily for $j < n$, and these choices determine the the f_{in} in a unique way. This shows that the number of possible quadratic maps $\mathbf{f}$ is always the same. In turn, this ensures that $\mathrm{D}_{\mathbf{y}}\mathbf{f}$ is uniformly distributed as long as $\mathbf{f}$ itself is uniformly distributed. □

So, coming back to the regularity of the pencils, lemma 17.1 tells us that $\mathrm{D}_{\mathbf{x}}\mathbf{b}$ is in fact a random matrix. It must be noted that this matrix is statistically independent from $\mathbf{b}^{(1)}$ (because it depends only on the quadratic terms of $\mathbf{b}$). The pencil is therefore regular with probability:

$$\mathbb{P}\left[\mathbf{P} \text{ is regular}\right] \geq 1 - (1-\lambda(n))^2 = 1 - \frac{1}{q^2} - \frac{2}{q^3} + \mathcal{O}\left(\frac{1}{q^4}\right)$$

Distribution of $\dim \mathbb{V}$**.** Since we argued that the pencil is uniformly random, we will be able to directly make use of the results presented in chapter 14, notably of lemma 14.4, under the assumption that the pencil is regular. This yields:

$$\mathbb{P}\left[\dim \mathbb{V} = n\right] \geq \mathbb{P}\left[\text{a random } n \times n \text{ matrix over } \mathbb{F}_q \text{ is cyclic}\right] \cdot \mathbb{P}\left[\mathbf{P} \text{ is regular}\right]$$

Thus when n goes to $+\infty$, we find:

$$\mathbb{P}\left[\dim \mathbb{V} = n\right] \geq 1 - \frac{1}{q^2} - \frac{3}{q^3} - \frac{2}{q^4} - \frac{1}{q^5} + \mathcal{O}\left(\frac{1}{q^6}\right)$$

17.2.4 Analysis in Characteristic Two

Regularity of the Pencil. The situation is a bit different in characteristic two, since $\mathrm{D}_{\mathbf{x}}\mathbf{b}$ always vanishes on $\mathbf{x}$ (it is then *not* uniformly random). The pencil is always regular if $\mathbf{b}^{(1)}$, which is still uniformly distributed, is non-singular:

$$\mathbb{P}\left[\mathbf{P} \text{ is regular}\right] \geq \lambda_\infty(q) = 1 - \frac{1}{q} + \mathcal{O}\left(\frac{1}{q^2}\right)$$

This lower-bound is looser than the one we obtained using similar a similar reasoning in the odd case. To improve on this result, we must also take the matrix $\mathrm{D}_{\mathbf{x}}\mathbf{b}$ into account. For this purpose, we first need an analogous of lemma 17.1 in the even case.

Lemma 17.2 ([Dub07], theorem 2)**.** *Let $\mathbb{F}_q$ be a field of characteristic two. Given a matrix $M \in \mathcal{M}_n(\mathbb{F}_q)$ and a vector $\mathbf{y}$ such that $M\cdot\mathbf{x} = 0$, then the number of quadratic maps $\mathbf{f}$, such that $\mathrm{D}_{\mathbf{y}}\mathbf{f} = M$ is independent from M and $\mathbf{y}$. Therefore, the differential of a random quadratic map on a given point is a random matrix vanishing at that point.*

Proof. The proof is quite similar to that of lemma 17.1 (in fact, the proof of lemma 17.1 is heavily inspired by this one, which was originally given by Dubois). The reasoning is essentially the same, except that the "diagonal" terms are gone:

$$M_{\bullet i} = \sum_{j=1}^{i-1} f_{ji}\cdot\mathbf{y}_j + \sum_{j=i+1}^{n} f_{ij}\cdot\mathbf{y}_j, \qquad i = 1,\ldots,n \tag{17.5}$$

And again, as a "generalized" linear system:

$$\begin{pmatrix} M_{\bullet 1} \\ M_{\bullet 2} \\ \vdots \\ M_{\bullet n} \end{pmatrix} = \begin{pmatrix} 0 & f_{12} & \cdots & f_{1n} \\ f_{12} & 0 & \cdots & f_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ f_{1n} & f_{2n} & \cdots & 0 \end{pmatrix} \cdot \begin{pmatrix} \mathbf{y}_1 \\ \mathbf{y}_2 \\ \vdots \\ \mathbf{y}_n \end{pmatrix}$$

Since $\mathbf{y} \neq 0$, we again assume w.l.o.g. that $\mathbf{y}_n \neq 0$. This time, we find that a necessary and sufficient condition for $\mathrm{D}_{\mathbf{y}}\mathbf{f} = M$ is:

$$f_{in} = \frac{1}{\mathbf{y}_n}\left(M_{\bullet i} - \sum_{j=1}^{i-1} f_{ji}\cdot\mathbf{y}_j - \sum_{j=i+1}^{n-1} f_{ij}\cdot\mathbf{y}_j\right), \qquad 1 \leq i < n$$

These values are chosen to satisfy equation (17.5) for $1 \leq i < n$. We readily verify that (17.5) holds even when $i = n$, because M vanishes on $\mathbf{y}$:

$$\begin{aligned}\sum_{i=1}^{n-1} f_{in}\cdot\mathbf{y}_i &= \frac{1}{\mathbf{y}_n}\left(\sum_{i=1}^{n-1}\mathbf{y}_i\cdot M_{\bullet i}\right) - \frac{1}{\mathbf{y}_n}\left(\sum_{i=1}^{n-1}\sum_{j=1}^{i-1} f_{ji}\cdot\mathbf{y}_i\mathbf{y}_j\right) - \frac{1}{\mathbf{y}_n}\left(\sum_{i=1}^{n-1}\sum_{j=i+1}^{n-1} f_{ij}\cdot\mathbf{y}_i\mathbf{y}_j\right)\\ &= M_{\bullet n}\end{aligned}$$

To summarize, the f_{ij} can again be chosen arbitrarily for $j < n$, while the f_{in} are determined by the choice of the others. □

Back to our pencil regularity problem, lemma 17.2 tells us that $\mathrm{D}_{\mathbf{x}}\mathbf{b}$ is uniformly distributed amongst the matrices that vanish at $\mathbf{x}$. To obtain a tighter lower-bound on the probability that the pencils are regular, we observe that if $\mathbf{b}^{(1)} + \mathrm{D}_{\mathbf{x}}\mathbf{b}$ is non-singular, then the pencil is regular. We will thus compute the probability that $\mathbf{b}^{(1)} + \mathrm{D}_{\mathbf{x}}\mathbf{b}$ is non-singular when $\mathbf{b}^{(1)}$ is singular. This probability is in fact given by lemma 10.27:

$$\mathbb{P}\left[\det\left(\mathbf{b}^{(1)} + \mathrm{D}_{\mathbf{x}}\mathbf{b}\right) \neq 0 \;\middle|\; \det\mathbf{b}^{(1)} = 0\right] = 1 - \left(1 - \lambda(n-1)\right)^2 + \mathcal{O}\left(\frac{1}{q^n}\right)$$

All-in-all we find:

$$\begin{aligned}\mathbb{P}\left[\mathbf{P} \text{ is regular}\right] &\geq \mathbb{P}\left[\det\mathbf{b}^{(1)} \neq 0\right] + \mathbb{P}\left[\det\mathbf{b}^{(1)} = 0\right]\cdot\mathbb{P}\left[\det\left(\mathbf{b}^{(1)} + \mathrm{D}_{\mathbf{x}}\mathbf{b}\right) \neq 0 \;\middle|\; \det\mathbf{b}^{(1)} = 0\right]\\ &\geq \lambda(n) + (1 - \lambda(n))\cdot\left(1 - \left(1 - \lambda(n-1)\right)^2\right) + \mathcal{O}\left(\frac{1}{q^n}\right)\end{aligned}$$

And when n goes to infinity, we find:

$$\mathbb{P}\left[\mathbf{P} \text{ is regular}\right] \geq 1 - \frac{1}{q^3} - \frac{3}{q^4} - \frac{3}{q^5} + \mathcal{O}\left(\frac{1}{q^6}\right)$$

This improved bound seems better than the one we derived in the odd case. This is not very surprising, as we did a somewhat deeper analysis, taking into account the interaction between the two matrices of the pencil.

Dimension of $\mathbb{V}$. Under the assumption that the pencil is regular, we may as usual use theorem 14.3. However, we cannot use lemma 14.4 as in the odd case, because $\mathcal{C} = \mathrm{D}_{\mathbf{x}}\mathbf{b} \times \left(\mathbf{b}^{(1)}\right)^{-1}$ is always singular (and thus not random). It is in fact still possible to use lemma 14.4, but we must first circumvent the singularity of $\mathcal{C}$.

Theorem 17.3. *Let $p = \mathbb{P}\left[\dim\mathbb{V} \leq n+2 \;\middle|\; \mathbf{P} \text{ is regular}\right]$. Then p is greater than the probability that a random $(n-1)\times(n-1)$ matrix over $\mathbb{F}_q$ is cyclic. Thus:*

$$\lim_{n\to\infty} p \geq 1 - \frac{1}{q^3} + \mathcal{O}\left(\frac{1}{q^4}\right)$$

Proof. When $\mathbf{P}$ is regular, theorem 14.3 tells us that the dimension of $\mathbb{V}$ is the dimension of the commutant of $\mathcal{C} = \mathrm{D}_{\mathbf{x}}\mathbf{b} \times \left(\mathbf{b}^{(1)}\right)^{-1}$, and this dimension in turn only depends on the similarity invariants of $\mathcal{C}$.

Let $P \in \mathrm{GL}_n(\mathbb{F}_q)$ be such that $P \cdot e_1 = \mathbf{x}$, and define $\mathcal{C}' = P^{-1} \times \mathcal{C} \times P$. By definition, $\mathcal{C}$ and $\mathcal{C}'$ have the same similarity invariants, thus $\dim\mathbb{V}$ is a function of $\mathcal{C}'$. Next, we claim that $\mathcal{C}'$ is uniformly distributed amongst the matrices that have a first null column: lemma 17.2 tells us that $\mathrm{D}_{\mathbf{x}}\mathbf{b}$ is uniformly distributed amongst the matrices vanishing at $\mathbf{x}$. Because $\left(\mathbf{b}^{(1)}\right)^{-1}$ is a bijection, then the product $\mathcal{C}$ is uniformly distributed amongst the matrices that vanish at $\mathbf{x}$. The conjugation by P is a permutation of the set of matrices that transforms matrices vanishing at $\mathbf{x}$ into matrices with a null first column.

Let U be the $(n-1)\times(n-1)$ submatrix of $\mathcal{C}'$ obtained by removing the first row and the first column. It is clear from the previous point that U is a uniformly random matrix (to which lemma 14.4 may be applied). Also, it is easy to see that $\mathrm{CharPoly}(\mathcal{C}) = X\cdot\mathrm{CharPoly}(U)$, and that $\mathrm{MinPoly}(U)$ divides $\mathrm{MinPoly}(\mathcal{C})$.

Lemma 14.4 tells us that the probability that the minimal polynomial of U has degree $n-1$, and therefore that the the degree of $\mathrm{MinPoly}(\mathcal{C})$ is at least $n-1$, is $c(n,q)$. Assuming that $\deg \mathrm{MinPoly}(\mathcal{C}) \geq n-1$, there are two possible cases. Either $\mathcal{C}$ is cyclic, and it has one similarity invariant, $\mathrm{MinPoly}(\mathcal{C})$, which is of degree n. In this case, $\dim \ker \mathcal{S} = n$. Or $\mathcal{C}$ has two similarity invariants: X, and $\mathrm{MinPoly}(\mathcal{C})$, which is of degree $n-1$, and $\dim \ker \mathcal{S} = n+2$. □

Assembling all the pieces, we find:

$$\mathbb{P}\left[\dim \mathbb{V} \leq n+2\right] \geq \mathbb{P}\left[\text{a random } (n-1)\times(n-1) \text{ matrix over } \mathbb{F}_q \text{ is cyclic}\right] \cdot \mathbb{P}\left[\mathbf{P} \text{ is regular}\right]$$

Thus when n goes to $+\infty$, we find:

$$\mathbb{P}\left[\dim \mathbb{V} \leq n+2\right] \geq 1 - \frac{2}{q^3} - \frac{4}{q^4} - \frac{5}{q^5} + \mathcal{O}\left(\frac{1}{q^6}\right)$$

17.2.4.1 Summary

We have seen that regardless of the parity of q, the dimension of $\mathbb{V}$ is of order n (resp. $n+2$) with a probability of order $1-1/q^2$. The set of equations $\mathcal{S}_{\text{quad}}$ is therefore made of $\mathcal{O}\left(n^3\right)$ equations in $\mathcal{O}\left(n\right)$ variables with high probability, a situation where we expect to be able to solve the equations in polynomial time by linearization *if they are linearly independent*. The pathetic failure of this hypothesis in chapter 15 has made us cautious. This time the problem is less structured, and we do not expect to observe the same funny behavior. Experiments confirm this intuition, and we have observed that we are always able to solve $\mathcal{S}_{\text{quad}}$ by linearization in time $\mathcal{O}\left(n^6\right)$.

The main obstacle in a possible extension of the analysis is that it is difficult to control the dimension of $\mathbb{V}$ without assuming that the linear part of the instance is invertible. However, one may expect that if it is of rank, say, $n-1$, then the situation is not very different in practice.

17.3 An IQMLE Library in MAGMA

We have implemented the two algorithms described in this chapter using the MAGMA computer algebra system, as well as the Gröbner-basis algorithm of Faugère and Perret (described in §13.2). Given an instance of the problem, the most natural strategy seems to first try the fastest algorithm (the inversion-free to-n-fro), and in case of failure to fall back on the second fastest algorithm (the pencil-based strategy). In case of a second failure, we would then use the Gröbner-basis algorithm, that never fails. We again rely on MAGMA's efficient implementation of Gröbner basis algorithms. Our implementation is in the public-domain, and is available at:

`http://www.di.ens.fr/~bouillaguet/implem/iqmle.magma`

Figure 17.1 shows the running time of our implementation of Pencil-IQMLE. The actual running time of the function seems to be cubic.

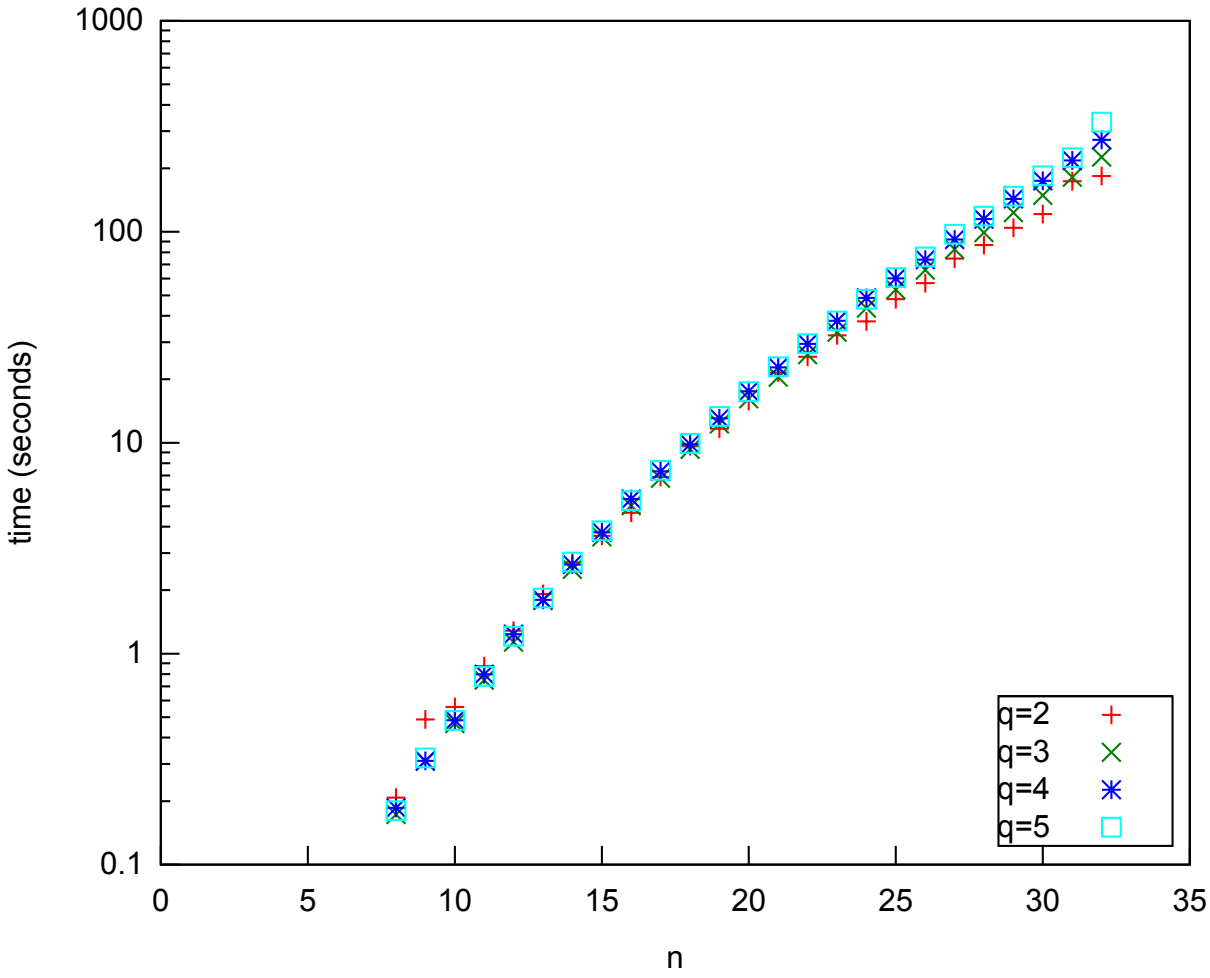


Figure 17.1: Running time of Pencil-IQMLE (when the function does not abort).

Chapter 18

Linear Equivalence of Homogeneous Quadratic Maps

In this chapter we present several new algorithms for the (homogeneous) QMLE problem. All known techniques are exponential in this case. Our algorithms rely on the birthday paradox to improve on naive exhaustive search. When possible, we establish a rigorous upper-bound on the time complexity of these algorithms. This analysis is the result of a joint work with Amandine Véber.

We conclude our study of polynomial linear equivalence problem by presenting new algorithms for the hard (*i.e.*, homogeneous) case of QMLE. The method designed in the previous chapter for IQMLE cannot be directly applied in the homogeneous case for obvious reasons. However, the fact that IQMLE is *easy* is of utmost importance, because the most promising strategy to tackle QMLE is apparently to reduce it to one or several instances of IQMLE.

Let us recall that given two homogeneous quadratic maps $\mathbf{a}$ and $\mathbf{b}$, the problem is to find two bijective endomorphisms of $(\mathbb{F}_q)^n$, S and T such that:

$$T \circ \mathbf{b} = \mathbf{a} \circ S \tag{18.1}$$

In this chapter, we propose two *Monte-Carlo* algorithms for QMLE. The first one is simpler and runs in time equivalent to $\mathrm{Poly}(n) \cdot q^{2n/3}$ arithmetic operations in $\mathbb{F}_q$. Our second algorithm is more sophisticated and runs in time equivalent to $\mathrm{Poly}(n) \cdot q^{n/2}$ field operations, yet so far it is only applicable when $q = 2$. Recall that the to-n-fro algorithm (Algorithm 13.1) has a complexity of $\mathcal{O}\left(n \cdot q^{2n}\right)$, and that in the homogeneous setting standard estimates suggest that the Gröbner-based algorithm of §13.2 has a complexity of order $\mathcal{O}\left(2^{18n}\right)$

Interestingly, while all our other algorithms are deterministic, these ones are probabilistic, and an essentially algebraic problem is addressed by essentially statistical methods. These algorithms sometimes fail to detect that two quadratic maps are equivalent (there are false negatives), but never wrongly claim that they are equivalent. In addition, when they do not explicitly fail, the algorithms explicitly determine S and T, which can be used to certify the result. For the sake of simplicity, in this whole chapter we will assume that q is *even*, and we will discuss how the techniques we present can be adapted to fields of odd characteristic.

Dehomogenization. Our key idea when dealing with QMLE is to *dehomogenize* the problem: if the image of S is known on a given vector $\mathbf{x}$, then we can explicitly determine the two related *inhomogeneous* quadratic maps $\mathbf{a}'(\mathbf{z}) = \mathbf{a}(\mathbf{z} + S \cdot \mathbf{x})$ and $\mathbf{b}'(\mathbf{z}) = \mathbf{b}(\mathbf{z} + \mathbf{x})$. The solutions of equation (18.1) are then precisely the solutions of the inhomogeneous instance: $T \circ \mathbf{b}' = \mathbf{a}' \circ S$. We expect to be able to solve the IQMLE instance in practical time, so that the whole problem boils down to acquiring the image of S on (at least) one point. This already leads to a semi-trivial algorithm: fix a non-zero vector $\mathbf{x}$, enumerate all the $q^n - 1$ possible non-zero vectors $\mathbf{y}$, dehomogenize the problem assuming that $\mathbf{y} = S \cdot \mathbf{x}$, and solve the resulting IQMLE instance using the techniques of chapter 17. We discuss this semi-trivial approach a bit further in §18.1.

Main Ideas. Our global strategy to improve on the (semi-)trivial approach is to first discover the image of S on a single point faster than exhaustive search, then to dehomogenize the problem and turn it into an instance of IQMLE. The intuitions behind our techniques to recover the image of S are nicely explained in a graph-theoretic framework. Consider the pair of undirected graphs $G_{\mathbf{a}} = (V, E_{\mathbf{a}})$ and $G_{\mathbf{b}} = (V, E_{\mathbf{b}})$, where the common set of vertices is $V = \mathbb{F}_q - \{0\}$, and the edges are such that $(x, y) \in E_{\mathbf{a}}$ if and only if $x \in \ker \mathrm{D}_y \mathbf{a}$ (the same goes for $E_{\mathbf{b}}$). Figure 18.1 shows a "medium-sized" connected component of such a graph. It is striking that the graph is formed by small triangles, but is otherwise tree-like. The presence of the small triangles follows from the vector-space structure of the kernel: if there are edges $x \leftrightarrow y$ and $x \leftrightarrow z$, then there is necessarily an edge $x \leftrightarrow (y + z)$. In particular, a node is always connected to itself, so that

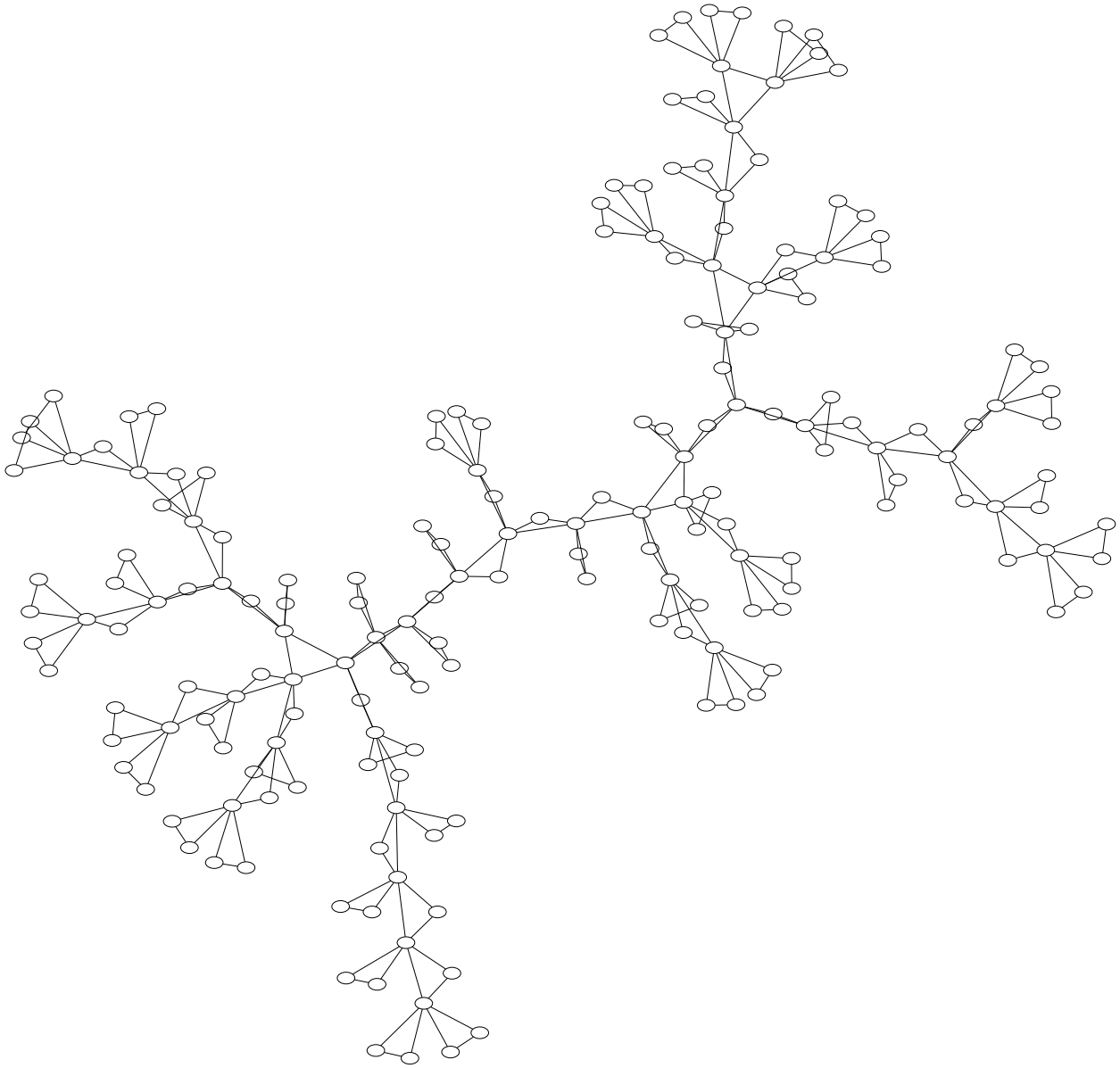


Figure 18.1: A "large" connected component of $G_{\mathbf{a}}$, with $q = 2$ and $n = 64$. Self-edges are not shown.

as soon as there is an edge $x \leftrightarrow y$, there is also an edge $x \leftrightarrow (x + y)$, and an edge $(x + y) \leftrightarrow y$, forming a triangle. The interest of these graphs is that they nicely capture the action of S on $\mathbf{a}$ and $\mathbf{b}$.

Lemma 18.1. *The secret matrix S is a permutation of V that, when seen as a graph isomorphism, transforms $G_{\mathbf{b}}$ into $G_{\mathbf{a}}$.*

This result is an easy consequence of the fact that S transform $\ker \mathrm{D}_{\mathbf{x}}\mathbf{b}$ into $\ker \mathrm{D}_{(S\cdot\mathbf{x})}\mathbf{a}$, which is itself an easy consequence of the "differential" equation:

$$T \times \mathrm{D}_{\mathbf{x}}\mathbf{b} = \mathrm{D}_{(S\cdot\mathbf{x})}\mathbf{a} \times S \tag{18.2}$$

This equation can be obtained by "differentiating" equation (18.1) at $\mathbf{x}$, as was done in the previous chapter.

The usual algorithms for Graph Isomorphism cannot really be used to recover the isomorphism between $G_{\mathbf{a}}$ and $G_{\mathbf{b}}$, since they are at least linear in the size of the graph. In addition, recovering the full isomorphism (whose size is exponential) is not necessary, as knowing how the isomorphism transforms a single vertex is sufficient to completely retrieve it by dehomogenization and usage of an IQMLE solver.

Recall that the *distance* between two nodes $x, y \in V$ is the minimal number of edges that must be crossed to move from x to y. If x and y do not belong to the same connected component, then the distance between them is $+\infty$. The *neighborhood* of radius k of a given node x is the set of all nodes y such that the distance between x and y is less than or equal to k. The *Weisfeiler-Lehman* method for Graph Isomorphism exploit

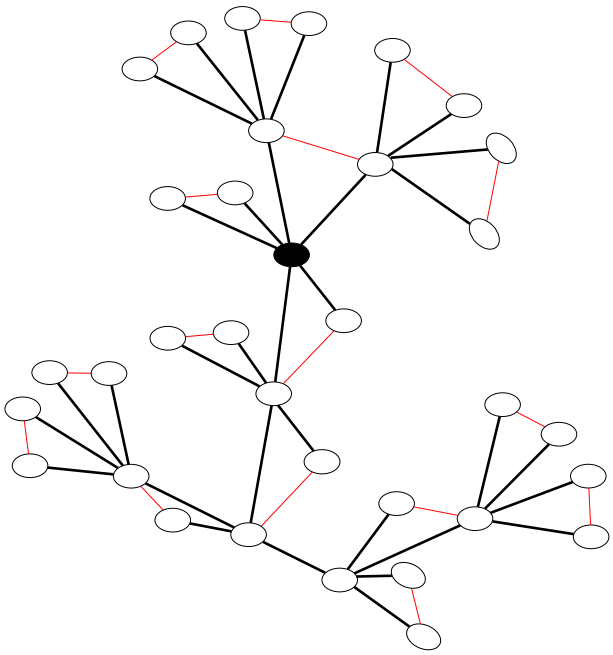


Figure 18.2: A breadth-first exploration of the top part of the graph shown in Figure 18.1. The exploration starts from the black node. Thin red edges are "backwards" and will not appear in the tree. Thick edges form the BFS tree.

the fact that the size of neighborhoods are preserved by the isomorphism. This can be used to restrict the number of possible images in $G_{\mathbf{a}}$ of a given node in $G_{\mathbf{b}}$: their neighborhoods of all possible radii must have the same size. A quite direct and simple application of this idea yields a simple algorithm faster than exhaustive search.

Theorem 18.2. *There is a Monte-Carlo* QMLE *algorithm that makes* $\mathcal{O}\left(q^{2n/3}\right)$ *calls to an external* IQMLE *solver, after a preprocessing of complexity* $\mathcal{O}\left(q^{2n/3}\right)$.

This algorithm is the object of §18.3. It works by isolating nodes having degree $q^{\sqrt{n/3}}$ in $G_{\mathbf{a}}$ and $G_{\mathbf{b}}$. It can be estimated that there are $\mathcal{O}\left(q^{2n/3}\right)$ such nodes, and therefore after building two sets U and V of $\mathcal{O}\left(q^{n/3}\right)$ nodes with $q^{\sqrt{n/3}}$ neighbors in $G_{\mathbf{a}}$ and $G_{\mathbf{b}}$ respectively, we are guaranteed by the birthday paradox to find a "right pair" in $U \times V$, *i.e.*, a pair (x, y) such that $y = S \cdot x$. The $\mathcal{O}\left(q^{2n/3}\right)$ pairs in $U \times V$ can be tested using the available IQMLE algorithms.

Theorem 18.3. *When* $q = 2$*, there is a* heuristic *Monte-Carlo* QMLE *algorithm that makes* $\mathcal{O}(1)$ *calls to an external* IQMLE *solver, after a preprocessing of complexity* $\mathcal{O}\left(2^{n/2}\right)$.

This second algorithm, which is the object of §18.4, reduces the number of candidate pairs to $\mathcal{O}(1)$, by using the ideas of the Weisfeiler-Lehman method more in-depth. The point is that our first algorithm only exploits the fact that the isomorphism preserves the number of adjacent nodes. Our second algorithm exploit the fact that the isomorphism preserves the actual topology of neighborhoods. One of the problems is to represent the topological structure of the neighborhoods in a way that is both independent from the labels of the vertices, and practical to deal with (*e.g.*, deterministic).

The main point is that the two graphs $G_{\mathbf{a}}$ and $G_{\mathbf{b}}$ are very sparse, as each node only has a single neighbor on average (besides itself). We thus do not expect to find other cycles than the triangles while exploring only a small fraction of the full graph. Starting a polynomial-radius Breadth-First Search from any point of the graph and removing "backwards" edges (*i.e.*, edges pointing to an already explored node) yields an unordered tree, as illustrated by Figure 18.2. These tree are somewhat random, and the distribution of the number of children of each node is known (and it is the same for all nodes). We can thus associate a random-looking *unordered, unlabeled, tree* $T(\mathbf{f}, \mathbf{x})$ to a quadratic map $\mathbf{f}$ and a vector $\mathbf{x}$ of $(\mathbb{F}_2)^n$. It is then possible to show that these trees are valid encoding of the topology of neighborhoods, since $T(\mathbf{b}, \mathbf{x}) = T(\mathbf{a}, S \cdot \mathbf{x})$.

Because most connected components in $G_{\mathbf{a}}$ and $G_{\mathbf{b}}$ are very small, and thus do not provide enough interesting information, we restrict our attention to nodes with having "large" neighborhoods (more precisely, to nodes whose neighborhood is turned into a "high" tree by the BFS), as these neighborhoods are nearly all pairwise distinct. We thus select two sets U and V of $2^{n/2}$ nodes with large neighborhoods in $G_{\mathbf{a}}$ and

$G_{\mathbf{b}}$ respectively, and the birthday paradox guarantees that there will be a right pair in $U \times V$. Because the topology of the neighborhoods contain more informtion than the degree, it can be used to efficiently identify the right pair in a nearly-unique way. We analyze the behavior of this algorithm under the *heuristic assumption* that the trees resulting from the Breadth-First Searches are critical Galton-Watson trees. The assumption that $q = 2$ is critical, as it ensures that the graph has a certain shape, and that trees can easily be found by a BFS.

Rank Distributions. Two probability distributions are ubiquitous in this chapter. Let N be smaller than q^n, $\mathbf{x}_1, \ldots, \mathbf{x}_N$ be non-zero distinct points of $(\mathbb{F}_q)^n$, and let $\mathbf{f}$ be a uniformly random quadratic map. We pay special attention to the family of random variables:

$$X_i = \dim \ker \mathrm{D}_{\mathbf{x}_i}\mathbf{f}$$

Next, let $L_1, \ldots, L_N$ a sequence of independent and uniformly distributed random endomorphisms of $(\mathbb{F}_q)^n$ such that $L_i \cdot \mathbf{x}_i = 0$. We also pay special attention to the sequence of random variables:

$$Y_i = \dim \ker L_i$$

We will show in §18.2 that the X_i and the Y_i have the same distribution. However, the Y_i are independent, while the X_i are not even pairwise independent. We will however show in §18.2.2 that the bias is extremely small, and that no algorithm can distinguish (X_1, X_2) from (Y_1, Y_2) with advantage better than $1/q^n$. This leads us to state the following conjecture.

Conjecture 18.1. No algorithm with running time $\mathcal{O}\left(Poly(n) \cdot q^{n/2}\right)$ can distinguish with non-negligible advantage between $\left(X_1, \ldots, X_{q^{n/2}}\right)$ and $\left(Y_1, \ldots, Y_{q^{n/2}}\right)$.

This computational conjecture supports the heuristic assumption that the trees obtained by locally exploring the graphs are indeed independent random trees.

18.1 The Global Strategy: Dehomogenization

As argued in the introduction of this chapter, dehomogenizing a homogeneous instance requires the knowledge of the image of S on one point. Indeed, if $\mathbf{y} = S \cdot \mathbf{x}_1$, then we define $\mathbf{a}'(\mathbf{z}) = \mathbf{a}(\mathbf{z} + \mathbf{y})$ and $\mathbf{b}'(\mathbf{z}) = \mathbf{b}(\mathbf{z} + \mathbf{x})$, and we have $T \circ \mathbf{b}' = \mathbf{a}' \circ S$. Lemma 12.2 allows us to easily compute $\mathbf{a}'$ and $\mathbf{b}'$, while ensuring that they are inhomogeneous with overwhelming probability:

$$\mathbf{a}^{(1)} = \mathrm{D}_{\mathbf{y}}\mathbf{a} \qquad \mathbf{a}^{(0)} = \mathbf{a}(S \cdot \mathbf{x})$$

$$\mathbf{b}^{(1)} = \mathrm{D}_{\mathbf{x}}\mathbf{b} \qquad \mathbf{b}^{(0)} = \mathbf{b}(\mathbf{x})$$

This immediately shows that dehomogenized instances are *not* random inhomogeneous instances. First of all, their linear and quadratic homogeneous components are correlated. Much worse, since we assume q to be a power of two, their linear homogeneous components are *always singular*. This means for instance that the inversion-free to-n-fro algorithm (aka Algorithm 17.1) cannot successfully solve them, and that the pencil-based algorithm (aka Algorithm 17.2) has a lower success probability on dehomogenized instances than on random instances.

It is possible to make the pencil-based IQMLE algorithm work smoothly in this context though. The problem is that the algorithm will only work if the dehomogenized instance is *nice* (as defined in §17.2). It is fortunately not difficult to make the dehomogenized instances nice. We will say that $\mathbf{x}$ is *nice with* $\mathbf{f}$ if $\mathbf{f}(\mathbf{x})$ is non-zero and belongs to the image of $\mathrm{D}_{\mathbf{x}}\mathbf{f}$. The dehomogenized instance is therefore nice when $\mathbf{b}$ is nice with $\mathbf{x}$. In this case, we can run the pencil-based IQMLE algorithm on the dehomogenized instance, and we know that it will eventually succeed. It is therefore sufficient to choose a vector $\mathbf{x}$ that is nice with $\mathbf{b}$. This results in algorithm 18.1.

This algorithm is almost easy to analyze, and the only question not quickly receiving an obvious answer is the number of iteration of the first loop (that selects $\mathbf{x}$). We will assume that $\mathbf{x}$ and $\mathbf{b}$ being random, $\mathbf{b}(\mathbf{x})$ should be non-zero with probability $1 - q^{-n}$. Lemma 17.2 tells us that $\mathrm{D}_{\mathbf{x}}\mathbf{b}$ is a random linear map vanishing on x, and lemma 10.26 combined with conjecture 10.1 gives us the probability that each one contains $\mathbf{b}(\mathbf{x})$ in its image. Therefore the expected number of iterations of this loop is:

$$\mathbb{E}\left[\#\text{iterations}\right] = q - \frac{1}{q} + \frac{1}{q^3} - \frac{1}{q^4} + \frac{1}{q^5} + \mathcal{O}\left(\frac{1}{q^6}\right)$$

This shows that with overwhelming probability, the cost of determining $\mathbf{x}$ in Algorithm 18.1 is negligible in front of that of invoking Pencil-IQMLE about $q^n - 1$ times. The same reasoning carries over with minor tweaks to fields of odd characteristic.

Algorithm 18.1 Semi-trivial algorithm based on dehomogenization.

function EXHAUSTIVE-DEHOMOGENIZATION($\mathbf{a}, \mathbf{b}$)
  repeat
    Pick a random vector $\mathbf{x} \in (\mathbb{F}_q)^n$
  until $\mathbf{b}(\mathbf{x}) \neq 0$ and there exist $\mathbf{z} \in (\mathbb{F}_q)^n$ such that $\mathbf{b}(\mathbf{x}) = \mathrm{D}_{\mathbf{x}}\mathbf{b} \cdot \mathbf{z}$
  for all $0 \neq \mathbf{y} \in (\mathbb{F}_q)^n$ **do**
    $\mathbf{a}'(\mathbf{z}) \leftarrow \mathbf{a}(\mathbf{z} + \mathbf{y})$
    $\mathbf{b}'(\mathbf{z}) \leftarrow \mathbf{b}(\mathbf{z} + \mathbf{x})$
    $(S, T) \leftarrow$ PENCIL-IQMLE($\mathbf{a}', \mathbf{b}'$)
    if solution found **then return** (S, T)
  end for
  return "Not Equivalent"
end function

18.2 Distribution of the Rank of the Differential

In this section we study the distributions of the two families of random variables X_i and Y_i defined in the introduction of this chapter. It follows from lemma 10.19 (with $s = 1$) and lemma 17.2 that when q is even,

$$\mathbb{P}[X_i = k] = \mathbb{P}[Y_i = k] = \frac{\lambda(n)\lambda(n-1)}{\lambda(k)\lambda(k-1) \cdot \lambda(n-k)} \cdot q^{-k(k-1)}, \qquad 1 \leq k \leq n$$

The X_i and Y_i have the same distribution because given any non-zero vector $\mathbf{x}$, and any linear map L vanishing on $\mathbf{x}$, the number of quadratic maps $\mathbf{f}$ such that $\mathrm{D}_{\mathbf{x}}\mathbf{f} = L$ is independent from the choice of $\mathbf{x}$ and L (as per lemma 17.2).

18.2.1 Correlation between the X_i's

If the X_i were independent, we would have $\mathbb{P}[X_i = r, X_j = s] = \mathbb{P}[X_j = r]\,\mathbb{P}[X_j = s]$. It is unfortunately not the case, and the bias comes from the symmetry of the differential: $\mathbf{x}_i \in \ker \mathrm{D}_{\mathbf{x}_j}\mathbf{f}$ is equivalent to $\mathbf{x}_j \in \ker \mathrm{D}_{\mathbf{x}_i}\mathbf{f}$, and therefore the two random variables X_i and X_j are correlated. It is nevertheless possible to compute the joint distribution.

Lemma 18.4. *When q is a power of two, then for all $r, s \in [1; n]$:*

$$\frac{\mathbb{P}[X_i = r, X_j = s]}{\mathbb{P}[X_i = r]\,\mathbb{P}[X_j = s]} = \frac{q^n}{q^n - 1}\left(1 - \frac{q^r + q^s - 2q}{q^n - q} + \frac{q^n}{q^n - q} \cdot \frac{q^{r+s} - q^{r+1} - q^{s+1} + q^2}{q^n - q}\right)$$

Proof. The proof proceeds in two steps. First, we claim that if $\mathbf{x} \neq \mathbf{y}$ are both non-zero, then the distribution of $(\dim\ker \mathrm{D}_{\mathbf{x}}\mathbf{f}, \dim\ker \mathrm{D}_{\mathbf{y}}\mathbf{f})$ when $\mathbf{a}$ is random is the same as the distribution of $(\dim\ker L, \dim\ker L')$ where (L, L') are random linear maps such that $L(\mathbf{x}) = 0, L'(\mathbf{y}) = 0$ and $L(\mathbf{y}) = L'(\mathbf{x})$. This assertion is in fact demonstrated in [DGS06] (and it is lemma 12 in [Dub07]).

Next, we determine the distribution of $(\dim\ker L, \dim\ker L')$ where (L, L') are random linear maps such that $L(\mathbf{x}) = L'(\mathbf{y}) = 0$ and $L(\mathbf{y}) = L'(\mathbf{x})$. First of all, there are $q^{n(2n-3)}$ such pairs (L, L'). It remains to count the number of pairs satisfying the rank conditions. Recall from §10.7 that we denote by $D(n, k, s)$ the set of all $n \times n$ matrices with a kernel of dimension k containing a fixed subspace of dimension s.

We first observe that there are $D(n, r, 2) \cdot D(n, s, 2)$ pairs such that both L and L' vanish on both x and y. Next, we count the pairs such that $L(\mathbf{y}) = L'(\mathbf{x}) \neq 0$. The first member of the pair can be chosen arbitrarily as long as it is of rank r, that $L(\mathbf{x}) = 0$ and that $L(\mathbf{y}) \neq 0$. There are therefore $D(n, r, 1) - D(n, r, 2)$ possible choices. For L', the image of L' on $\mathbf{x}$ is fixed, therefore only a fraction $1/(q^n - 1)$ of the matrices with rank s vanishing on $\mathbf{y}$ and non-vanishing on $\mathbf{x}$ are admissible.

All-in-all this yields:

$$\mathbb{P}\begin{bmatrix} X(\mathbf{x}) = r \\ X(\mathbf{y}) = s \end{bmatrix} = \frac{1}{q^{n(2n-3)}}\Bigg(D(n, r, 2) \cdot D(n, s, 2) + \frac{[D(n, r, 1) - D(n, r, 2)] \cdot [D(n, s, 1) - D(n, s, 2)]}{q^n - 1}\Bigg)$$

It follows from lemma 10.19, using easy q-binomial manipulations that:

$$D(n, r, 2) = \frac{[r-1]_q}{[n-1]_q} \times D(n, r, 1) = \frac{q^r - q}{q^n - q} \times D(n, r, 1)$$

This allows the "simplification":

$$\mathbb{P}\left[\begin{array}{l} X(\mathbf{x})=r \\ X(\mathbf{y})=s \end{array}\right] = \frac{D(n,r,1)\cdot D(n,s,1)}{q^{n(2n-3)}}\left(\frac{q^r-q}{q^n-q}\cdot\frac{q^s-q}{q^n-q}+\frac{\left[1-\frac{q^r-q}{q^n-q}\right]\cdot\left[1-\frac{q^s-q}{q^n-q}\right]}{q^n-1}\right)$$

Now, we know that $\mathbb{P}\left[X(\mathbf{x})=r\right] = D(n,r,1)\cdot q^{-n(n-1)}$. Therefore we find after some other manipulations:

$$\frac{\mathbb{P}\left[\begin{array}{l} X(\mathbf{x})=r \\ X(\mathbf{y})=s \end{array}\right]}{\mathbb{P}\left[X(\mathbf{x})=r\right]\mathbb{P}\left[X(\mathbf{y})=s\right]} = \frac{q^n}{q^n-1}\left(1-\frac{q^r+q^s-2q}{q^n-q}+\frac{q^n}{q^n-q}\cdot\frac{q^{r+s}-q^{r+1}-q^{s+1}+q^2}{q^n-q}\right)$$

□

18.2.2 Distinguishing the two distributions

The bias between the X_i's and the Y_i's seems quite small. To formalize this notion, we consider the following game, parametrized by an integer N:

1. The Challenger flips a coin b
2. If $b=0$, the Challenger samples a tuple $(X_1,\ldots,X_N)$, and if $b=1$ she samples $(Y_1,\ldots,Y_N)$
3. The Challenger sends her tuple to the adversary
4. The adversary then outputs her guess for the value of b

We first note that given $N=1$ sample, no adversary can do better than random guessing, because what she receives always has the same distribution. However, given $N=2$ samples, an adversary can try to detect the eventual presence of a correlation between the two samples thanks to her knowledge of lemma 18.4. It is however possible to show that when $N=2$, all adversaries only achieve a very small advantage over random guessing. To establish this fact, we build an *optimal* Neyman-Pearsons (maximum-likelihood) distinguisher $\mathcal{D}_2$ between (X_1,X_2) and (Y_1,Y_2), and we upper-bound its success probability. It follows from lemma 18.4 that:

$$\mathbb{P}\left[X_i=r, X_j=s\right] > \mathbb{P}\left[X_i=r\right]\mathbb{P}\left[X_j=s\right] \Longleftrightarrow \left\{\begin{array}{c} r=s=1 \\ \text{or} \\ r\geq 2 \text{ and } s\geq 2 \end{array}\right.$$

Therefore our maximum-likelihood distinguisher works as follows:

$$\mathcal{D}_2(r,s) = \begin{cases} 0 & \text{if } (r=s=1) \text{ or } (r\geq 2 \text{ and } s\geq 2) \\ 1 & \text{otherwise} \end{cases}$$

Lemma 18.5. *The success probability of $\mathcal{D}_2$ is:*

$$\varepsilon = \frac{1}{2}+\lambda(n)^2\frac{q^2\left(q^{n+1}+q^n-q\right)}{\left(q^n-1\right)\left(q^n-q\right)\left(q-1\right)^2} = \frac{1}{2}+\mathcal{O}\left(\frac{1}{q^n}\right)$$

Proof. The generic expression of the success probability is given by:

$$\varepsilon = \frac{1}{2}\left(1-\mathbb{P}\left[\mathcal{D}_2=1 \;\middle|\; b=0\right]+\mathbb{P}\left[\mathcal{D}_2=1 \;\middle|\; b=1\right]\right)$$

Let us assume that $\mathcal{D}_2$ receives the pair of integers (r,s). Taking into account the description of $\mathcal{D}_2$ gives:

$$\begin{aligned} 2\varepsilon \;=\; 1- &\;\mathbb{P}\left[(r=1\wedge s\geq 2)\vee(r\geq 2\wedge s=1) \;\middle|\; b=0\right] \\ + &\;\mathbb{P}\left[(r=1\wedge s\geq 2)\vee(r\geq 2\wedge s=1) \;\middle|\; b=1\right] \end{aligned}$$

Exploiting the knowledge of b yields:

$$\varepsilon = \frac{1}{2}+\sum_{s=2}^{n}\Big(\mathbb{P}\left[Y_i=1,Y_j=s\right]-\mathbb{P}\left[X_i=1,X_j=s\right]\Big)$$

Now, exploiting the independence of the Y_i, lemma 18.4, and the fact that the X_i's and Y_i's have the same distribution, we find:

$$\varepsilon = \frac{1}{2}+\mathbb{P}\left[Y_i=1\right]\sum_{s=2}^{n}\mathbb{P}\left[Y_j=s\right]\left[1-\frac{q^n}{q^n-1}\left(1-\frac{q^s-q}{q^n-q}\right)\right]$$

To lighten the notational burden, let us write $\alpha = \mathbb{P}[Y_i = 1]$. We thus have:

$$\varepsilon = \frac{1}{2} + \alpha \sum_{s=2}^{n} \mathbb{P}[Y_j = s] - \frac{\alpha \cdot q^n}{q^n - 1} \left(\sum_{s=2}^{n} \mathbb{P}[Y_j = s] - \sum_{s=2}^{n} \mathbb{P}[Y_j = s] \frac{q^s - q}{q^n - q} \right)$$

This can be further simplified into:

$$\varepsilon = \frac{1}{2} + \alpha(1-\alpha) - \frac{\alpha \cdot q^n}{q^n - 1} \left(1 - \alpha - \frac{1}{q^n - q} \left(\sum_{s=2}^{n} \mathbb{P}[Y_j = s] q^s - q \cdot (1-\alpha) \right) \right)$$

Lemma 10.20 shows that

$$\sum_{s=0}^{n} \mathbb{P}[Y_j = s] q^s = q + 1 - \frac{1}{q^{n-1}}.$$

Using this identity, we obtain :

$$\varepsilon = \frac{1}{2} + \alpha(1-\alpha) - \frac{\alpha \cdot q^n}{q^n - 1} \left(1 - \alpha - \frac{1}{q^n - q} \left(1 + q \cdot \alpha - \frac{1}{q^{n-1}} \right) \right)$$

Factoring α and simplifying (a lot!) yields:

$$\varepsilon = \frac{1}{2} + \alpha^2 \frac{q^n - q + q^{n+1}}{(q^n - 1)(q^n - q)}$$

Lemma 10.19 tells us that $\alpha = \lambda(n)/\lambda(1) = \lambda(n)\frac{q}{q-1}$. The announced result follows. □

Justification of Conjecture 18.1 Building a provably efficient distinguisher between the two tuples $(X_1, \ldots, X_N)$ and $(Y_1, \ldots, Y_N)$ seems difficult. However, the distinguisher $\mathcal{D}_2$ could be run on the N^2 pairs, and it seems plausible that the advantage of doing so would scale linearly with the number of pairs. In view of lemma 18.5, an advantage of order one should be achieved with q^n pairs, and these can be obtained with $N = q^{n/2}$ samples. However, processing the q^n pairs should require at least q^n elementary operations. Conjecture 18.1 relies on the idea that if even if the right amount of data is available, it cannot be processed in the limited amount of time allowed.

18.3 Sieving With Adjacent Vertices

In this section we discuss Algorithm 18.2. It is made of an auxiliary function, RankList and a main function Rank-QMLE. The function RankList$(\mathbf{f}, \ell, k)$ generates by rejection sampling an ℓ-element random subset of $(\mathbb{F}_q)^n$ composed of vectors $\mathbf{x}$ satisfying $\dim \ker \mathrm{D}_{\mathbf{x}}\mathbf{f} = k$. For the sake of convenience, let us define:

$$R_k(\mathbf{f}) = \{\mathbf{x} \in (\mathbb{F}_q)^n \mid \dim \ker \mathrm{D}_{\mathbf{x}}\mathbf{f} = k\}$$

Thus, RankList$(\mathbf{f}, \ell, k)$ returns a uniformly random subset of $R_k(\mathbf{f})$ of cardinality ℓ. The main function, Rank-QMLE, first uses RankList to generate two random subsets U and V of $R_k(\mathbf{b})$ and $R_k(\mathbf{a})$ respectively (with $k = \sqrt{n/3}$), of cardinality $2q^{n/3}$. It then enumerates all the pairs $(x, y) \in U \times V$, and checks whether $\mathbf{y} = S \cdot \mathbf{x}$ by building the corresponding inhomogeneous instance $(\mathbf{a}', \mathbf{b}')$ and sending it to an external IQMLE solver. The IQMLE solver will at some point return a (valid) solution as soon as there is a *right pair* (*i.e.*, such that the relation $\mathbf{y} = S \cdot \mathbf{x}$ actually holds) in $U \times V$. In the remaining of this section, we will first prove theorem 18.2 by showing that Rank-QMLE returns a correct pair (S, T) when $\mathbf{a}$ and $\mathbf{b}$ are equivalent with probability greater than $1 - 1/e \approx 63\%$. We will then attempt to replace the generic IQMLE-Solver by Pencil-IQMLE (*i.e.*, Algorithm 17.2). We will adapt Algorithm 18.2 accordingly, discuss the complexity of the resulting combination and present practical results.

18.3.1 Proof of Theorem 18.2 : Generic Analysis

When $\mathbf{a}$ and $\mathbf{b}$ are not equivalent, the derived inhomogeneous instance cannot have a solution, and thus Rank-QMLE always returns "probably not equivalent". When $\mathbf{a}$ and $\mathbf{b}$ are equivalent, on the other hand, Rank-QMLE will only find a valid solution if there is a pair $(\mathbf{x}, \mathbf{y})$ in $U \times V$ such that $\mathbf{y} = S \cdot \mathbf{x}$.

The core idea the algorithm relies on is that equation (18.2) implies that the two linear maps $\mathrm{D}_{\mathbf{x}}\mathbf{b}$ and $\mathrm{D}_{S\cdot\mathbf{x}}\mathbf{a}$ have the same rank. In other terms, for any k, the unknown (invertible) matrix S maps the elements of $R_k(\mathbf{b})$ to those of $R_k(\mathbf{a})$, which means that we may restrict our attention to one of the $R_k(\mathbf{a})$ for a well-chosen value of k. Another way of expressing the same idea would be to say that we restrict our attention to vertices of $G_{\mathbf{a}}$ and $G_{\mathbf{b}}$ having exactly $q^k - q^2$ adjacent nodes.

Algorithm 18.2 Rank/Birthday Based Algorithm

```
1: function RankList(f, ℓ, k)
2:     L ← {}
3:     repeat
4:         repeat
5:             x ← random element of (F_q)^n
6:         until dim ker D_x f = k
7:         L ← L ∪ {x}
8:     until |L| = ℓ
9:     return L
10: end function
11: function Rank-QMLE(a, b)
12:     U ← RankList(a, √2 q^{n/3}, √(n/3))
13:     V ← RankList(b, √2 q^{n/3}, √(n/3))
14:     for all (x, y) ∈ U × V do
15:         a'(z) ← a(z + y)
16:         b'(z) ← b(z + x)
17:         (S, T) ← IQMLE-Solver(a', b')
18:         if solution is found then return (S, T)
19:     end for
20:     return "Probably not equivalent"
21: end function
```

The sets $R_k(\mathbf{a})$ form a partition of $(\mathbb{F}_q)^n$ when k ranges across $[1; n]$, and interestingly they have different sizes. By linearity of the expectation, we find that $\mathbb{E}\left[|R_k(\mathbf{a})|\right] = q^n \cdot \mathbb{P}\left[X_i = k\right]$, and the distribution of the X_i's then gives us the expected cardinality of $R_k(\cdot)$. If $\mathbf{a}$ is a random quadratic map, then:

$$q^{n-k(k-1)} \leq \mathbb{E}\left[|R_k(\mathbf{a})|\right] \leq 2q^{n-k(k-1)}.$$

It follows that when $k = \sqrt{n/3}$, the expected cardinality of $R_k(\mathbf{a})$ is less than $2 \cdot q^{2n/3}$ (up to rounding errors). The two random subsets U and V defined in Rank-QMLE both contain $\sqrt{2} \cdot q^{n/3}$ elements, and this is precisely the square root of the (upper-bound on the) expected size of $R_k(\mathbf{a})$.

Both U and V have more than $\sqrt{|R_k(\mathbf{a})|}$ elements, and thus lemma 3.2 ensures that Algorithm 18.2 succeeds with probability greater than $1 - 1/e$ (the blue balls are the images of the vectors in U by S, while the red balls are the vectors in V).

Let us now complete the proof of theorem 18.2. We remark that $U \times V$ contains $\mathcal{O}\left(q^{2n/3}\right)$ elements, so that the main loop invokes the external IQMLE solver $\mathcal{O}\left(q^{2n/3}\right)$ times. It remains to show that determining U and V can be done with $\mathcal{O}\left(q^{2n/3}\right)$ operations. Turning our attention to RankList, we first argue that as long as ℓ does not exceed $\sqrt{|R_k(\mathbf{f})|}$, then the number of iterations of the outer loop will be $\mathcal{O}(\ell)$. Indeed, the inner-loop of lines 4–6 chooses a random vector $\mathbf{x} \in R_k(\mathbf{f})$, and "collisions" inside $R_k(\mathbf{f})$ are unlikely before $\sqrt{|R_k(\mathbf{f})|}$ random vectors are sampled. We also find that the number of iterations of the inner loop follows a geometric distribution of parameter $|R_k(\mathbf{f})|/q^n \approx q^{-n/3}$, so that the expected number of iterations of the inner loop is $\mathcal{O}\left(q^{n/3}\right)$. This means that in average each of the two invocations of RankList will perform $\mathcal{O}\left(q^{2n/3}\right)$ matrix operations.

18.3.2 Specialization to Pencil-IQMLE

We now move on to replace the abstract and generic IQMLE-Solver of line 17 by our best candidate for the job, namely Pencil-IQMLE (Algorithm 17.2). This raises certain problems though. First of all, the inhomogeneous instances are not random, if only because the kernel of the linear component has dimension $\sqrt{n/3}$. Note that this rules out the inversion-free to-n-fro algorithm, which would fail systematically. It is quite difficult to say anything meaningful regarding the performance of the Gröbner-basis algorithm of [FP06] on these special instances, but we have all reasons to believe that it would be slower than on random instances, even if we do not know by how much. We thus stick to Pencil-IQMLE, because it should be faster and we understand it better.

Still, the failure probability of Pencil-IQMLE should be *much* higher on these "low-rank" instances than on random instances. Some specific counter-measures have to be adopted, otherwise Pencil-IQMLE might fail on the probably-single right-pair of $U \times V$, and the whole process would fail. To circumvent

this problem, we patch the RankList function so that it only selects vectors $\mathbf{x}$ making the dehomogenized instances *nice*. The patched function is shown in Algorithm 18.3. The corresponding patched definition of R_k is:

$$R'_k(\mathbf{f}) = \{\mathbf{x} \in (\mathbb{F}_q)^n \mid \dim\ker \mathrm{D}_{\mathbf{x}}\mathbf{f} = k \text{ and } \mathbf{f}(\mathbf{x}) \neq 0 \text{ and } \mathbf{f}(\mathbf{x}) \in \mathrm{Im}\,\mathrm{D}_{\mathbf{x}}\mathbf{f}\}$$

Algorithm 18.3 Specialized function for Pencil-IQMLE.

```
1: function RankList(f, ℓ, k)
2:     L ← {}
3:     repeat
4:         repeat
5:             x ← random element of (F_q)^n
6:         until dim ker D_x f = k and b(x) ≠ 0 and b(x) ∈ Im D_x f
7:         L ← L ∪ {x}
8:     until |L| = ℓ
9:     return L
10: end function
```

It remains to estimate the complexity of the patched RankList function, and that of Pencil-IQMLE on the corresponding instances. The parameters of the algorithm will then be modified to minimize the total running time.

Complexity of the Patched RankList. When q is a power of two, the differential of a random quadratic map at a random non-zero point $\mathbf{x}$ is a random linear map vanishing on $\mathbf{x}$. The number of linear maps whose kernel has dimension k and contains a given vector, and whose image contains another fixed vector is given by lemma 10.23 (with $s = t = 1$), and is:

$$\binom{n-1}{k-1}_q \binom{n-1}{k}_q \langle\!\langle n-k \rangle\!\rangle_q !$$

Exploiting the fact that $\binom{n-1}{k}_q$ has order $q^{k(n-k)}$ reveals that the expected cardinality of $R'_k(\mathbf{a})$ is essentially q^{n-k^2}, up to small constant multiplicative factors. It also follows that for each iteration of the outer loop of the patched RankList, there will be on average q^{k^2} iterations of the inner loop.

As previously, we choose the size of U and V to be the square root of the expected cardinality of R'_k, for some value of k that we will determine. The complexity of building U and V is therefore $\mathcal{O}\left(q^{n/2+k^2/2}\right)$ matrix operations.

Complexity of Pencil-IQMLE. When the homogeneous linear component has a kernel of dimension k, then the main loop of Pencil-IQMLE performs q^k iterations. We will therefore assume that the complexity of running Pencil-IQMLE on these particular instances is that of q^k matrix operations (this is a somewhat crude approximation). Testing all the $q^n - k^2$ pairs in $U \times V$ therefore costs roughly $\mathcal{O}\left(q^{n-k^2+k}\right)$ matrix operations.

Adapting the Parameters. To minimize the total running time, we must therefore choose k such that:

$$\frac{n}{2} + \frac{k^2}{2} \approx n - k^2 + k$$

And the solution is:

$$k = \sqrt{n/3} - \frac{1}{3} + o(1)$$

Leading to a total complexity of roughly $\mathcal{O}\left(q^{2n/3+\sqrt{25n/27}}\right)$ matrix operations.

18.3.3 Practical Results.

We have implemented the combination of Algorithm 18.2 with Pencil-IQMLE inside the MAGMA computer algebra system, running on one core of a 2.8 Ghz Xeon machine. As shown in Table 18.1, we found out that in practice it is difficult to balance the cost of the two parts of the algorithm because k can only take $\sqrt{n}$ integer values. We could nevertheless verify in practice that the complexity of building the lists and the expected number of right pairs in them is consistent with our theoretical results. The source code is in the public domain, and is available at:

n	q	generating U and V	total time	target rank	$\lvert U\rvert$	$\lvert\mathcal{P}\rvert$
16	2	0s	68s	3	1	4
22	2	28s	35000s	4	13	400
28	2	4913s	8087s	5	8	64

Table 18.1: Experimental results on Rank-QMLE.

`http://www.di.ens.fr/~bouillaguet/implem/rank-qmle.magma`

18.4 Sieving Using Whole Neighborhoods

In the remaining of this chapter, we assume that $q = 2$, and we investigate Algorithm 18.4, which is more efficient and more sophisticated. Note that we still use the notation q, as it helps distinguishing numerical constants depending on the size of the field and "accidental" ones.

Rank-QMLE used the rank of the differential as a *discriminating function* that could tell apart wrong pairs $\mathbf{y} \neq S \cdot \mathbf{x}$. However, this discriminating function splits $(\mathbb{F}_q)^n$ in only $\sqrt{n}$ classes, resulting in many false positives, and an exponential number of candidate pairs have to be tested. To improve on the previous approach, we seek a more precise discriminating function in order to partition the search space into exponentially many classes. A possible solution consists in exploring larger neighborhoods of $G_{\mathbf{a}}$ and $G_{\mathbf{b}}$ than the immediately adjacent nodes. However, representing this more complex topological structure is more difficult than just storing the number of neighbors. Our new discriminating function performs a radius-k BFS starting from a given node $\mathbf{x}$, and generates the corresponding tree (assuming no cycle occurs and removing backwards edges). It has the following recursive definition, where the double-brace notation denotes a *multiset*:

$$\begin{aligned}
\textsc{Tree}^{[k]}(\mathbf{f}, \mathbf{x}) &= \textsc{Node}^{[k]}(\mathbf{f}, 0, \mathbf{x}) \\
\textsc{Node}^{[k]}(\mathbf{f}, \mathbf{x}, \mathbf{y}) &= \left\{\!\!\left\{ \textsc{Node}^{[k-1]}(f, \mathbf{y}, \mathbf{z}) \;\middle|\; \mathbf{z} \in \ker \mathrm{D}_{\mathbf{y}}\mathbf{f} \text{ and } \mathbf{z} \notin \langle \mathbf{x}, \mathbf{y} \rangle \right\}\!\!\right\} \\
\textsc{Node}^{[0]}(\mathbf{f}, \mathbf{x}, \mathbf{y}) &= \emptyset
\end{aligned}$$

After noting that $\textsc{Tree}^{[k]}$ can be evaluated in finite time for any given value of k, we claim that it can be used as a discriminating function.

Lemma 18.6. *For all $k \in \mathbb{N}$ and $0 \neq \mathbf{x} \in (\mathbb{F}_2)^n$, we have $\textsc{Tree}^{[k]}(\mathbf{b}, \mathbf{x}) = \textsc{Tree}^{[k]}(\mathbf{a}, S \cdot \mathbf{x})$.*

Proof. We show by induction on k that $\textsc{Node}^{[k]}(\mathbf{b}, \mathbf{x}, \mathbf{y}) = \textsc{Node}^{[k]}(\mathbf{a}, S \cdot \mathbf{x}, S \cdot \mathbf{y})$, as this implies the result of the lemma. This is trivially true when $k = 0$, and when $k > 0$, equation (18.2) implies that $\ker \mathrm{D}_{\mathbf{y}}\mathbf{b}$ is transformed into $\ker \mathrm{D}_{S\cdot\mathbf{y}}\mathbf{a}$ by S. This means that it is possible to write $\ker \mathrm{D}_{\mathbf{y}}\mathbf{b} = \{u_1, \ldots, u_\ell\}$ and $\ker \mathrm{D}_{S\cdot\mathbf{y}}\mathbf{a} = \{v_1, \ldots, v_\ell\}$ such that $v_i = S \cdot u_i$. Then:

$$\begin{aligned}
\textsc{Node}^{[k]}(\mathbf{b}, \mathbf{x}, \mathbf{y}) &= \left\{\!\!\left\{ \textsc{Node}^{[k-1]}(\mathbf{b}, \mathbf{y}, u_i) \;\middle|\; i \in [1;\ell] \text{ and } u_i \notin \langle \mathbf{x}, \mathbf{y} \rangle \right\}\!\!\right\} \\
\textsc{Node}^{[k]}(\mathbf{a}, S\cdot\mathbf{x}, S\cdot\mathbf{y}) &= \left\{\!\!\left\{ \textsc{Node}^{[k-1]}(\mathbf{a}, S\cdot\mathbf{y}, S\cdot u_i) \;\middle|\; i \in [1;\ell] \text{ and } S\cdot u_i \notin \langle S\cdot\mathbf{x}, S\cdot\mathbf{y} \rangle \right\}\!\!\right\}
\end{aligned}$$

In the definition of the second multiset, the condition $S \cdot u_i \notin \langle S \cdot \mathbf{x}, S \cdot \mathbf{y} \rangle$ is clearly equivalent to $u_i \notin \langle \mathbf{x}, \mathbf{y} \rangle$, because S is invertible. The induction hypothesis tells us that

$$\textsc{Node}^{[k-1]}(\mathbf{b}, \mathbf{y}, u_i) = \textsc{Node}^{[k-1]}(\mathbf{a}, S \cdot \mathbf{y}, S \cdot u_i),$$

and the two multisets are in fact equal. □

The purpose of the condition "$\mathbf{z} \notin \langle \mathbf{x}, \mathbf{y} \rangle$" in the definition of Node is to keep the multisets as small as possible without discarding relevant information. For instance, zero always belong to $\ker \mathrm{D}_{\mathbf{x}}\mathbf{f}$, just because the kernel is a vector space, yet this does not help in discriminating (this is why zero was removed from the graphs). In addition, because the differential is symmetric, then $\mathbf{y} \in \ker \mathrm{D}_{\mathbf{x}}\mathbf{f}$ implies that $\mathbf{x} \in \ker \mathrm{D}_{\mathbf{y}}\mathbf{f}$. Finally, since $q = 2$, we always have $\mathbf{x} \in \ker \mathrm{D}_{\mathbf{x}}\mathbf{a}$. We thus remove from the multisets values that we know would be here independently of $\mathbf{a}$ and $\mathbf{b}$. It is equivalent to say that we do not record self-edges in $G_{\mathbf{a}}$ and $G_{\mathbf{b}}$, that the graph is undirected, and that we do not want to follow "backwards" edges during the BFS.

Algorithm 18.4 Branching Process Based Algorithm.

function TreeList($\mathbf{f}, \ell$)
  $L \leftarrow \{\}$
  repeat
    repeat
      $\mathbf{x} \leftarrow$ random element of $(\mathbb{F}_2)^n$
      $T \leftarrow \text{Tree}^{[n \log n]}(\mathbf{f}, \mathbf{x})$
    until Height$(T) = n \log n$
    and SpineDecomp$(T, n\sqrt{\log n})$
    $L \leftarrow L \cup \{\mathbf{x}\}$
  until $|L| = \ell$
  return L
end function

function Branching-QMLE($\mathbf{a}, \mathbf{b}$)
  $U \leftarrow$ TreeList $\left(\mathbf{a}, q^{n/2}/\sqrt{n}\right)$
  $V \leftarrow$ TreeList $\left(\mathbf{b}, q^{n/2}/\sqrt{n}\right)$
  $\mathcal{P} \leftarrow \{(\mathbf{x}, \mathbf{y}) \in U \times V \mid \text{Tree}(\mathbf{a}, \mathbf{x}) = \text{Tree}(\mathbf{b}, \mathbf{y})\}$
  for all $(x, y) \in \mathcal{P}$ **do**
    $\mathbf{a}'(\mathbf{z}) \leftarrow \mathbf{a}(\mathbf{z} + \mathbf{y})$
    $\mathbf{b}'(\mathbf{z}) \leftarrow \mathbf{b}(\mathbf{z} + \mathbf{x})$
    $(S, T) \leftarrow$ IQMLE-Solver$(\mathbf{a}', \mathbf{b}')$
    if solution is found **then return** (S, T)
  end for
  return "Probably not equivalent"
end function

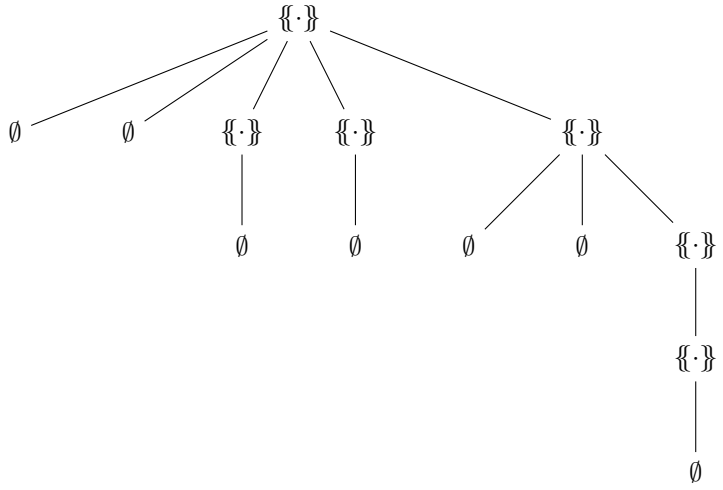


Figure 18.3: A representation of $\{\!\{\emptyset, \emptyset\, \{\!\{\emptyset\}\!\}\, \{\!\{\emptyset\}\!\}, \{\!\{\emptyset, \emptyset, \{\!\{\{\!\{\emptyset\}\!\}\}\!\}\}\!\}$.

The connection between the nested multisets returned by $\text{Tree}^{[k]}$ and the trees resulting for locally breadth-first searching the graphs is that the nested multisets are naturally represented by *unordered tree*. For instance, $\{\!\{\emptyset, \emptyset\, \{\!\{\emptyset\}\!\}\, \{\!\{\emptyset\}\!\}, \{\!\{\emptyset, \emptyset, \{\!\{\{\!\{\emptyset\}\!\}\}\!\}\}\!\}$ is the tree of height 4 with 13 nodes shown in Figure 18.3 (with the convention that the root has height 0). We denote by Height the function that returns the height of a tree.

We say that a tree has a *unique spine decomposition* (or *unique backbone decomposition*) if there is a unique path starting from the root that reach a leaf of maximal height (this implies that there is a unique leaf of maximal height). We also say that a tree has a unique spine decomposition *up to height* k if there is a unique path starting from the root and reaching height k that extends to a path reaching nodes of maximal height. Figure 18.4 shows a tree with a spine decomposition up to a certain level. The predicate SpineDecomp(T, k) returns **true** if and only if the tree T admits a unique spine decomposition up to height k. Note that this predicate can easily be evaluated in linear time.

Algorithm 18.4 makes use of an auxiliary function, TreeList$(\mathbf{f}, \ell)$, which generates by rejection sampling

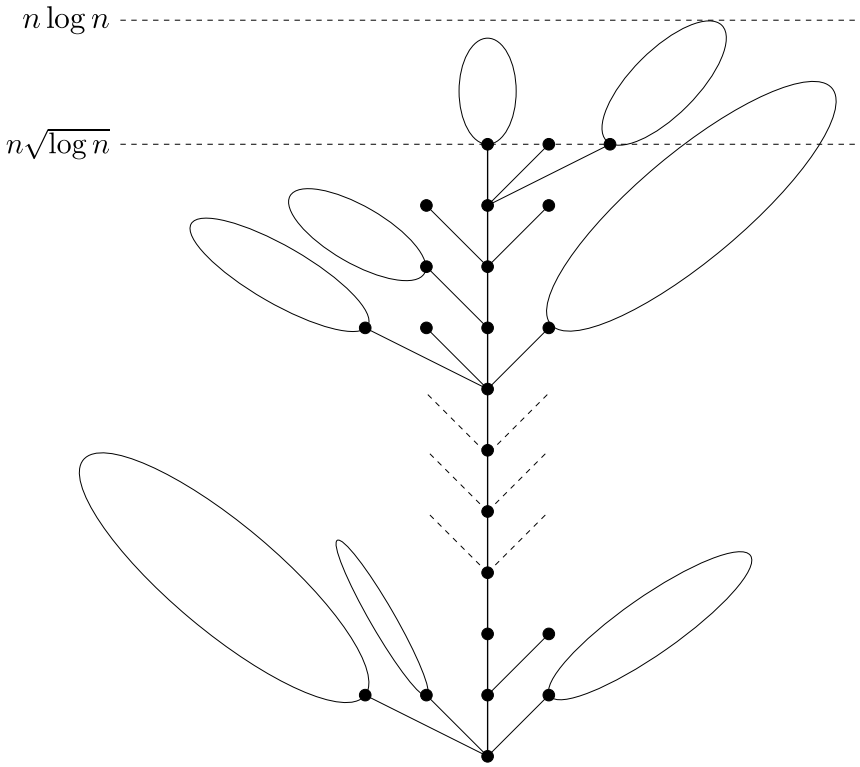


Figure 18.4: A Tree of height $n \log n$ with a spine decomposition up height $n\sqrt{\log n}$ (there has a unique path starting from the root and reaching height $n\sqrt{\log n}$ that extends to a path of maximal length).

a random subset of $(\mathbb{F}_2)^n$ composed of ℓ vectors $\mathbf{x}$ such that $\textsc{Tree}^{[n \log n]}(\mathbf{f}, \mathbf{x})$ has height $n \log n$ and has a unique spine decomposition up to height $n\sqrt{\log n}$.

It seems natural to question how good a discriminating function $\textsc{Tree}^{[k]}$ is, or, more precisely, how often do we have $\textsc{Tree}^{[k]}(\mathbf{b}, \mathbf{x}) = \textsc{Tree}^{[k]}(\mathbf{a}, \mathbf{y})$ when $\mathbf{y} \neq S \cdot \mathbf{x}$. In Algorithm 18.4, this directly influences the cardinality of $\mathcal{P}$, and thus the number of inhomogeneous instances to discharge to the IQMLE solver. In §18.4.1 we will show that when $\mathbf{x}$ is randomly chosen, then $\textsc{Tree}^{[k]}(\mathbf{b}, \mathbf{x}) = \emptyset$ with high probability, making the discriminating function essentially useless over the whole $(\mathbb{F}_2)^n$. This is why in TreeList we restrict our attentions to the following subset of $(\mathbb{F}_2)^n$:

$$H(\mathbf{f}) = \left\{ \mathbf{x} \in (\mathbb{F}_2)^n \;\middle|\; \begin{array}{l} \mathcal{T} \leftarrow \textsc{Tree}^{[n \log n]}(\mathbf{f}, \mathbf{x}), \\ \quad \textsc{Height}(\mathcal{T}) = n \log n, \\ \quad \textsc{SpineDecomp}\left(\mathcal{T}, n\sqrt{\log n}\right) = \textbf{true} \end{array} \right\}$$

We will show in §18.4.3 that sampling a random tree from $H(\mathbf{f})$ can be done in expected time $\mathcal{O}\left(n^2 \log^2 nq^3\right)$, which makes it reasonable to consider this particular subset of $(\mathbb{F}_2)^n$. We will next show in §18.4.4 that $\textsc{Tree}^{[n \log n]}$ is *nearly injective* on $R(\mathbf{f})$. This makes $\textsc{Tree}^{[n \log n]}$ a nearly-optimal discriminating function, when restricted to $H(\mathbf{f})$. This allows us to show in §18.4.5 that the expected cardinality of $\mathcal{P}$ is $\mathcal{O}(1)$. The proofs of these claims are the *pièce de résistance* of this chapter. Theorem 18.3 (from the introduction) follows from these claims by combining the ingredients together.

18.4.1 Discriminating with Random Trees

We may consider $\textsc{Tree}^{[k]}(\mathbf{f}, \mathbf{x})$ as a random variable when $\mathbf{f}$ is randomly chosen, and we will now try to derive useful information on its distribution. We argue that the trees produced by $\textsc{Tree}^{[k]}$ in fact result from a *branching process*, because the number of children of each node in the tree is random and follows the same fixed distribution (except at the root). This in turn follows from the fact that the number of descendants of a given node is a function of the rank of the differential of $\mathbf{a}$ or $\mathbf{b}$ on the corresponding point of $(\mathbb{F}_2)^n$. We therefore use the standard terminology of branching processes to describe the output of $\textsc{Tree}^{[k]}$. We focus on the usual properties of the random trees produced by $\textsc{Tree}^{[k]}$: what is the *offspring distribution* (number of children of each node)? the *average progeny* (mean number of children per node)?

We unfortunately cannot directly use the distribution of the X_i's to answer these questions, for the following reason: by definition of $\textsc{Node}^{[k]}(\mathbf{f}, \mathbf{x}, \mathbf{y})$, we always have $\mathbf{y} \in \ker \mathrm{D}_{\mathbf{x}}\mathbf{f}$, so that by symmetry of

the differential, $\ker \mathrm{D}_{\mathbf{y}}\mathbf{f}$ always contains $\mathbf{x}$. Except when $\mathbf{x} = 0$ at the root of the tree, this means that $\dim \ker \mathrm{D}_{\mathbf{y}}\mathbf{f}$ is always greater than or equal to two, and thus does not follow the distribution of the X_i's.

Lemma 18.7. *The* offspring distribution *of every non-root node is:*

$$\ell_n(i) = \mathbb{P}\left[\textit{a given particle produces } i \textit{ offspring}\right] = \begin{cases} p_{n,k} & \textit{when } i = q^k - q^2 \\ 0 & \textit{otherwise} \end{cases}$$

where

$$p_{n,k} = \frac{\lambda(n)\lambda(n-2)}{\lambda(k)\lambda(k-2)\lambda(n-k)} \cdot q^{-k(k-2)}$$

The expected progeny *of each node is:*

$$\mu = \sum_{k=2}^{n} \left(q^k - q^2\right) \cdot p_{n,k} = 1 - \frac{1}{q^{n-2}}$$

And the variance of the offspring distribution is:

$$\sigma^2 = \sum_{k=2}^{n} \left(2^k - q^2\right)^2 \cdot p_{n,k} - \mu^2 = q^2(q-1)\left(1 - \frac{q^2+1}{q^n} + \frac{q^2}{q^{2n}}\right)$$

Proof. It is clear that the number of descendants of each node can only be of the form $q^k - q^2$, for some k. This k is precisely the dimension of $\ker \mathrm{D}_{\mathbf{x}}\mathbf{f}$, conditionned on $\mathbf{y} \in \ker \mathrm{D}_{\mathbf{x}}\mathbf{f}$. The expression of $p_{n,k}$ is then a direct consequence of lemma 10.19 (via an implicit use of lemma 17.2). Next, by linearity of the expectation and because the $p_{n,k}$ add up to one, we find that

$$\mu = \sum_{k=2}^{n} q^k \cdot p_{n,k} - q^2.$$

In this expression, the value of the sum is given by lemma 10.20.

The variance can be established in the same way. Expanding its expression yields:

$$\sigma^2 = \sum_{k=2}^{n} q^{2k} \cdot p_{n,k} - 2q^2 \sum_{k=2}^{n} q^k \cdot p_{n,k} + q^4 - \mu^2$$

The second sum is equal to $(\mu + q^2)$, and therefore the previous equation simplifies to:

$$\sum_{k=2}^{n} q^{2k} \cdot p_{n,k} - (q^2 + \mu)^2$$

This is exactly the expression of the variance of the cardinality of the kernel of a random endomorphism conditioned to vanish on two given points, and its value is given by lemma 10.21. □

Before any further considerations, we note that the probability that a node has no children is:

$$\mathbb{P}\left[\text{no offspring}\right] = p_{n,2} = \frac{\lambda(n)}{\lambda(1)} \xrightarrow[n\to\infty]{} 1 - \frac{1}{q^2} + \mathcal{O}\left(\frac{1}{q^3}\right)$$

In particular, this shows that $\textsc{Tree}^{[k]}$ returns the empty tree with high probability, making it a seemingly pretty lousy discriminating function (as previously advertized). We shall see that it is possible to get much more from $\textsc{Tree}^{[k]}$.

Does $\textsc{Tree}^{[k]}$ Produces Random Trees? We will analyze the algorithm under the *heuristic assumption* that the output of $\textsc{Tree}^{[k]}$ is a *(truncated) Galton-Watson* tree with offspring distribution ℓ_n. This is very convenient because these objects are well-known and have been extensively studied.

If the graph were a random graph with degree distribution ℓ_n, then it would be possible to argue that a connected component is a tree asymptotically almost surely, and that $\textsc{Tree}^{[k]}$ does return a random tree with offspring distribution ℓ_n. The problem is that the graph is not random, if only because of the triangles. Removing backwards edges in the BFS takes care of the triangles, but this does not mean that the resulting tree is random, because the graphs are deterministic products of $\mathbf{a}$ and $\mathbf{b}$.

Our (heuristic) assumption that $\textsc{Tree}^{[k]}$ produces Galton-Watson trees with law ℓ_n (*i.e.*, trees where the number of descendants of each node is chosen independently according with law ℓ_n) relies on two arguments.

First, the offspring distribution of trees produced by $\textsc{Tree}^{[k]}$ is actually ℓ_n, and it is the same for each node, as long as no cycles are encountered. Second, Conjecture 18.1 means that it is impossible to detect if the number of descendants of each node in the trees are correlated. The running time of Algorithm 18.4 (yet to be established) is too short to allow a successful detection of the correlation. We shall therefore assume in the sequel that for all useful purposes the output of $\textsc{Tree}^{[k]}$ is a truncated Galton-Watson tree with offspring distribution ℓ_n.

Slightly More Advanced Properties of the Trees. Let us denote by $\mathbb{P}_n$ the law of Galton-Watson trees with offspring distribution ℓ_n. Figure 18.5 shows a random tree sampled according to this distribution (we chose a big one). Because $\mu \leq 1$, the trees are finite with probability one [AN72, chapter I, part A, section 5,theorem 1].

In addition, the probability that a tree sampled according to $\mathbb{P}_n$ has height greater than h is equivalent to $2/(h\sigma^2) \approx 2/(h \cdot q^3)$ [AN72, chapter I, part B,section 9, theorem 1]. A first consequence is that sampling a tree of height $n \log n$ requires sampling $\mathcal{O}\left(q^3 \cdot n \log n\right)$ trees on average. Another consequence is that the expected height of trees sampled according to $\mathbb{P}_n$ is not finite; this justifies why we restrict the height of the result of Tree. It is also known that the expected total number of nodes after h generation is $h+1$ [Pak71], so that computing $\textsc{Tree}^{[k]}$ requires on average $\mathcal{O}(k)$ matrix operations. This means that the complexity of finding a vector $\mathbf{x}$ such that $\textsc{Tree}^{[n \log n]}(\mathbf{f}, \mathbf{x})$ has height at least $n \log n$ is essentially the same as that of performing $\mathcal{O}\left(q^3 \cdot n^2 \log^2 n\right)$ rank computations on $n \times n$ matrices.

The probability that two random Galton-Watson trees are *equal* is easy to determine, however this would be useless here, because Galton-Watson trees are *ordered* trees (*i.e.*, sibling nodes are distinguished, there is a leftmost brother and a rightmost brother for instance). We could not find in the available literature the probability that two Galton-Watson trees are *isomorphic* (*i.e.*, equal if considered as unordered trees, where sibling nodes are indistinguishable). Figure 18.6 shows two trees that are distinct if considered as ordered trees, but isomorphic and thus equal if considered as unordered (and unlabeled) trees. We address this issue using the spine decomposition.

18.4.2 The Spine Decomposition in Detail

For any $n \geq 3$, let $\mathcal{T}_n$ be a tree sampled according to $\mathbb{P}_n$, and let $\mathbb{P}_n^H$ be the law of $\mathcal{T}_n$ conditioned to have height at least $n \log n$ (without loss of generality, we write $n \log n$ instead of $\lfloor n \log n \rfloor$). In fact, $\mathbb{P}_n^H[\cdot]$ stands for $\mathbb{P}_n\big[\cdot \;\big|\; \textsc{Height}(\mathcal{T}_n) \geq n \log n\big]$, and allows to make notations less cumbersome.

We need a criterion to decide whether two conditioned trees are isomorphic or not, and we need it to be simple enough so that we may evaluate the probability that it holds. The criterion we will use is the following: two isomorphic trees with a unique spine decomposition must have empty subtrees emanating from the backbone at the exact same heights. Of course, if the spine decomposition is unique up to height h, then this holds only up to height h. This will intuitively show that two random trees with a unique spine decomposition up to height h are isomorphic with a probability that gets exponentially small in h. We will make this intuition formal later, but we must first introduce some properties of the spine decomposition.

We decompose a conditioned tree (*i.e.*, a tree of law $\mathbb{P}_n^H$) into a *backbone* (or *spine*) going from the root to height $n \log n$, on which we graft a given number of unconditioned Galton-Watson trees at each of its nodes. Looking at all nodes of height $n\sqrt{\log n}$, if only one of them has descendants at height $n \log n$ then the spine up to height $n\sqrt{\log n}$ is uniquely determined: necessarily, it is the path in the tree going from the root to this node (Figure 18.4 illustrates this).

Let us work for a moment with ordered Galton-Watson trees. That is, we also record who is the descendant of each parent and offspring are ordered (so that we can talk about brothers to the left or to the right of an individual). In [Gei99], Geiger shows that if we define the sequence of independent random variables $(V_m^n, Y_m^n), m \in \mathbb{N}$ by

$$\mathbb{P}\big[V_m^n = j, Y_m^n = k\big] = \frac{\mathbb{P}_n\left[\textsc{Height}(\mathcal{T}_n) \geq m-1\right]}{\mathbb{P}_n\left[\textsc{Height}(\mathcal{T}_n) \geq m\right]} \cdot \mathbb{P}_n\left[\textsc{Height}(\mathcal{T}_n) < m-1\right]^{j-1} \cdot \ell_n(k), \qquad 1 \leq j \leq k < \infty, \quad (18.3)$$

then $\mathcal{T}_n$ conditioned to have height at least h has the same law as the random tree constructed inductively as follows:

- The root (i.e., the first node of the spine) has Y_h^n offspring.
- To each of the $V_h^n - 1$ first offspring node we graft a Galton-Watson tree with offspring distribution ℓ_n and conditioned to have height (strictly) less than $h-1$. These $V_h^n - 1$ trees are independent of each other (and of the rest of the construction). These subtrees are on the left of the backbone on Figure 18.7.

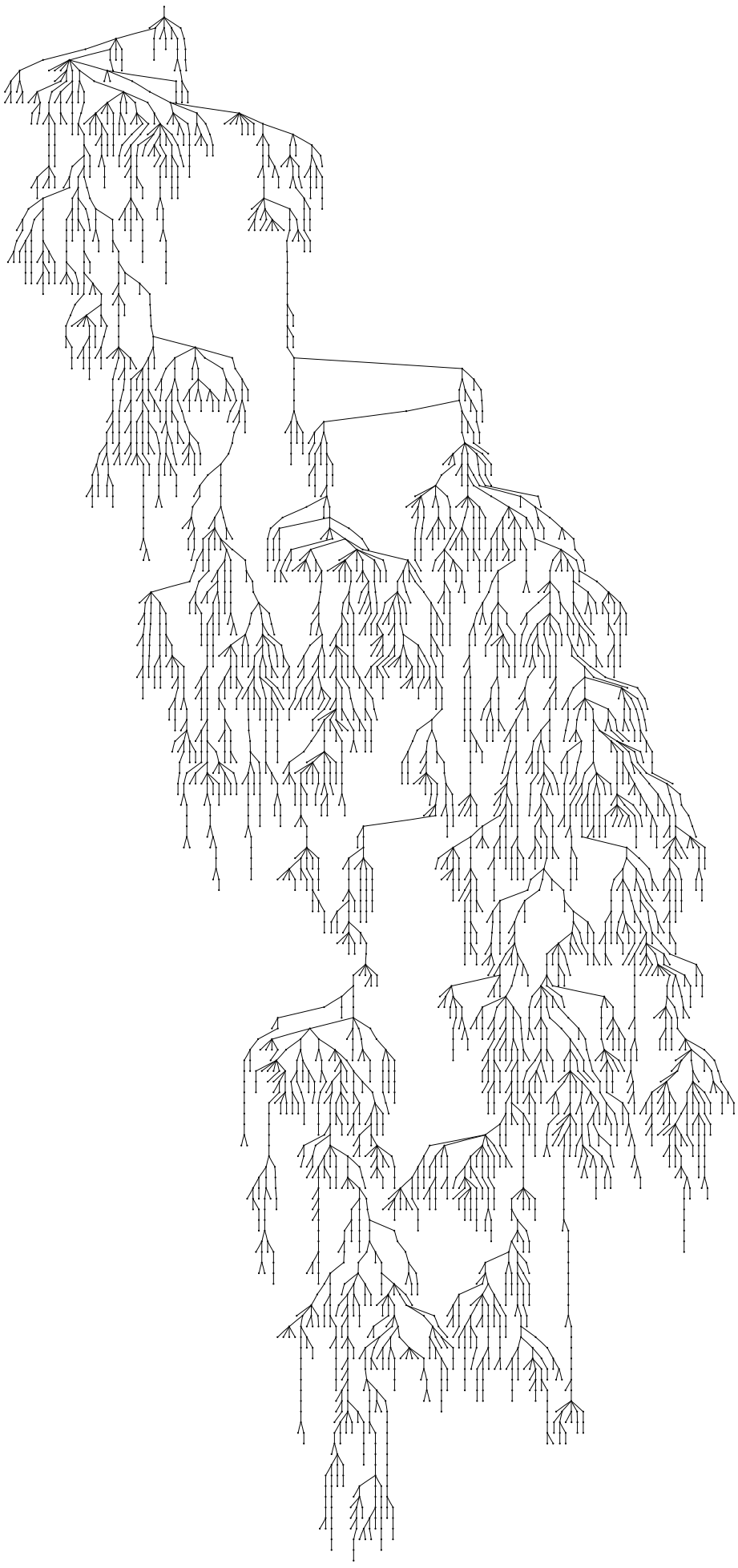


Figure 18.5: A random Galton-Watson tree with distribution $\mathbb{P}$ (the leaves are not drawn).

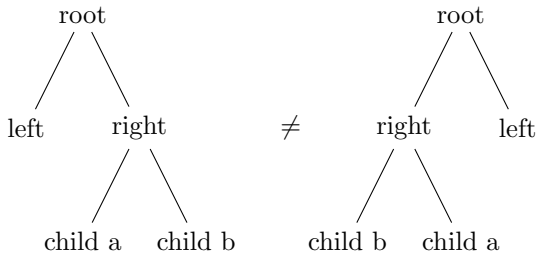


Figure 18.6: Two distinct, but isomorphic trees. Both are represented as unordered trees by $\{\!\{\emptyset, \{\!\{\emptyset, \emptyset\}\!\}\}\!\}$.

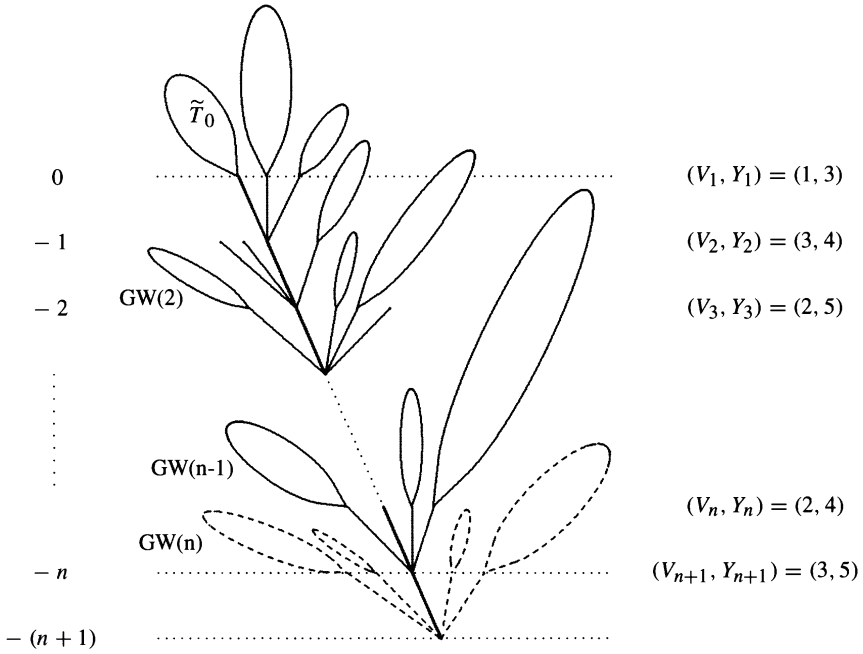


Figure 18.7: Illustration of the spine decomposition (this is Figure 1 from [Gei99]). This shows the Galton-Watson tree conditionned on non-extinction at generation n and $n+1$ respectively. $GW(k)$ denotes a Galton-Watson tree conditioned to be extinct at generation k. The subtrees to the right of the line of descent of the left-most particle are ordinary Galton-Watson trees.

- To each of the $Y_h^n - V_h^n$ last offspring, we graft an unconditioned Galton-Watson tree with offspring distribution ℓ_n (again, these trees are independent of each other and of the rest of the construction). These subtrees are on the right of the backbone on Figure 18.7.
- The V_h^n-th offspring node continues the spine. It has Y_{h-1}^n offspring, the first V_{h-1}^n ones are the roots of i.i.d. Galton-Watson trees conditioned to have height less than $h-2$, the last $Y_{h-1}^n - V_{h-1}^n$ are the roots of i.i.d. unconditioned Galton-Watson trees and the spine carries on with the V_{h-1}^n-th offspring, which has Y_{h-2}^n offspring nodes, and so on.

Observe that the marginal distribution of Y_m^n is given by

$$\mathbb{P}\left[Y_m^n = y\right] = \frac{1 - \mathbb{P}_n\left[\textsc{Height}(\mathcal{T}_n) < m-1\right]^y}{\mathbb{P}_n\left[\textsc{Height}(\mathcal{T}_n) \geq m\right]} \cdot \ell_n(y), \tag{18.4}$$

which is 0 if $y = 0$ or if y is not of the form $q^k - q^2$ for some $k \in \{3, \ldots, n\}$. Hence, the spine can be seen as a "prolific" line of descent that survives up to generation h by producing a biased number of offspring, while the other individuals of the population reproduce essentially according to the initial offspring distribution (we refer to [Gei99] for an explanation of the fact that trees emanating from brothers to the left of the spine are conditioned not to have descendants at generation h).

18.4.3 Sampling in $H(\mathbf{f})$ Takes Polynomial Time

We are ready to show that our random trees, when conditioned to have height $n \log n$, have a spine decomposition up to height $n\sqrt{\log n}$ with very high probability.

Theorem 18.8. *There exists a constant C such that the probability that a random tree sampled according to $\mathbb{P}_n^H$ has a spine decomposition up to height $n\sqrt{\log n}$ is greater than $1 - C/\sqrt{\log n}$.*

Before proving this theorem, we review its implications. Informally speaking, it means that when $\textsc{Tree}^{[n \log n]}$ produces a tree of height $n \log n$, then this tree has a unique spine decomposition up to height $n \log \sqrt{n}$ asymptotically almost surely. In the auxiliary function TreeList, only a negligible fraction of the trees will be rejected because it does not meet the unique spine decomposition condition. Therefore, the number of iterations of the inner loop of TreeList is $\mathcal{O}\left(n^2 \cdot q^2\right)$, which is the number required to sample a sufficiently high tree. In addition, this shows that, by linearity of the expectation, that:

$$\frac{2q^n}{\sigma^2 \cdot n \log n}\left(1 - \frac{C}{n\sqrt{\log n}}\right) \leq \mathbb{E}\left[|H(\mathbf{f})|\right] \leq \frac{2q^n}{\sigma^2 \cdot n \log n}. \tag{18.5}$$

proof of theorem 18.8. We show that under $\mathbb{P}_n^H$, with high probability only one path from the root to height $n\sqrt{\log n}$ extends to a path reaching height $n \log n$. Call this event A_n. Since this property is purely topological, then it does not matter whether the tree is ordered or not. We obtain the desired result by bounding from below the probability of A_n by the probability that all trees emanating from the spine under height $n\sqrt{\log n}$ are of height less than $n(\log n - \sqrt{\log n})$. The independence of this family of trees, together with the fact (easy to check) that for every integer i in the interval $\{1, \ldots, n\sqrt{\log n} - 1\}$

$$\mathbb{P}_n\left[\textsc{Height}(\mathcal{T}_n) < n(\log n - \sqrt{\log n}) \;\middle|\; \textsc{Height}(\mathcal{T}_n) < n \log n - i\right] \geq \mathbb{P}_n\left[\textsc{Height}(\mathcal{T}_n) < n(\log n - \sqrt{\log n})\right],$$

enables us to write

$$\begin{aligned}
\mathbb{P}_n^H[A_n] &\geq \prod_{i=0}^{n\sqrt{\log n}-1} \mathbb{E}\left[\mathbb{P}_n\left[\textsc{Height}(\mathcal{T}_n) < n(\log n - \sqrt{\log n})\right]^{Y^n_{n \log n - i} - 1}\right] \\
&\geq \mathbb{E}\left[\mathbb{P}_n\left[\textsc{Height}(\mathcal{T}_n) < n(\log n - \sqrt{\log n})\right]^{\sum_{i=0}^{n\sqrt{\log n}-1} Y^n_{n \log n - i}}\right].
\end{aligned} \tag{18.6}$$

Now, as $n \to +\infty$, all the $p_{n,k}$ (for $k \in \{3, \ldots, n\}$) converge to a finite limit $p_{\infty,k}$, and we also have $\mu_n \to 1$ (recall that $\mu_n < 1$ for every n) and $\sigma_n^2 \to q^3 - q^2 =: \sigma^2$. The last two convergences happen exponentially fast in n, therefore the same proof as that of Theorem 3.1 in [Gei99] (in which $\mu_n = 1$ for all n) shows that whenever $(m_n)_{n \geq 1}$ tends to infinity at most polynomially, we have

$$\lim_{n \to \infty} m_n \cdot \mathbb{P}_n\left[\textsc{Height}(\mathcal{T}_n) \geq m_n\right] = \frac{2}{\sigma^2}. \tag{18.7}$$

Furthermore, we have the following lemma.

Lemma 18.9. *There exist $C_3, C_4 > 0$ such that for every $n \geq 3$,*

$$\mathbb{P}\left[\sum_{i=0}^{n\sqrt{\log n}-1} Y^n_{n \log n - i} > C_3 \cdot n\sqrt{\log n}\right] \leq \frac{C_4}{n\sqrt{\log n}}.$$

We postpone the proof of Lemma 18.9 until the end of the proof of Theorem 18.8. Armed with (18.7) and Lemma 18.9, we can come back to (18.6) and write for every n

$$\begin{aligned}
\mathbb{P}_n^H[A_n] &\geq \mathbb{E}\left[\mathbb{P}_n\left[\textsc{Height}(\mathcal{T}_n) < n(\log n - \sqrt{\log n})\right]^{C_3 n\sqrt{\log n}} \mathbf{1}_{\left\{\sum_{i=0}^{n\sqrt{\log n}-1} Y^n_{n \log n - i} \leq C_3 n\sqrt{\log n}\right\}}\right] \\
&\geq \left(1 - \frac{C_5}{\sigma^2 n \log n}\right)^{C_3 n\sqrt{\log n}} \times \mathbb{P}\left[\sum_{i=0}^{n\sqrt{\log n}-1} Y^n_{n \log n - i} \leq C_3 \cdot n\sqrt{\log n}\right] \\
&\geq e^{-C_6/\sqrt{\log n}}\left(1 - \frac{C_4}{n\sqrt{\log n}}\right) \geq 1 - \frac{C_7}{\sqrt{\log n}}.
\end{aligned} \tag{18.8}$$

Note that for the third inequality, use the fact that $1 - x \geq e^{-2x}$ for every $x \in [0, 1/2]$. What (18.8) shows is that for every $n \geq 3$, if we sample a Galton-Watson tree $\mathcal{T}_n$ according to $\mathbb{P}_n^H$, then with probability at least $1 - C_7/\sqrt{\log n}$ there will be a unique spine decomposition under height $n\sqrt{\log n}$. □

proof of lemma 18.9. We use Markov's inequality (in a Chebychev-like fashion) as follows: if $C_3 > 0$, we have for each $n \geq 3$

$$\begin{aligned}
&\mathbb{P}\left[\sum_{i=0}^{n\sqrt{\log n}-1} Y^n_{n\log n-i} > C_3 n\sqrt{\log n}\right] \\
&= \mathbb{P}\left[\sum_{i=0}^{n\sqrt{\log n}-1} \left(Y^n_{n\log n-i} - \mathbb{E}\left[Y^n_{n\log n-i}\right]\right) > C_3\cdot n\sqrt{\log n} - \sum_{i=0}^{n\sqrt{\log n}-1} \mathbb{E}\left[Y^n_{n\log n-i}\right]\right] \\
&\leq \frac{\mathbb{E}\left[\left(\sum_{i=0}^{n\sqrt{\log n}-1} \left(Y^n_{n\log n-i} - \mathbb{E}\left[Y^n_{n\log n-i}\right]\right)\right)^2\right]}{\left(C_3\cdot n\sqrt{\log n} - \sum_{i=0}^{n\sqrt{\log n}-1} \mathbb{E}\left[Y^n_{n\log n-i}\right]\right)^2}.
\end{aligned} \tag{18.9}$$

Let us show that the numerator in the right-hand side of (18.9) is of order $n\sqrt{\log n}$, while the denominator is of order $n^2\log n$ whenever $C_3 > 0$ is large enough. These two points rely on appropriate bounds on the first two moments of all $Y^n_{n\log n-i}$'s (observe that the numerator is in fact the sum of the variances of the $Y^n_{n\log n-i}$'s). Indeed, recall from (18.4) that for every $k \in \{3,\dots,n\}$,

$$\mathbb{P}\left[Y^n_{n\log n-i} = q^k - q^2\right] = \frac{1-\mathbb{P}_n\left[\textsc{Height}(\mathcal{T}_n) < n\log n - i - 1\right]^{q^k-q^2}}{\mathbb{P}_n\left[\textsc{Height}(\mathcal{T}_n) \geq n\log n - i\right]}\cdot p_{n,k}$$

and these are the only possible values for $Y^n_{n\log n-i}$. Because $1-e^{-x} \leq x$ for all $x \geq 0$, we can write for every $i \leq n\sqrt{\log n} - 1$:

$$\begin{aligned}
1-\mathbb{P}_n\left[\textsc{Height}(\mathcal{T}_n) < n\log n - i - 1\right]^{q^k-q^2} &\leq -\left(q^k-q^2\right)\log \mathbb{P}_n\left[\textsc{Height}(\mathcal{T}_n) < n\log n - i - 1\right] \\
&\leq -\left(q^k-q^2\right)\log \mathbb{P}_n\left[\textsc{Height}(\mathcal{T}_n) < n\log n - n\sqrt{\log n} - 2\right].
\end{aligned}$$

We thus have for every such integer i

$$\frac{1-\mathbb{P}_n\left[\textsc{Height}(\mathcal{T}_n) < n\log n - i - 1\right]^{q^k-q^2}}{(q^k-q^2)\cdot\mathbb{P}_n\left[\textsc{Height}(\mathcal{T}_n) \geq n\log n - i\right]} \leq -\frac{\log \mathbb{P}_n\left[\textsc{Height}(\mathcal{T}_n) < n\log n - n\sqrt{\log n} - 2\right]}{\mathbb{P}_n\left[\textsc{Height}(\mathcal{T}_n) \geq n\log n\right]}.$$

Moreover, because $\lambda(\cdot)$ is decreasing,

$$\frac{\lambda(n)\lambda(n-2)}{\lambda(k)\lambda(k-2)\lambda(n-k)} \leq \lim_{n\to\infty}\frac{1}{\lambda(n)} =: C_q.$$

Combining the above, we arrive at

$$\mathbb{P}\left[Y^n_{n\log n-i} = q^k - q^2\right] \leq -\frac{\log \mathbb{P}_n\left[\textsc{Height}(\mathcal{T}_n) < n\log n - n\sqrt{\log n} - 2\right]}{\mathbb{P}_n\left[\textsc{Height}(\mathcal{T}_n) \geq n\log n\right]}\cdot C_q\cdot\left(q^k-q^2\right)q^{-k(k-2)}$$

for every $n \geq 3$ and $k \in \{2,\dots,n\}$. This yields

$$\begin{aligned}
\mathbb{E}\left[Y^n_{n\log n-i}\right] &\leq -\frac{\log \mathbb{P}_n\left[\textsc{Height}(\mathcal{T}_n) < n\log n - n\sqrt{\log n} - 2\right]}{\mathbb{P}_n\left[\textsc{Height}(\mathcal{T}_n) \geq n\log n\right]}\cdot C_q\cdot\sum_{k=3}^{n}\left(q^k-q^2\right)^2 q^{-k(k-2)} \\
\mathbb{E}\left[\left(Y^n_{n\log n-i}\right)^2\right] &\leq -\frac{\log \mathbb{P}_n\left[\textsc{Height}(\mathcal{T}_n) < n\log n - n\sqrt{\log n} - 2\right]}{\mathbb{P}_n\left[\textsc{Height}(\mathcal{T}_n) \geq n\log n\right]}\cdot C_q\cdot\sum_{k=2}^{n}\left(q^k-q^2\right)^3 q^{-k(k-2)}
\end{aligned}$$

Now, by (18.7) we have

$$\lim_{n\to\infty} -\frac{\log \mathbb{P}_n\left[\textsc{Height}(\mathcal{T}_n) < n\log n - n\sqrt{\log n} - 2\right]}{\mathbb{P}_n\left[\textsc{Height}(\mathcal{T}_n) \geq n\log n\right]} = 1,$$

and furthermore,

$$\sum_{k=3}^{\infty}\left(q^k-q^2\right)^2 q^{-k(k-2)} =: m_1 < \infty \qquad \text{and} \qquad \sum_{k=3}^{\infty}\left(q^k-q^2\right)^3 q^{-k(k-2)} =: m_2 < \infty.$$

As a consequence, there exists $C > 0$ such that for every $n \geq 3$, we have

$$\sum_{i=0}^{n\sqrt{\log n}-1} \mathbb{E}\left[Y^n_{n\log n - i}\right] \leq C m_1 n\sqrt{\log n},$$

and (using the independence of all Y^n_m's)

$$\mathbb{E}\left[\left(\sum_{i=0}^{n\sqrt{\log n}-1} Y^n_{n\log n-i} - \mathbb{E}\left[Y^n_{n\log n-i}\right]\right)^2\right] = \sum_{i=0}^{n\sqrt{\log n}-1} \mathrm{Var}\left(Y^n_{n\log n-i}\right) \leq C' n\sqrt{\log n},$$

for a constant $C' > 0$ depending on m_1 and m_2. Choosing $C_3 > C m_1$ and coming back to (18.9), we obtain the existence of $C_4 > 0$ such that for every $n \geq 3$,

$$\mathbb{P}\left[\sum_{i=0}^{n\sqrt{\log n}-1} Y^n_{n\log n-i} \geq C_3 n\sqrt{\log n}\right] \leq C_4 \frac{n\sqrt{\log n}}{n^2 \log n} = \frac{C_4}{n\sqrt{\log n}}.$$

This completes the proof of Lemma 18.9. □

18.4.4 Near-injectivity of $\textsc{Tree}^{[n\log n]}$ over $H(\mathbf{f})$.

Our key argument to establish the properties of Algorithm 18.4 is that $\textsc{Tree}^{[n\log n]}$ is *close to injective* over $H(\mathbf{f})$.

Theorem 18.10. *There is a constant $\kappa \in]0;1[$ such that if two trees sampled according to $\mathbb{P}^H_n$ have a unique spine decomposition up to height $n\sqrt{\log n}$, then the probability that they are isomorphic is upper-bounded by $\kappa^{n\sqrt{\log n}}$.*

Over $H(\mathbf{f})$, our discriminating function $\textsc{Tree}^{[n\log n]}$ produces random trees meeting the conditions of the theorem, which then tells us that the result of two distinct invocations of $\textsc{Tree}^{[n\log n]}$ are non-isomorphic with overwhelming probability.

proof of theorem 18.10. Let us use again $\mathcal{T}_n$ (from §18.4.2) and its spine decomposition under the additional conditioning that all trees emanating from the spine under height $n\sqrt{\log n}$ are of height smaller than $n(\log n - \sqrt{\log n})$. We write $\tilde{\mathbb{P}}^H_n$ for the law of this tree. By construction, each brother of the i-th node of the spine ($0 \leq i \leq n\sqrt{\log n} - 1$) has no offspring with probability

$$\begin{aligned} e_n &:= \mathbb{P}_n\left[\mathcal{T}_n = \emptyset \;\middle|\; \textsc{Height}(\mathcal{T}_n) \leq n(\log n - \sqrt{\log n})\right] \\ &= \frac{\mathbb{P}_n\left[\mathcal{T}_n = \emptyset\right]}{\mathbb{P}_n\left[\textsc{Height}(\mathcal{T}_n) \leq n(\log n - \sqrt{\log n})\right]} \\ &= \frac{\ell_n(0)}{\mathbb{P}_n\left[\textsc{Height}(\mathcal{T}_n) \leq n(\log n - \sqrt{\log n})\right]}. \end{aligned} \tag{18.10}$$

Brother to the right or to the left does not matter here since the condition at the denominator is stronger than $\textsc{Height}(\mathcal{T}) < n\log n - i - 1$ for our range of integers i. Let us use (18.10) to obtain some bounds (away from 0 and 1), uniform in n and $i \leq n\sqrt{\log n} - 1$, for the probability that all of the $Y^n_{n\log n - i} - 1$ brothers of the i-th node of the spine have zero offspring. Because $\ell_n(0) = p_{n,2}$ and using (18.7), the right-hand side of (18.10) is equivalent as $n \to \infty$ to

$$\frac{\ell_n(0)}{1 - 2/(\sigma^2 \cdot n\log n)} \underset{n\to\infty}{\simeq} \lim_{n\to\infty} \frac{\lambda(n)}{\lambda(2)} =: \mathbf{e} \in]0,1[. \tag{18.11}$$

Thus, if we denote $\alpha = \tilde{\mathbb{P}}^H_n$ [no nephews at height i], then by definition

$$\alpha \geq \mathbb{P}\left[Y^n_{n\log n - i} = q^3 - q^2\right] \cdot (e_n)^{q^3 - q^2 - 1}$$

Using (18.4) and (18.7),

$$\alpha \geq \frac{1 - \left(1 - \dfrac{2}{\sigma^2(n\log n - i - 1)} + o\left(\frac{1}{n\log n}\right)\right)^{q^3-q^2}}{\dfrac{2}{\sigma^2(n\log n - i)} + o\left(\frac{1}{n\log n}\right)} \cdot p_{n,2} \cdot (e_n)^{q^3-q^2-1}$$

The fraction is equal to $q^3-q^2+o(1/(n\log n))$, and given the expression of $p_{n,3}$ as well as (18.11), the lower bound on α is equivalent to

$$\frac{q^3-q^2}{q^3}\cdot\frac{\mathbf{e}^{q^3-q^2-1}}{\lambda(1)\lambda(3)}\cdot\prod_{j=1}^{\infty}\left(1-\frac{1}{q^j}\right)=\mathbf{e}^{q^3-q^2-1}\prod_{j=4}^{\infty}\left(1-\frac{1}{q^j}\right)\ \in]0,1[.$$

Likewise,

$$\begin{aligned}\tilde{\mathbb{P}}_n^H[\text{at least one nephew at height } i] &\geq \mathbb{P}\left[Y^n_{n\log n-i}=q^3-q^2\right]\ \left(1-(e_n)^{q^3-q^2-1}\right)\\ &\simeq (1-\mathbf{e}^{q^3-q^2-1})\prod_{j=4}^{\infty}\left(1-\frac{1}{q^j}\right)\ \in]0,1[.\end{aligned}$$

Hence, since these two probabilities belong to $]0,1[$ for all $n\geq 3$ and $i\leq n\sqrt{\log n}-1$, and belong to a smaller interval of $]0,1[$ bounded away from 0 and 1 whenever n is large enough, this provides the existence of $\kappa_l,\kappa_u\in]0,1[$ such that for every $n\geq 3$ and $i\in\{0,\dots,n\sqrt{\log n}-1\}$,

$$1-\kappa_l\leq\tilde{\mathbb{P}}_n^H[\text{no nephews at height } i]\leq\kappa_u. \tag{18.12}$$

Now, let $\mathcal{T}_n,T'_n$ be two trees of height at least $n\log n$ and such that their spine decompositions are unique under height $n\sqrt{\log n}$. For every $i\in\{0,n\sqrt{\log n}-1\}$, let γ_i^n (resp. γ'^n_i) be the indicator function of the event that all brothers of the i-th node of the spine have no offspring. It follows from the properties of the spine decomposition that for every $n\geq 3$, $\{\gamma_i^n,\ 0\leq i\leq n\sqrt{\log n}-1\}$ form a family of independent random variables and by (18.12), we have

$$\tilde{\mathbb{P}}_n^H\left[\gamma_i^n=1\right]\leq\kappa_u\qquad\text{and}\qquad\tilde{\mathbb{P}}_n^H\left[\gamma_i^n=0\right]\leq\kappa_l.$$

Comparing the absence or presence of nephews of the spine in $\mathcal{T}_n$ and in $\mathcal{T}'_n$, and defining the constant $\kappa=\max(\kappa_l,\kappa_u)<1$, we obtain:

$$\tilde{\mathbb{P}}_n^H[\mathcal{T}=\mathcal{T}']\leq\kappa^{n\sqrt{\log n}}.$$

□

18.4.5 Back to Algorithm 18.4: Proof of Theorem 18.3

Armed with theorems 18.8 and 18.10, we are ready to show that Algorithm 18.4 lives up to the expectations we expressed in theorem 18.3. Note that thanks to lemma 3.2, there will be a right pair in $U\times V$ with probability $1-1/e$, as claimed. We must still investigate the complexity of building U and V, and the number of pairs in $\mathcal{P}$ that will have to be tested.

Corollary 18.11. *Let $\ell=\sqrt{2}/(\sqrt{n\log n})\cdot q^{n/2-1}$, $(\mathbf{x}_1,\dots,\mathbf{x}_\ell)$ and $(\mathbf{y}_1,\dots,\mathbf{y}_\ell)$ be two collections of distinct vectors in $H(\mathbf{f})$. Define $U_i=\textsc{Tree}^{[n\log n]}(\mathbf{a},\mathbf{x}_i)$ and $V_i=\textsc{Tree}^{[n\log n]}(\mathbf{b},\mathbf{y}_i)$.*

All the U_i's (resp. V_i's) are pairwise non-isomorphic with otherwheling probability. In addition, the U_i's are pairwise non-isomorphic from the V_i's with overwhelming probability.

Proof. Observe that the U_i's and V_i's are unlabeled and unordered trees sampled according to $\mathbb{P}_n^H$ and with spine decomposition up to height $n\sqrt{\log n}$. The number of collisions between the U_i's and the V_i's is

$$\mathcal{N}=\sum_{I\leq i<j\leq\ell}\mathbf{1}_{U_i=V_j}.$$

If all the U_i's are different [1] —it is a worst-case assumption—, then applying Markov's inequality to $\mathcal{N}$ and using theorem 18.10 yields:

$$\mathbb{P}\left[\mathcal{N}\geq 1\right]\leq\sum_{1\leq i<j\leq\ell}\mathbb{P}\left[U_i=V_j\right]\leq\ell^2\cdot\kappa^{n\sqrt{\log n}}\leq q^n\cdot\kappa^{n\sqrt{\log n}}\xrightarrow{n\to+\infty}0$$

Since $\kappa<1$, then $q^n\kappa^{n\sqrt{\log n}}$ converges to zero faster than the inverse of any polynomial, regardless of the value of q. The exact same reasoning shows that the U_i (resp. V_i) are pairwise non-isomorphic with the same probability. □

Corollary 18.11 is all we need to conclude the study of Algoritm 18.4. It shows that in TreeList all the trees selected by the inner loop will be pairwise non-isomorphic, so that the total number of iterations of the outer loop is $\mathcal{O}(\ell)$. It also shows that with overwhelming probability only the right pairs will remain in $\mathcal{P}$. As a last remark, note that $\mathcal{P}$ can be formed in time $\mathcal{O}\left(q^{n/2}\right)$ by various methods (hash-table, sorting, ...). This completes the proof of theorem 18.3, under the assumption that $\textsc{Tree}^{[n\log n]}$ produces independent random trees.

1. *i.e.*, non-isomorphic

n	q	generating U and V	computing $\mathcal{P}$	$\lvert U\rvert$	$\lvert\mathcal{P}\rvert$
16	2	3.6 s	1s	64	6
24	2	123 s	13s	836	5
32	2	61 min	200s	11585	2
40	2	31 h	2h	165794	7

Table 18.2: Experimental results on Branching-QMLE

18.4.6 Practical Results.

We have implemented Algorithm 18.4 using MAGMA, and we found out that it works well in practice, as Table 18.2 shows. The experiment clearly shows that $|\mathcal{P}|$ is constant. This justify our heuristic assumption *a posteriori*. The implementation is also in the public domain and is available at:

`http://www.di.ens.fr/~bouillaguet/implem/branching-qmle.magma`

Possible improvement. Our current implementation stores the whole trees. It would be possible to save some memory by hashing the trees on-the-fly. For instance, to hash the trees to 128-bit digests (suitable with q^n up to 2^{128}), pick up two 128-bit random primes p_1 and p_2, and let:

$$\begin{aligned}
\textsc{Tree}^{[k]}(\mathbf{f},\mathbf{x}) &= \textsc{Node}^{[k]}(\mathbf{f},0,\mathbf{x}) \\
\textsc{Node}^{[k]}(\mathbf{f},\mathbf{x},\mathbf{y}) &= \left(p_2 + \sum_i {p_1}^i \cdot \textsc{Node}^{[k-1]}(f,\mathbf{y},\mathbf{z}_i)\right) \bmod 2^{128} \qquad \mathbf{z} = \ker \mathrm{D}_{\mathbf{y}}\mathbf{f} - \langle \mathbf{x},\mathbf{y}\rangle \\
\textsc{Node}^{[0]}(\mathbf{f},\mathbf{x},\mathbf{y}) &= p_2
\end{aligned}$$

18.4.7 Getting Rid of the $q = 2$ Limitation

So far, our best algorithm is restricted to the case where $q = 2$. One of the problems when $q > 2$ is that the vector space structure of the kernels creates cycles of length 4 in the graph. For instance, with $q = 3$, when there is an edge $x \leftrightarrow y$, there is also an edge $x \leftrightarrow 2y$, and there are also two edges $y \leftrightarrow 2x$ and $2y \leftrightarrow 2x$. This complicates (if it does not simply prevents) using the same methodology to encode the topological structure of neighborhoods into trees. All hope is not lost though, because many decompositions of graphs into tree may be used for this purpose. For instance, the *modular decomposition* introduced by Gallai in 1967 comes to mind, as it would likely pack together collinear vector in a single "module" of the graph. In addition, it can be computed in linear time [TCHP08].

Chapter 19

A Class of Weak Keys in HFE

In this chapter we present a new cryptographic application of QMLE algorithms. A class of weak keys for HFE is identified, the security of which reduces to the hardness of QMLE. Ironically, the use of such weak keys was suggested as a way to reduce key sizes. In that case the attack described in this chapter recovers the secret key in practice. This joint work with Antoine Joux and Joana Treger-Marim resulted in a publication in the Journal of Mathematical Cryptology.

The HFE cryptosystem has been proposed in 1996 by Patarin in [Pat96b] in order to avoid his own attack on the C* cryptosystem [MI88, Pat95]. The latter basically hides the power function $X \mapsto X^{1+q^\theta}$ in an extension field of degree n over $\mathbb{F}_q$, using two secret linear bijections S and T. In order to invert it, it suffices to remark that this power function, as the RSA power function, can be easily inverted provided that $\left(1+q^\theta\right)$ is invertible modulo (q^n-1). In [Pat96b], Patarin proposed to change the internal *known* monomial into a *secret* polynomial $\mathbf{f}$ of small degree. The legitimate user can still easily invert the public key since she knows S and T, and can invert the small degree polynomial using the Berlekamp algorithm for instance.

Related Work. From the adversary point of view, the action of S and T transforms the secret internal polynomial into a very sparse univariate polynomial of very high degree, as shown for instance by Kipnis and Shamir in [KS99].

A possible decryption attack would consist in inverting or factorizing this polynomial. However, there are no efficient algorithms to perform these tasks (an attempt [1] can be found in [SG03]), and merely deciding the existence of roots is in fact NP-complete [KS99].

HFE belongs to the category of public-key cryptosystems based on the hardness of computing a *functional decomposition*: given the composition of two functions f and g, can one identify the two components? Other examples include C^*, SFLASH [PCG01a], FAPKC [TC86], 2R [PG97] and McEliece [McE78]. With the exception of the latter, the former have all been broken because computing a functional decomposition was not as hard as expected. In the context of HFE, computing such a decomposition is related to decomposing the univariate representation of the public key, in order to recover the secret internal polynomial $\mathbf{f}$ as well as polynomial representations of S and T. Computing polynomial decompositions is a simple and natural mathematical problem which has a long history, going back to the works of Ritt and Ore in 1922 and 1930 respectively [Rit22, Ore34]. Today, polynomial decomposition algorithms exist for some classes of polynomials over finite fields [vzG90a, vzG90b], but no such algorithm is applicable to HFE. One step of the attack presented in this article amounts to computing a polynomial decomposition, and makes use of Gröbner bases.

The complexity of existing attacks gainst HFE, which all amount to solving systems of quadratic equations, depends on the degree d of the secret internal polynomial. When this degree is fixed, their complexity is polynomial in the security parameter n, although the exponent can be ridiculously large. In order for decryption to be polynomial, d must grow at most polynomially in n, and in that case the attacks are no longer polynomial. We consider this setting to be the most natural one and we will then suppose in the remaining of the chapter that d is polynomial in n.

A simple decryption attack against HFE consists, given a ciphertext, in trying to solve the equations given by the public key. In 2003, Faugère and Joux experimentally showed that the HFE equations are not random systems of multivariate equations, because computing a Gröbner basis for these equations is much easier than the corresponding problem with random quadratic equations [FJ03]. This allowed a custom implementation of the $\mathbf{F_5}$ algorithm [Fau02] to break the first HFE challenge, for which the public key has 80 quadratic equations in 80 unknowns over $\mathbb{F}_2$. Later, Granboulan *et al.* [GJS06] showed that specific algebraic properties of the HFE equations make the complexity of inverting HFE subexponential, in $\mathcal{O}\left(\exp\left(\log^2 n\right)\right)$ when $q=2$.

1. unsuccessful, as far as we understand

In general, the hardness of recovering the secret key of HFE from the public key is unrelated to PLE problems, unless the internal polynomial is made public. A key recovery attack in the usual case where this polynomial is secret was presented in [KS99] and turns the problem of recovering T into an instance of the MinRank problem, the decisional version of which is NP-Complete [BFS99]. Solving this instance of MinRank can be done by solving an overdetermined system of about n^2 quadratic equations in about $(n \cdot \log d)$ variables. The complexity of solving these equations is subexponential in $\mathcal{O}\left(\exp\left(\log^3 n\right)\right)$. This is too high to be practical, even for parameters corresponding to the HFE challenge that was broken.

These results show that HFE is not as robust as expected. However, can we consider that HFE is really completely broken? Is it still a viable alternative to RSA?

The cryptographic community often perceives HFE as broken, because of the practical attacks on some instances, and vastly lost both trust and interest in it. We would like to argue that the situation of HFE is slightly more complex. The complexity of some Gröbner basis algorithms, such as $\mathbf{F_5}$, is better understood [BFSY05] and allows to estimate the complexity of the decryption attacks, which remains relatively high for general instances. Moreover, standard modification—such as removing some equations from the public key—destroy the algebraic structure exposed by the public key and that was exploited by Gröbner basis algorithms. HFE with removed public equations is often called HFE$^-$, and seems suitable as a signature scheme. No attack against HFE$^-$ faster than exhaustive search have been found (yet). In particular, the second HFE cryptanalytic challenge, with removed public equations, is currently far from being broken. Furthermore, Dubois and Gama have studied the degree of regularity of various classes of HFE instances in [DG10b]. While they only provide an upper-bound on the complexity of computing a Gröbner basis of the public key, their result show that there are wide ranges of parameters that are not provably broken by the direct Gröbner attack. If their upper-bound were tight (this is a small leap of faith), then many interesting ranges of HFE parameters would be secure.

All in all, HFE is comparatively in better shape than the SFLASH signature scheme for which polynomial time algorithms are known both to invert [DFS07, DFSS07] and to recover equivalent private keys [FMRS08]. SFLASH is based on C^*, hence has a single internal monomial. The attacks against SFLASH exploit the fact that multiplication matrices commute in some way with this internal monomial. Using this property, it is possible to recover conjugates of the multiplications by the secret matrix S using simple linear algebra on the differential of the public key [FMRS08]. However, for general HFE, the multiplications no longer commute with the secret polynomial. Another issue is that we also need to recover the internal secret polynomial.

In this chapter, we consider the key recovery problem on a class of *weak keys* for HFE. As opposed to the decryption attack of Faugère and Joux [FJ03], we recover an equivalent representation of the secret key that subsequently allows to inverse the trapdoor with the same complexity as the legitimate user. The weak instances we attack have an internal polynomial with coefficients in the ground field and not in the extension field as it was originally specified, or instances that are reducible to these specific ones (by considering equivalent transformations S and T, see §19.2). Some instances belonging to this category were proposed by Patarin himself in [Pat96c] (an extended version of [Pat96b]) with the aim of reducing the size of the HFE public key (the so-called "subfield" variant). However, notice that the family of weak keys described here does not reduce to this subfield variant, and choosing the coefficients of the secret polynomial in the base field could be used to reduce the secret-key size. While in general, the hardness of the key-recovery does not depend on the hardness of QMLE, we show that on weak keys, the key-recovery problem can be reduced to an instance of the QMLE problem, and that the solutions of this problem allow us to efficiently recover all the secret elements (or equivalent data). Our QMLE algorithms allow to solve the instances in practice for realistic parameter sets.

Coming back to the subfield variant, other schemes, including UOV [KPG99] for instance, also have subfield variants, and the default in the design of an older version of SFLASH (v1) was to choose the secrets in a subfield. These schemes, or their subfield variants have all been broken: SFLASH v1 was attacked by Gilbert and Minier in [GM02], and subfield-UOV was shown to be insecure as well [BWP05]. Although SFLASH and HFE share a similar structure, the Gilbert-Minier attack against SFLASH v1 cannot be applied to subfield-HFE, since it is based on Patarin's attack against C^*. Because this latter attack has no equivalent for HFE, there is no known attack against the subfield variant of HFE.

As mentioned above, the complexity of nearly all existing attacks on HFE depends on the degree of the internal secret polynomial. Even the most concrete and realistic threat, namely computing a Gröbner basis of the public-key, will become unrealistic if this degree is chosen high enough (a drawback is that decryption then becomes slower). A nice feature of the attack presented in this paper is that its asymptotic complexity is only marginally affected by the degree of the internal polynomial. As such, it be applied *in practice* to HFE instances on which existing attacks would be completely intractable. We also argue that under standard conjectures on the complexity of Gröbner basis computation, it is possible to establish that the complexity of our remains polynomial when the degree of the internal polynomial grows polynomially with n.

19.1 Hidden Field Equations

The HFE scheme was designed in [Pat96b] by Patarin. Note that specific variations of HFE exist, but we will focus on the basic HFE scheme. Let us briefly recall its mechanism.

Let $\mathbb{K} = \mathbb{F}_q$. The HFE secret key is made up of an extension $\mathbb{L}$ of degree n over $\mathbb{K}$, a low-degree polynomial $\mathbf{f}$ over $\mathbb{L}$, and two invertible affine mappings S and T over $\mathbb{K}^n$. The secret polynomial $\mathbf{f}$ has the following particular shape:

$$\mathbf{f}(X) = \sum_{\substack{0 \leq i,j \leq n \\ q^i + q^j \leq d}} a_{i,j} \cdot X^{q^i+q^j} + \sum_{\substack{0 \leq k \leq n \\ q^k \leq d}} b_k \cdot X^{q^k} + c, \tag{19.1}$$

with the $a_{i,j}$, the b_k and c lying in $\mathbb{L}$. Polynomials with the same shape as $\mathbf{f}$ are called HFE polynomials. Note that these polynomials were also studied much earlier in a completely different context by Dembowski and Ostrom [DO68], so they are sometimes referred to as D–O polynomials in the literature. Because decryption requires to invert $\mathbf{f}$, the maximum degree of $\mathbf{f}$, denoted by d, has to be chosen so that the factorization of $\mathbf{f}$ over $\mathbb{L}$ is efficient. All known algorithms for factorizing over finite fields are at least quadratic in the degree of the polynomial, which restricts d to values smaller than about 2^{16}. It also makes sense to consider degree bounds of the form $d = 2 \cdot q^D$, because in equation (19.1), we may then consider the sum over values of i and j smaller than D. Because the iterates of the Frobenius are $\mathbb{K}$-linear, then $\mathbf{f}$, seen as a transformation of $\mathbb{K}^n$, can be represented by a vector of n *quadratic* polynomials in n variables over $\mathbb{K}$. This property extends to the public key of the basic HFE scheme, defined by $\mathbf{PK} = T \circ \mathbf{f} \circ S$. In order to offer non-trivial encryption, $\mathbf{f}$ must logically be non-linear. Also, because HFE was designed specifically to circumvent the attack that destroyed C*, we will assume that the internal polynomial always has at least two non-linear terms.

Note that when $\mathbb{K} = \mathbb{F}_2$, we may assume $a_{i,i} = 0$, by choosing b_{i+1} accordingly. As such, there are $D(D+5)/2+3$ terms in $\mathbf{f}$ when $q \neq 2$ and $(D+2)(D+1)/2+2$ terms when $q = 2$.

19.1.1 Equivalent Keys for HFE

In HFE, the public key can be derived from the secret key in polynomial time by an algorithm **PKGen** that takes as argument $T, \mathbf{f}, S$ and $\mathbb{L}$ (*i.e.*, the irreducible polynomial P defining $\mathbb{L}$ and the correspondance φ between $\mathbb{L}$ and $\mathbb{K}^n$). Two secret keys are *equivalent* if they yield the same public key. For instance, it was shown in [WP05a] that an HFE public-key is always generated by a secret key in which S and T are linear (as opposed to affine). The affine part of S and T can be removed by changing the constant component of $\mathbf{f}$. Next, if $\alpha, \beta \in \mathbb{L}$, then it is possible to simultaneously replace T by $T \cdot M_\alpha$ and S by $M_\beta \cdot S$. It is sufficient to replace $\mathbf{f}$ by $\alpha \cdot \mathbf{f}(\beta^{-1} \cdot X)$ in order for the public key to remain the same, and this allows to choose the values of both S and T on one point.

As a consequence, a set of $q^{2n} \cdot (q^n - 1)^2$ equivalent secret keys is identified (this number assumes that $\mathbb{L}$ is fixed). It was not formally established that all the equivalent secret keys belong to this set, even though this seems likely when $\mathbf{f}$ is a randomly-chosen HFE polynomial.

19.1.2 Irrelevance of Keeping the Extension Field Secret

While the original description of HFE [Pat96b] explicitly specifies that the extension field $\mathbb{L}$ must be part of the secret key, the same paper notes that this does not improve the security of the trapdoor, because there always exist equivalent secret keys for all the possible descriptions of $\mathbb{L}$. As a matter of fact, the specifications of both Quartz [PCG01b] and SFLASH [PCG01a] make the extension field public. In any case, it is possible to generate the *same* public key from the *same* secret polynomial, while fixing an arbitrary irreducible polynomial P defining $\mathbb{L}$, and an arbitrary correspondence between $\mathbb{L}$ and $\mathbb{K}^n$. It simply requires slight modifications on S and T. Indeed, recall that any isomorphism between $\mathbb{K}^n$ and $\mathbb{L}$ must be an invertible $\mathbb{K}$-linear map [LN97]. We will use this in the following proposition to justify our point:

Proposition 19.1. *Let* $\mathbf{SK} = (T, \mathbf{f}, S, P, \varphi)$ *be an HFE secret key. Then for any choice of an extension field* $\mathbb{L}' = \mathbb{K}[X]/P'(X)$ *of degree* n*, and for any choice of an isomorphism* φ' *between* $\mathbb{L}'$ *and* $\mathbb{K}^n$*, there exist two affine bijections* S' *and* T' *such that* $\mathbf{SK}' = (T', \mathbf{f}, S', P', \varphi')$ *is equivalent to* $\mathbf{SK}$ *(i.e., generates the same public key).*

Proof. It is well-known [LN97] that all finite fields of the same cardinality are isomorphic. Therefore let us consider a field isomorphism $\zeta : \mathbb{L} \to \mathbb{L}'$. Recall that $\varphi : \mathbb{K}^n \to \mathbb{L}$ and $\varphi' : \mathbb{K}^n \to \mathbb{L}'$ are both isomorphisms as well. The notation $\mathbf{PK} = T \circ \mathbf{f} \circ S$ is unambiguous when the extension field $\mathbb{L}$ is clearly defined. Here we will write:

$$\begin{aligned} \mathbf{PK} &= T \circ \varphi^{-1} \circ \mathbf{f}_{\mathbb{L}} \circ \varphi \circ S \\ \mathbf{PK}' &= T' \circ \varphi'^{-1} \circ \mathbf{f}_{\mathbb{L}'} \circ \varphi' \circ S' \end{aligned}$$

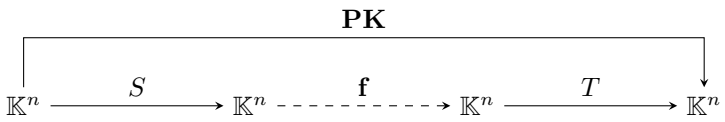


Figure 19.1: Description of weak keys. The broken arrow indicates that $\mathbf{f}$ has coefficients in $\mathbb{K}$.

Let us solve $\mathbf{PK} = \mathbf{PK}'$ for S' and T'. Because the two internal polynomial in $\mathbf{PK}$ and $\mathbf{PK}'$ are the same, we can write:

$$\varphi \circ T^{-1} \circ \mathbf{PK} \circ S^{-1} \circ \varphi^{-1} = \zeta^{-1} \circ \varphi' \circ T'^{-1} \circ \mathbf{PK}' \circ S'^{-1} \circ \varphi'^{-1} \circ \zeta$$

And it follows that in order to enforce $\mathbf{PK} = \mathbf{PK}'$ it is sufficient to have:

$$\begin{aligned} T' &= T \cdot \left(\varphi'^{-1} \circ \zeta \circ \varphi\right)^{-1} \\ S' &= S \cdot \left(\varphi'^{-1} \circ \zeta \circ \varphi\right) \end{aligned}$$

And since $\varphi'^{-1} \circ \zeta \circ \varphi$ is an automorphism of $\mathbb{K}^n$, we have $S', T' \in \mathrm{GL}_n(\mathbb{K})$. Thus the secret key $(T', \mathbf{f}, S', P', \varphi')$ is equivalent to $(T, \mathbf{f}, S, P, \varphi)$. □

Thus, keeping the representation of $\mathbb{L}$ secret does not improve the resistance of HFE to key-recovery attacks. Would the extension be secret, one could just arbitrarily fix its own and be guaranteed that an equivalent secret key exists. As a consequence, throughout the sequel, we assume that the description of $\mathbb{L}$ is public.

19.1.3 Linear Polynomials

Let f be an endomorphism of $\mathbb{K}^n$. It can be represented by a matrix M over $\mathbb{K}^n$, but also as a polynomial over $\mathbb{L}$. Such $\mathbb{K}$-linear (or "additive") polynomials only have monomials of degree q^i, for $0 \leq i \leq n-1$. In the sequel, we will always identify an endormophism of $(\mathbb{F}_q)^n$ with its polynomial representation over $\mathbb{L}$, and we will refer to the *polynomial representation over* $\mathbb{L}$ of such an endomorphism. The set of matrices commuting with F over $\mathcal{M}_n(\mathbb{K})$ is the $\mathbb{K}$-vector space of dimension n generated by $\left(\mathrm{F}^0, \mathrm{F}, \ldots, \mathrm{F}^{n-1}\right)$.

19.2 A Specific Family of HFE Secret Polynomials

Similarly to the attacks against C^* or SFLASH, the main idea we exploit is that some HFE secret polynomials may *commute* with some special functions. This commutativity property can then in turn be used to acquire informations on the secret elements.

A Commutativity property for Some HFE Secret Polynomials. Let us first consider the *à la* C^* case, where the secret polynomial $\mathbf{f}$ over $\mathbb{L}$ is just a monomial $a \cdot X^{q^i+q^j}$, with $a \in \mathbb{L}$. Then the public key $\mathbf{PK} = T \circ \mathbf{f} \circ S$ can also be written as $(T \cdot M_a) \circ X^{q^i+q^j} \circ S$, by "absorbing" the multiplication by the constant a into the outer secret linear transformation. As a consequence, without loss of generality, we can assume that $a = 1$.

This secret monomial has very special commutativity properties, which were used in [DFS07, DFSS07] to break SFLASH. More precisely, composing it on the right hand size by multiplications M_x by an element $x \in \mathbb{L}$ is equivalent to composing it on the left hand size by $M_{x^{q^i+q^j}}$. Another property, not used in [DFS07, DFSS07], is that it also commutes with the Frobenius map F (and hence with the iterates of the Frobenius map).

When we consider an arbitrary HFE secret polynomial, the two commutation properties no longer hold in general. However, if we restrict the HFE polynomials to have their coefficients in $\mathbb{K}$ (instead of the extension field $\mathbb{L}$), we lose commutativity with multiplications but commutativity with the Frobenius map still remains. In the sequel, we show how this specific property can be exploited to perform a key-recovery attack, described in §19.3. Therefore, we will say the the secret keys in which the internal polynomial has coefficients in $\mathbb{K}$ are *weak secret keys*. Such instances of HFE are illustrated by Figure 19.1.

Our key-recovery attack could also apply to monomial instances of HFE, but this is not the point of this paper, as it has already been efficiently done [DFS07, DFSS07, FMRS08].

Corresponding Public Keys. The attack we discuss recovers a useful secret key by exploiting only knowledge of the public key, and it works at the sole condition that the public key can be generated by a weak secret key. Therefore, we will say that such public keys are *weak public keys*. We note that a weak public key has not necessarily been generated by a weak secret key. For instance, let $\mathbf{f}$ be an HFE polynomial with coefficients in $\mathbb{K}$, and let $a, b \in \mathbb{L}$. Then let us define $\mathbf{f}' : x \mapsto a \cdot \mathbf{f}(b \cdot x)$. This HFE polynomial has coefficients in $\mathbb{L}$. Now let $S, T \in \mathrm{GL}_n(\mathbb{K})$, and consider the HFE secret key $(T, \mathbf{f}', S)$. It does not fall into our definition of weak secret keys, because $\mathbf{f}' \notin \mathbb{K}[X]$, but it generates a weak public key: it is straightforward that it is equivalent to $(T \cdot M_a, \mathbf{f}, M_b \cdot S)$. To summarize, our attack will be applicable whenever the internal polynomial has the following shape:

$$\mathbf{f}(X) = \sum_{\substack{0 \le i,j \le n \\ q^i + q^j \le d}} \left(a \cdot u_{i,j} \cdot b^{q^i+q^j}\right) \cdot X^{q^i+q^j} + \sum_{\substack{0 \le k \le n \\ q^k \le d}} \left(a \cdot v_k \cdot b^{q^k}\right) \cdot X^{q^k} + a \cdot c, \tag{19.2}$$

with $u_{i,j}, v_k, c \in \mathbb{K}$, $a, b \in \mathbb{L}$. Notice that legitimate users could easily check whether their secret key generates a weak public key by checking if their internal polynomial can be written as in equation (19.2).

19.2.1 An Estimation of the Cardinality of this Family

We now show that the probability that the uniform random choice of a secret key yields a weak public key is by all means negligible. This unfortunate event happens if and only if the internal polynomial $\mathbf{f}$ has the shape given by equation (19.2). We therefore go on to count the number of such polynomials (we will call them "weak polynomials" for the sake of brevity). Let us denote their number by $\#WP$, and the number of HFE polynomials of degree d over $\mathbb{L}$ by $\#HFE(d, n, \mathbb{L})$. An obvious upper-bound is found by assuming that for any choice of $\alpha, \beta \in \mathbb{L}^*$ and $\mathbf{f} \in \mathbb{K}[X]$, $M_\alpha \circ \mathbf{f} \circ M_\beta$ is a distinct polynomial. This yields:

$$\#WP \le (q^n - 1)^2 \cdot \#HFE(d, n, \mathbb{K})$$

However this bound can be improved, because some polynomials are counted several times. For instance, if $\pi \in \mathbb{K}$ and $\Pi \in \mathbb{L}$, then $\Pi \cdot \mathbf{f} = \left(\Pi \cdot \pi^{-1}\right) \cdot (\pi \cdot \mathbf{f})$. The same goes for right-composition. The bound therefore improves to:

$$\#WP \le \left(\frac{q^n - 1}{q - 1}\right)^2 \cdot \#HFE(d, n, \mathbb{K})$$

In the simpler case where $d = 2q^D$ and $q \neq 2$, then we have $\#HFE(d, n, \mathbb{K}) = q^{\frac{D(D+5)}{2}+3}$, and therefore the probability of randomly generating a weak public key is upper-bounded by:

$$\frac{\#WP}{\#HFE(d, n, \mathbb{L})} = \left(\frac{q^n - 1}{q - 1}\right)^2 \cdot \left(q^{\frac{D(D+5)}{2}+3}\right)^{-(n-1)} = \mathcal{O}\left(\left(q^{2-D^2/2}\right)^{n-1}\right)$$

This shows that the probability of generating a weak key out of bad luck is exponentially small in the security parameter. In the same vein, we could obtain a fairly obvious lower-bound on this probability by counting only the polynomials of the form $M_\alpha \circ \mathbf{f}$. Indeed, nothing rules out the existence of several pairs (α, β) and (α', β') such that $M_\alpha \circ \mathbf{f} \circ M_\beta = M_{\alpha'} \circ \mathbf{f} \circ M_{\beta'}$, and for this reason our upper-bound is not necessarily a lower-bound. Their number is delicate to estimate, because it depends on the shape and coefficients of $\mathbf{f}$.

19.3 The Key-Recovery

We now describe a key-recovery attack against the class of weak keys described in §19.2. In the sequel, we then assume that the internal secret polynomial $\mathbf{f}$ has coefficients in $\mathbb{K}$.

As the attack is a bit complex, let us first give an overview. A pseudo-code of the attack is given in Algorithm 19.1. First, we show that the representation of $\mathbb{L}$ can be supposed to be public. Then, as already mentioned in §19.2, we use the commutation of the Frobenius map with the secret polynomials considered, which propagates to the public key $\mathbf{PK}$. This key property allows us to recover applications closely related to S and T. An interpolation of $\mathbf{PK}$ combined with these applications then gives us a polynomial over $\mathbb{K}$ from which we recover $\mathbf{f}$ or an equivalent low-degree polynomial by computing a functional decomposition. In any case, we obtain the original secret key or an equivalent one that allows us to decrypt as efficiently as the secret key owner. All these assertions are made explicit and justified in this section.

Algorithm 19.1 Pseudo-code of the attack

Require: An HFE public key **PK**, generated by $(T, \mathbf{f}, S)$ such that $\mathbf{f} \in \mathbb{K}[X]$.
Ensure: An equivalent secret key: $(T', \mathbf{f}', S')$, with $\deg \mathbf{f}' \leq \deg \mathbf{f}$.

```
repeat
    Let U, V ∈ GL_n(K) be a (random) solution to the IP problem: U ∘ PK = PK ∘ V.
until there exist P, Q ∈ GL_n(K) such that U = P · F · Q
for all i_0 in [1; n − 1] co-prime with n do
    Let k = i_0^{-1} mod n
    Compute S̃, T̃ such that F = S̃ · V^k · S̃^{-1} = T̃^{-1} · U^k · T̃
    Interpolate g = T̃^{-1} · PK · S̃^{-1}.
    if g has all coefficients in K then
        Compute F_1, F_2 ∈ GL_n(K) and f_2 ∈ K[X], such that deg f_2 ≤ deg f and g ∘ F_1 = F_2^{-1} ∘ f_2.
        return (T̃ · F_2^{-1}, f_2, F_1^{-1} · S̃)
    end if
end for
```

19.3.1 A Useful Property of HFE Secret Polynomials Lying in $\mathbb{K}[X]$

Recall from §10.1 that because $\mathbf{f}$ has coefficients in $\mathbb{K}$, then it commutes with F:

$$\mathbf{f} \circ \mathrm{F}(X) = \mathrm{F} \circ \mathbf{f}(X) \tag{19.3}$$

Patarin left as an open problem whether this property has security implications or not. We shall demonstrate that it does indeed. Most importantly, this property is detectable in the public-key.

Proposition 19.2. *There exist non-trivial endomorphisms L such that $L \circ \mathbf{PK} = \mathbf{PK} \circ L$. More precisely, the invertible map ψ defined below transforms a matrix M that commutes with $\mathbf{f}$ into a solution of the above equation:*

$$\psi : M \mapsto \left(T \cdot M^{-1} \cdot T^{-1}, S^{-1} \cdot M \cdot S\right)$$

As a consequence, $\psi(F), \ldots, \psi\left(F^{n-1}\right)$ are non-trivial automorphisms of **PK**.

Proof. Let M be a matrix such that $\mathbf{f} \circ M = M \circ \mathbf{f}$. Then we get:

$$\begin{aligned} \mathbf{PK} \circ (S^{-1} \cdot M \cdot S) &= T \circ \mathbf{f} \circ S \cdot S^{-1} \cdot M \cdot S \\ &= T \circ M \circ \mathbf{f} \circ S \\ &= (T \cdot M \cdot T^{-1}) \circ \mathbf{PK} \\ \Leftrightarrow \quad \mathbf{PK} &= (T \cdot M \cdot T^{-1})^{-1} \circ \mathbf{PK} \circ (S^{-1} \cdot M \cdot S) \end{aligned}$$

Then, because of (19.3), $\psi(F), \ldots, \psi\left(F^{n-1}\right)$ are automorphisms of the public key. □

The existence of other solutions besides those mentioned in proposition 19.2 is extremely unlikely, unless the situation is very degenerate. Indeed, this would imply the existence of other linear applications commuting with the (non-linear) internal polynomial. However, besides the monomial instances, where multiplication matrices commute in some sense with $\mathbf{f}$, we are not aware of instances that would verify such a property. Thus, if we consider a particular solution of the problem of retrieving an automorphism of the public-key, we can assume that it is $\psi\left(F^{i_0}\right)$, for some unknown power i_0.

Hardness of the QMLE Problem. Determining the automorphisms of the public key can be done by running a PLE algorithm. We will be able to use IQMLE algorithms when:

i) The secret transformations S and T are linear (as opposed to affine).

ii) The $\mathbb{K}$-linear coefficients b_k of (19.2) are not all zero.

iii) The constant coefficient c of (19.2) is non-zero.

The first condition can only be satisfied if choosing linear S and T was a deliberate decision (otherwise it will only happen with negligible probability). There are good reasons of doing so: first it reduces a bit the size of the private key. Second, as shown in §19.1.1, S and T can be assumed to be linear, because of the existence of equivalent keys. However, we stress that this last fact is *no longer true* if the internal polynomial $\mathbf{f}$ is chosen in $\mathbb{K}[X]$ instead of $\mathbb{L}[X]$. Series of bad design decisions could still lead to the combination of a restricted $\mathbf{f}$ *and* linear S and T.

The second condition will always be satisfied with high probability, and the third will be satisfied with probability $1/q$. It must be noted that if S and T are linear, and if $c = 0$ in (19.1), then the public-key sends zero to zero, which might not be desirable.

In the case where S and T are affine, the situation is much more painful, as we are facing a homogeneous instance of QMLE, and our best shot is algorithm 18.4, that has a workload of $q^{n/2}$. In the case of the "subfield variant" though, all the numerical quantities lie in a subfield $\mathbb{F}_{q'}$ of $\mathbb{F}_q$, where q' is quite small (the typical value is $q' = 2$). This makes breaking the QMLE instances feasible in practice for the subfield variant (we discuss this issue further in §19.4).

19.3.2 Retrieving "nearly S" and "nearly T" Applications

Let us assume that we have found an automorphism $(U, V) = \psi\left(\mathrm{F}^{i_0}\right)$ of the public-key, for some unknown integer i_0 in the interval $[1; n-1]$. The whole point of the attack is to "extract" enough information about S and T from this automorphism. For this purpose, the value of i_0 has to be known, and it is required that i_0 and n are relatively prime. This latter condition can be easily checked for: F^i and F^j are similar if and only if $\gcd(i, n) = \gcd(j, n)$. Therefore, i_0 is relatively prime with n if U and F are similar. If this turns out not to be the case, we take another automorphism of **PK**, until it passes the test. Since there are $\phi(n)$ values of i_0 that are prime with n, we expect to check $n/\phi(n) = \mathcal{O}(\log \log n)$ candidates.

To derive the actual value of i_0, we simply guess its value, and check whether the remaining steps of the attack are carried out successfully. Fortunately, there is a way to discard bad guesses systematically before the most computationally expensive step of the attack, as we will explain in §19.3.3. With the preceding notations, we have the following result:

Proposition 19.3. *Let $(U, V) = \psi\left(\mathrm{F}^{i_0}\right)$, with $\gcd(i_0, n) = 1$. Let k be such that $k \cdot i_0 = 1 \bmod n$.*

i) There exist $\widetilde{S}$, $\widetilde{T}$ in $\mathrm{GL}_n(\mathbb{K})$ such that $\mathrm{F} = \widetilde{S} \cdot V^k \cdot \widetilde{S}^{-1}$ and $\mathrm{F} = \widetilde{T}^{-1} \cdot U^k \cdot \widetilde{T}$.

ii) Both $\widetilde{S} \cdot S^{-1}$ and $\widetilde{T} \cdot T^{-1}$ commute with F, hence their polynomial representations over $\mathbb{L}$ are in fact polynomials with coefficients in $\mathbb{K}$.

Proof. The first point follows from the fact that U and V are both similar to F^{i_0}. Thus U^k and V^k are both similar to $\mathrm{F}^{i_0 \cdot k} = \mathrm{F}$.

Regarding the second point, let us consider the case of $\widetilde{S}$ (something similar holds for $\widetilde{T}$). We have:

$$\begin{aligned} \mathrm{F} &= \widetilde{S} \cdot V^k \cdot \widetilde{S}^{-1} \\ &= \widetilde{S} \cdot S^{-1} \cdot \mathrm{F}^{i_0 \cdot k} \cdot S \cdot \widetilde{S}^{-1} \\ &= \widetilde{S} \cdot S^{-1} \cdot \mathrm{F} \cdot S \cdot \widetilde{S}^{-1} \end{aligned}$$

And thus $\mathrm{F} \cdot \widetilde{S} \cdot S^{-1} = \widetilde{S} \cdot S^{-1} \cdot \mathrm{F}$. This commutation property directly implies the announced result on the polynomial representations (cf. §10.1). □

In practice, $\widetilde{S}$ and $\widetilde{T}$ can be found very efficiently through linear algebra, given that i_0 is known. Note that for now, this proposition cannot be used to test whether our current guess for i_0 is correct, since we do not know S.

19.3.3 Building an Equivalent Secret Key

The information about S (resp. T) contained in $\widetilde{S}$ (resp. $\widetilde{T}$) can be used to cancel the action of S and T on the public key. It follows from proposition 19.3 that $F_1 = \widetilde{S} \cdot S^{-1}$ and $F_2 = T^{-1} \cdot \widetilde{T}$ are linear combinations over $\mathbb{K}$ of powers of F, because they commute with F. We immediately obtain that:

$$\begin{aligned} \widetilde{T}^{-1} \circ \mathbf{PK} \circ \widetilde{S}^{-1} &= F_2^{-1} \circ T^{-1} \circ T \circ \mathbf{f} \circ S \circ S^{-1} \circ F_1^{-1} \\ &= F_2^{-1} \circ \mathbf{f} \circ F_1^{-1}. \end{aligned} \tag{19.4}$$

We therefore define:

$$\mathbf{g} = \widetilde{T}^{-1} \circ \mathbf{PK} \circ \widetilde{S}^{-1} \bmod \left(X^{q^n} - X\right) \tag{19.5}$$

Because the HFE polynomials are stable by left and right composition by additive polynomials and by reduction modulo $X^{q^n} - X$, the "peeled off" polynomial $\mathbf{g}$ is still an HFE polynomial. Thus $\mathbf{g}$ has $\mathcal{O}\left(n^2\right)$ coefficients, and that they can be uniquely determined in polynomial time by interpolation (this was noted in [KS99]. Note that there would not be a unique solution if we did not perform the modular reduction of $\mathbf{g}$). By doing so, we obtain an equivalent secret key, namely $\left(\widetilde{T}, \mathbf{g}, \widetilde{S}\right)$.

By itself, this equivalent key is not particularly useful, since the degree of $\mathbf{g}$ is typically q^n, and we are therefore still facing our initial task of factorizing a sparse polynomial of very high degree. However, $\mathbf{g}$ has a very important property which brings us one step closer to the original secret-key:

Proposition 19.4. *The coefficients of $\mathbf{g}$ are in $\mathbb{K}$ (and not in $\mathbb{L}$).*

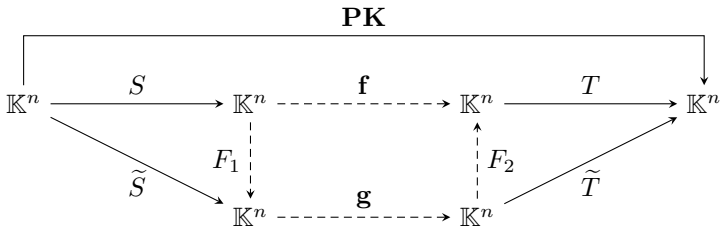


Figure 19.2: $\mathbf{PK} = T \circ \mathbf{f} \circ S = \widetilde{T} \circ \mathbf{g} \circ \widetilde{S}$. Broken arrows stand for applications with coefficients in $\mathbb{K}$.

Proof. By hypothesis, the coefficients of $\mathbf{f}$ are in $\mathbb{K}$. From proposition 19.3, item *ii*), we have that the coefficients of the polynomial representation of F_1 and F_2 are in $\mathbb{K}$, then, so are those of the polynomial representations of ${F_1}^{-1}$ and ${F_2}^{-1}$:

Lemma 19.5. *Let $M \in \mathrm{GL}_n(\mathbb{K})$ be an invertible matrix, and let $P = \sum_{i=0}^{n-1} a_i \cdot X^i$ (resp. $Q = \sum_{i=0}^{n-1} b_i \cdot X^i$) the polynomial representation of M over $\mathbb{L}$ (resp. M^{-1}). In general the a_i's and b_i's live in $\mathbb{L}$. But we have:*

$$(a_0, a_1, \ldots, a_{n-1}) \in \mathbb{K}^n \iff (b_0, b_1, \ldots, b_{n-1}) \in \mathbb{K}^n$$

proof of lemma 19.5. If the polynomial representation of M has coefficients in $\mathbb{K}$, then M commutes with F. This implies that M^{-1} also commutes with F, which in turn implies that the polynomial representation of M^{-1} has coefficients in $\mathbb{K}$. □

This and (19.4) shows that $\widetilde{T}^{-1} \circ \mathbf{PK} \circ \widetilde{S}^{-1}$ has coefficients in $\mathbb{K}$. From there, it is straightforward that $\mathbf{g}$ has coefficients in $\mathbb{K}$. □

The result of proposition 19.4 is illustrated in figure 19.2. This figure also helps remembering how the applications introduced so far intervene. This proposition can be used to verify if our guess for i_0 was right. Indeed, if $\mathbf{g}$ is found not to be in $\mathbb{K}[X]$, then the guess was wrong. We are aware that the fact that $\mathbf{g} \in \mathbb{K}[X]$ does not rigorously prove that we have found the right value of i_0. However, it does not matter, as $\mathbf{g} \in \mathbb{K}[X]$ is sufficient for the subsequent step to work.

19.3.4 Recovering a Low-Degree Equivalent Secret Key

To be useful, an equivalent secret key must have an internal polynomial of low degree. We now show how to obtain one, by actually computing the decomposition given by equation (19.4). This is in fact a *much easier* problem than computing the equivalent decomposition on the original public key, because we deal with applications whose coefficients belong to $\mathbb{K}$. They are then left invariant by the Frobenius (hence by F_1 and F_2), which implies that the problem of finding the decomposition reduces to finding a solution of an overdetermined system of *quadratic* equations. This system can be solved in practical time by computing a Gröbner basis, as we now show. To this end, we introduce the following notations, where all the coefficients in the expressions are now known to lie in $\mathbb{K}$:

$$\begin{aligned}
F_1(X) &= \sum_{k=0}^{n-1} x_k X^{q^k} \qquad F_1^{-1}(X) = \sum_{k=0}^{n-1} y_k X^{q^k} \\
F_2(X) &= \sum_{k=0}^{n-1} z_k X^{q^k} \qquad F_2^{-1}(X) = \sum_{k=0}^{n-1} t_k X^{q^k} \\
\mathbf{g}(X) &= \sum_{q^i+q^j<q^n} a_{ij} X^{q^i+q^j} + \sum_{i=0}^{n-1} b_i X^{q^i} + c \\
\mathbf{f}_2(X) &= \sum_{q^i+q^j\le d} e_{ij} X^{q^i+q^j} + \sum_{q^i\le d} f_i X^{q^i} + g
\end{aligned}$$

Then, we consider the following polynomial equation, also represented by figure 19.3, obtained by composing both sides of equation (19.4) with F_1:

$$\mathbf{g} \circ F_1 = F_2^{-1} \circ \mathbf{f}_2. \tag{19.6}$$

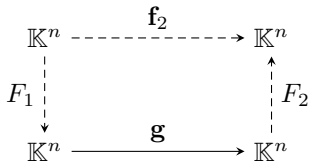


Figure 19.3: $\mathbf{g} = F_2^{-1} \circ \mathbf{f}_2 \circ F_1^{-1}$. Broken arrows stand for applications with unknown coefficients.

Let us now substitute the expression of F_1 and $\mathbf{g}$ in the left-hand-side of (19.6). We find:

$$\begin{aligned}\mathbf{g} \circ F_1 &= \sum_{q^i+q^j<q^n} a_{ij} \left(\sum_{k=0}^{n-1} x_k X^{q^k}\right)^{q^i+q^j} + \sum_{i=0}^{n-1} b_i \left(\sum_{k=0}^{n-1} x_k X^{q^k}\right)^{q^i} + c \\ &= \sum_{\substack{k,l\in\{0,\ldots,n-1\} \\ q^i+q^j<q^n}} a_{ij} \cdot x_k \cdot x_l \cdot X^{q^{i+k}+q^{l+j}} + \sum_{i,k\in\{0,\ldots,n-1\}} b_i \cdot x_k \cdot X^{q^{i+k}} + c,\end{aligned}$$

We observe that $\mathbf{g} \circ F_1$ is a polynomial whose coefficients are quadratic polynomials in the coefficients of F_1. Now, let us substitute the expression of F_2^{-1} and $\mathbf{f}_2$ in the right-hand-side of (19.6). We find:

$$\begin{aligned}F_2^{-1} \circ \mathbf{f}_2 &= \sum_{k=0}^{n-1} t_k \left(\sum_{q^i+q^j\le d} e_{ij} X^{q^i+q^j} + \sum_{q^i\le d} f_i X^{q^i} + g\right)^{q^k} \\ &= \sum_{\substack{k\in\{0,\ldots,n-1\} \\ q^i+q^j\le d}} t_k \cdot e_{ij} \cdot X^{q^{i+k}+q^{j+k}} + \sum_{\substack{q^i\le d \\ k\in\{0,\ldots,n-1\}}} t_k \cdot f_i \cdot X^{q^{i+k}} + g \cdot \sum_{k=0}^{n-1} t_k\end{aligned}$$

We again find that $F_2^{-1} \circ \mathbf{f}_2$ is a polynomial whose coefficients are quadratic polynomials in the coefficients of both $\mathbf{f}_2$ and F_2^{-1}.

This shows that (19.6) is equivalent to a system of multivariate quadratic equations over $\mathbb{K}$ with $\mathcal{O}\left(n^2\right)$ quadratic equations and $\mathcal{O}\left(n + D^2\right)$ unknowns, the unknowns being the coefficients of F_1, F_2^{-1} and $\mathbf{f}_2$. This system can be generated by reducing both sides of (19.6) modulo $X^{q^n} - X$ and identifying the coefficients of the monomials in X. The problem of computing the decomposition of equation (19.6) is therefore reduced to that of solving an overdetermined system of quadratic equations.

However, equation (19.6) admits many parasitic solutions (for example, $F_1 = \mathbf{f}_2 = 0$, F_2^{-1} being arbitrary). To avoid these trivial solutions, we in fact consider an extended system:

$$\left\{\begin{array}{rcl} F_1 \circ F_1^{-1} &=& Id \\ F_2 \circ F_2^{-1} &=& Id \\ \mathbf{g} \circ F_1 &=& F_2^{-1} \circ \mathbf{f}_2 \end{array}\right. \tag{19.7}$$

This new system avoids the parasitic solutions by forcing F_1 and F_2 to be invertible. We now argue that (19.7) is also equivalent to a system of quadratic equations, whose unknowns are the coefficients of $F_1, F_1^{-1}, F_2, F_2^{-1}$ and $\mathbf{f}_2$. The third equation has already been shown to be translatable to multivariate quadratic equations. Substituting the definitions of F_1 and F_1^{-1} in $F_1 \circ F_1^{-1}$ yields:

$$\sum_{k=0}^{n-1} \sum_{\ell=0}^{n-1} (x_k \cdot y_\ell) \cdot X^{q^{\ell+k}} = X$$

We observe that the coefficients of the left-hand side are quadratic in the x_k's and y_k's, therefore we obtain n quadratic equations by reducing the LHS modulo $X^{q^n} - X$ and equating the coefficients on both sides of the equation. The same goes for $F_2 \circ F_2^{-1} = Id$.

All in all, assuming that the degree of $\mathbf{f}$ is $d = 2q^D$, this yields $n(n+3)/2+1$ equations in $4n+D(D+5)/2+4$ variables, not counting eventual field equations (one per variable). The existence of at least one solution is guaranteed, because of equation (19.4), as long as we picked the right power of the Frobenius matrix in §19.3.1. In fact, even though we just need one, we know that many solutions exist: for instance because the Frobenius commutes with everything in equation (19.6), we can take a particular solution, compose both F_2^{-1} and F_1 with the Frobenius, and obtain a new solution.

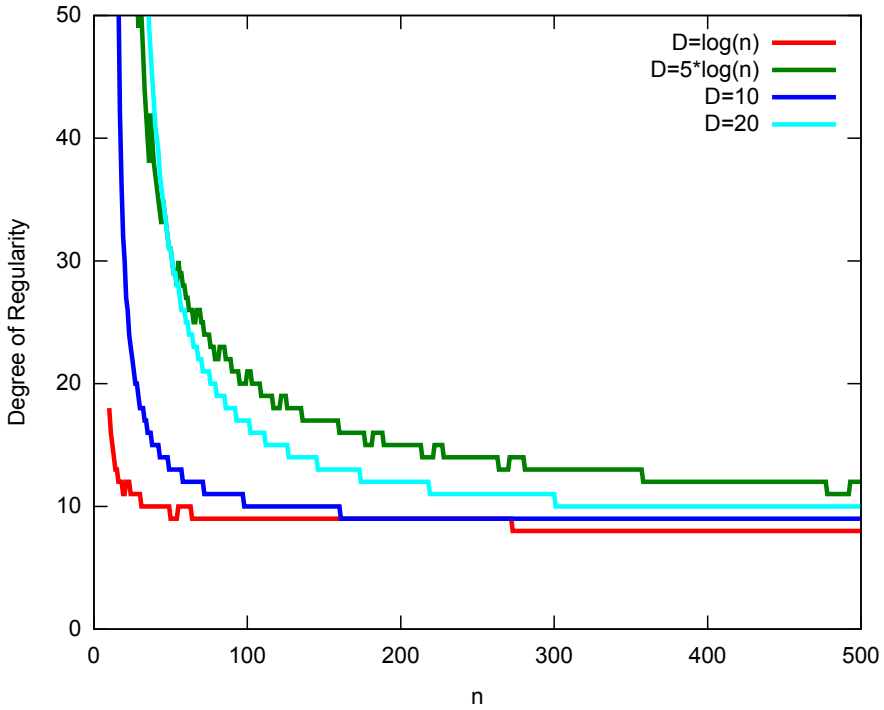


Figure 19.4: Degree of regularity of semi-generic systems of $n(n+3)/2+1$ quadratic equations in $4n + D(D+5)/2+4$ variables.

It turns out that these equations can be solved efficiently, even though the number of variables is higher than what is usually tractable, because it is very overdetermined: we have $\mathcal{O}\left(n^2\right)$ equations in $\mathcal{O}\left(n + D^2\right)$ variables, and D has to be small for decryption to be efficient (*i.e.*, $D = \mathcal{O}(\log n)$). In this setting, computing a Gröbner basis turns out to be feasible in practice.

Conjecture 19.1. The Gröbner basis of a system of random quadratic equations with the same number of variable and polynomials as our equations can be computed by manipulating polynomials of degree at most 8. Thus, it can be computed in time at most $\mathcal{O}\left(n^{24}\right)$ by the $\mathbf{F_5}$ algorithm. This is true if D is fixed, or even if grows polynomially with $\log n$.

Justification of the Conjecture. This conjecture is in fact equivalent to assuming that our system of quadratic equations forms a semi-regular sequence. Under this assumption, we might make use of the results on the complexity of solving very overdetermined systems stated in §10.6.1. We are indeed dealing with $n(n+3)/2+1$ quadratic equations in $4n + D(D+5)/2+4$ variables, so we let

$$\beta = \frac{n(n+3)/2+1}{(4n + D(D+5)/2+4)^2},$$

and we realize that we are dealing with $\beta \cdot n^2$ equations in n variables. When n grows to infinity, then β converges to $1/32$, even if $D = \log n$. Thus, as soon as n gets big enough, the degree of regularity of the corresponding equations should become close to its finite limit. Equation (10.3) and Figure 10.2 tells us that this limit is 8. This shows that computing a Gröbner basis of $n(n+3)/2+1$ semi-regular equations in $4n + D(D+5)/2+4$ variables is (asymptotically) polynomial. The actual degrees of regularity, computed using the Hilbert series, are shown in Figure 19.4 for various values of n and D.

Comments and Practical Results. While the above conjecture means that computing the polynomial decomposition we are dealing with should be polynomial, some remarks are in order. First, our equations are not random, not to mention semi-regular. This follows from the fact that they admit many solutions, while a random overdetermined system has no solutions with overwhelming probability. Next, our experiments (for various values of n and D) indicate that a Gröbner basis can be computed by manipulating polynomials of degree at most 3, leading to an empirical complexity of $\mathcal{O}\left(n^9\right)$. Our equations are thus *easier* to solve than random systems with the same parameters.

Once the equations are solved, we recover an equivalent secret-key $\left(\widetilde{T} \cdot F_2^{-1}, \mathbf{f}_2, F_1^{-1} \cdot \widetilde{S}\right)$, which allows us to decrypt with the same time complexity as the legitimate user, since $\mathbf{f}_2$ has essentially the same degree as $\mathbf{f}$.

19.4 Applications and Experiments

We have implemented the HFE key-generation and encryption, as well as the attack, in the MAGMA computer-algebra system. In Table 19.1, we give five different sets of parameters. The first three sets A, B and C show that HFE with weak keys can be broken for realistic choices of parameters when S, T are linear. Parameter set D (resp. E) is the original parameter set of [Pat96c] (resp. twice the original). All the experiments were run on one core of an Intel 2.3Ghz Xeon "Nehalem" computer with 74 Gbyte of RAM. We tested our attack on several sets of parameters described below. We forged the solution of the IP instance from the knowledge of the secret S and T.

19.4.1 Weak Keys

We first tested the attack on realistically-sized weak keys, corresponding to parameter sets A,B and C. The chosen parameters allows the encryption or signature of 256, 134 and 97 bits respectively. We choose the degree of the internal polynomial very conservatively (*i.e.*, much higher than what was proposed for the HFE challenges, and high enough to make decryption painfully slow). To make the QMLE part of the attack feasible, we choose the secret bijections S and T to be linear (as opposed to affine). Then solving the QMLE instance is a matter of seconds with the techniques presented chapter 17. We emphasize that none of the existing attack can be close to being practical on parameter sets A and B. On the other hand, parameter set C succumbs to the direct Gröbner basis attack (computing a Gröbner basis of the public key takes a bit less than a day).

19.4.2 Patarin's "Subfield" Variant of HFE

In order to reduce the size of the public key, Patarin suggested in [Pat96c] a "subfield" variant of HFE, in which the coefficients of the quadratic equations of $\mathbf{PK}$ live in a subfield $\Bbbk$ of $\mathbb{K}$. If $\mathbb{K} = \mathbb{F}_{256}$ and $\Bbbk = \mathbb{F}_2$, this reduces the size of the public key by a factor of 8. To achieve this, the coefficients of S and T, the coefficients of the defining polynomial of the extension field $\mathbb{L}$, and the coefficients of the internal polynomial $\mathbf{f}$ have to be chosen in $\Bbbk$ (instead of $\mathbb{K}$ or $\mathbb{L}$ for the latter). The linear masks S and T will be affine, so the IQMLE algorithms do not apply in this case.

In order for the reduction of the public key size to be effective, $\mathbb{K}$ has to be relatively big and $\Bbbk$ relatively small. The former implies that D cannot be very large, otherwise decryption is impractical, while the latter means little entropy in the internal polynomial. This opens a possible way of attack, consisting in guessing $\mathbf{f}$ and then solving the QMLE problem to recover S and T. We shall compare the attack presented in this paper with this simple one.

Patarin's "concrete proposal" is parameter set D in Table 19.1. For practical decryption, we have to choose $D = 2$ (yielding an internal polynomial of degree at most 131072), and decryption can take at most 4 minutes on our machine. The internal polynomial has at most 10 terms with coefficients in $\mathbb{F}_2$. The simple "guess-$\mathbf{f}$-then-QMLE" key recovery attack therefore needs to solve 2^{10} affine QMLE instances for which $q = 2$ and $n = 29$. Such instances are in fact tractable even with older techniques (though no one ever noticed it), for instance using the "to-and-fro" algorithm of [PGC98b]. In that case, the "guess-then-QMLE" attack has a workload of 2^{68}. With the new attack presented in this chapter, and the more advanced QMLE techniques described chapter 18, solving the QMLE instance takes about one second, and our attack takes less than one hour.

To show that the "subfield" variant is broken beyond repair, we show that it is possible to attack in practice parameters twice as big as the concrete proposal. This is parameter set E. The internal polynomial now has 21 terms, so the simple attack requires breaking 2^{21} affine instances of QMLE with $q = 2$ and $n = 59$. The exact workload required to break these instance is not precisely known, but we note that solving a system of 59 quadratic equations over $\mathbb{F}_2$ using the techniques described in chapter 11 would take about one month using inexpensive hardware. As was shown in §18.1, this is sometimes sufficient to solve a QMLE instance, otherwise Algorithm 18.4 has a smaller asymptotic complexity, but probably a greater practical complexity for such values of n. In any case, the "guess-then-QMLE" attack is here clearly impractical with a running-time of at least 2^{21} months. Our attack requires about one month to break the QMLE instance, plus about 4 hours for the remaining steps.

Parameter set	A	B	C	D	E
block size (bits)	256	134	97	232	236
q	256	4	2	256	16
n	32	67	97	29	59
$\deg \mathbf{f}$	131072	131072	128	131072	131072
coefficients of $\mathbf{f}$ in	$\mathbb{F}_{256}$	$\mathbb{F}_4$	$\mathbb{F}_2$	$\mathbb{F}_2$	$\mathbb{F}_2$
S and T	linear	linear	linear	affine	affine
coefficients of S, T in	$\mathbb{F}_{256}$	$\mathbb{F}_4$	$\mathbb{F}_2$	$\mathbb{F}_2$	$\mathbb{F}_2$
Terms in $\mathbf{f}$	10	54	29	10	21
size of **PK** (bits)	143'616	314'364	461'138	13'485	107'970
IP	polynomial			$\approx$ 1s	$\approx$ 4 weeks
Interpolation of $\mathbf{g}$ (once)	79s	30 min	140 min	51s	23min
Gröbner	7h	1 day	1 week	45s	3h
Variables / Equations	136 / 593	322/4947	423/10028	124 / 494	253 / 1889
Memory required	2.1Gbyte	45Gbyte	180Gbyte	350Mbyte	13.9Gbyte
Order Change	15s	30 min	4h	0s	30s

Table 19.1: Timings for the Attack

Conclusion

We have studied in practice the hardness of the MQ problem over $\mathbb{F}_2$. Algorithmic improvement and careful implementations on parallel platforms enables us to push further the boundary between intractable and feasible computation. We have shown for instance that solving a quadratic system of 64 equations over $\mathbb{F}_2$ would be possible in practice with a modest budget and readily available hardware. However, sightly increasing the parameters would make any existing method intractable, thus making MQ an interesting hardness assumption to build new schemes.

We have also studied in detail several relevant cases of the "Isomorphism of Polynomials" problem. In two cases (QFSE and IQMLE) we could design very practical algorithms, leading to the practical break of an identification scheme. Even in the hardest case, we have shown that it is possible to improve on exhaustive search in some cases.

Challenging the hardness these problems requires a rather wide-spectrum toolbox, from Gröbner bases to matrix pencils, finite vector spaces combinatorics, random trees and graph theory. This makes it a rather difficult but fun problem to work on. In our experience, inventing new algorithms was not obvious, but trying to understand their behavior and analyzing their running time and success probability was the most challenging part. The analysis of previous algorithms was often sketchy or heuristic as well.

We believe that the existence of a generic algorithm (Algorithm 18.4) faster than exhaustive search for the most difficult case of QMLE shows that constructing multivariate Trapdoor One-Way Function by obfuscating an easily-invertible function with linear masks is an unsound approach. Moreover, in actual schemes, the fact that one of the two polynomial maps is easily invertible is likely to give additional options to the attacker. As an illustration, many schemes relying on the hardness of a structured variant of QMLE have been broken. Random homogeneous instances of QMLE are hard to deal with, but it is difficult to build *efficient* encryption and/or signature schemes out of the corresponding problem.

Comparatively, schemes such as HFE or UOV, where the internal (easily-invertible) map is kept secret, seem to be in better shape, and no *ad hoc* QMLE algorithm applies (yet). It is noticeable that in the case of HFE, keys from a weak class leak some information which can be retrieved by solving an instance of QMLE, leading to a practical key-recovery attack.

Overall, it is still tempting to use the hardness of MQ to build new cryptographic schemes, but it is not obvious to come up with constructions that are *provably* secure under the hardness of MQ. Some preliminary proposals are appearing at the time of this writing, and this is an exciting research topic.

List of Figures

List of Tables

List of Algorithms

Bibliography

[AB10] Noga Alon and Eric Blais. Testing boolean function isomorphism. In Maria J. Serna, Ronen Shaltiel, Klaus Jansen, and José D. P. Rolim, editors, *APPROX-RANDOM*, volume 6302 of *Lecture Notes in Computer Science*, pages 394–405. Springer, 2010.

[ABD+10] Elena Andreeva, Charles Bouillaguet, Orr Dunkelman, Pierre-Alain Fouque, Jonathan J. Hoch, John Kelsey, Adi Shamir, and Sébastien Zimmer. New Second Preimage Attacks on Hash Functions. Submitted to the Journal of Cryptology., 2010.

[ABDK09] Elena Andreeva, Charles Bouillaguet, Orr Dunkelman, and John Kelsey. Herding, second preimage and trojan message attacks beyond merkle-damgård. In Michael J. Jacobson Jr., Vincent Rijmen, and Reihaneh Safavi-Naini, editors, *Selected Areas in Cryptography*, volume 5867 of *Lecture Notes in Computer Science*, pages 393–414. Springer, 2009.

[Abe10] Masayuki Abe, editor. *Advances in Cryptology - ASIACRYPT 2010 - 16th International Conference on the Theory and Application of Cryptology and Information Security, Singapore, December 5-9, 2010. Proceedings*, volume 6477 of *Lecture Notes in Computer Science*. Springer, 2010.

[ABF+08] Elena Andreeva, Charles Bouillaguet, Pierre-Alain Fouque, Jonathan J. Hoch, John Kelsey, Adi Shamir, and Sébastien Zimmer. Second Preimage Attacks on Dithered Hash Functions. In Smart [Sma08], pages 270–288.

[ABM+09] Jean-Philippe Aumasson, Eric Brier, Willi Meier, María Naya-Plasencia, and Thomas Peyrin. Inside the Hypercube. In Colin Boyd and Juan Manuel González Nieto, editors, *ACISP*, volume 5594 of *Lecture Notes in Computer Science*, pages 202–213. Springer, 2009.

[ADG+08] Luca Aceto, Ivan Damgård, Leslie Ann Goldberg, Magnús M. Halldórsson, Anna Ingólfsdóttir, and Igor Walukiewicz, editors. *Automata, Languages and Programming, 35th International Colloquium, ICALP 2008, Reykjavik, Iceland, July 7-11, 2008, Proceedings, Part II - Track B: Logic, Semantics, and Theory of Programming & Track C: Security and Cryptography Foundations*, volume 5126 of *Lecture Notes in Computer Science*. Springer, 2008.

[AFI+04] Gwénolé Ars, Jean-Charles Faugère, Hideki Imai, Mitsuru Kawazoe, and Makoto Sugita. Comparison Between XL and Gröbner Basis Algorithms. In Lee [Lee04], pages 338–353.

[AKS09] Roberto Maria Avanzi, Liam Keliher, and Francesco Sica, editors. *Selected Areas in Cryptography, 15th International Workshop, SAC 2008, Sackville, New Brunswick, Canada, August 14-15, Revised Selected Papers*, volume 5381 of *Lecture Notes in Computer Science*. Springer, 2009.

[Alb38] A. Adrian Albert. Symmetric and alternate matrices in an arbitrary field, i. *Transactions of the American Mathematical Society*, 43(3):386–436, 1938.

[All94] J.-P. Allouche. Sur la complexité des suites infinies. *Bull. Belg. Math. Soc.*, 1:133–143, 1994.

[AMM09] Jean-Philippe Aumasson, Atefeh Mashatan, and Willi Meier. More on Shabal's permutation. OFFICIAL COMMENT, 2009.

[AMW07] Carlisle M. Adams, Ali Miri, and Michael J. Wiener, editors. *Selected Areas in Cryptography, 14th International Workshop, SAC 2007, Ottawa, Canada, August 16-17, 2007, Revised Selected Papers*, volume 4876 of *Lecture Notes in Computer Science*. Springer, 2007.

[AN72] Krishna B. Athreya and Peter Ney. *Branching processes*. Springer-Verlag, Berlin, New York„ 1972.

[AP08] Elena Andreeva and Bart Preneel. A three-property-secure hash function. In Avanzi et al. [AKS09], pages 228–244.

[Bar04] Magali Bardet. *Étude des systèmes algébriques surdéterminés. Applications aux codes correcteurs et à la cryptographie.* PhD thesis, Université de Paris VI, 2004.

[BB04] Dan Boneh and Xavier Boyen. Efficient selective-id secure identity-based encryption without random oracles. In Christian Cachin and Jan Camenisch, editors, *EUROCRYPT*, volume 3027 of *Lecture Notes in Computer Science*, pages 223–238. Springer, 2004.

[BB06] Dan Boneh and Xavier Boyen. On the impossibility of efficiently combining collision resistant hash functions. In *Advances in Cryptology—CRYPTO 2006*, volume 4117 of *Lecture Notes in Computer Science*, pages 570–583. Berlin: Springer-Verlag, 2006. Available at `http://www.cs.stanford.edu/~xb/crypto06b/`.

[BBAC+05] Come Berbain, Olivier Billet, Nicolas Courtois Anne Canteaut, Henri Gilbert, Louis Goubin, Aline Gouget, Louis Granboulan, Cédric Lauradoux, Marine Minier, Thomas Pornin, , and Hervé Sibert. *Sosemanuk, a fast software-oriented stream cipher.* Submission to eStream, 2005.

[BBG07] Côme Berbain, Olivier Billet, and Henri Gilbert. Efficient implementations of multivariate quadratic systems. In Eli Biham and Amr Youssef, editors, *Selected Areas in Cryptography*, volume 4356 of *Lecture Notes in Computer Science*, pages 174–187. Springer Berlin / Heidelberg, 2007.

[BCBP03] Alex Biryukov, Christophe De Cannière, An Braeken, and Bart Preneel. A toolbox for cryptanalysis: Linear and affine equivalence algorithms. In *EUROCRYPT*, pages 33–50, 2003.

[BCC+09] Daniel J. Bernstein, Tien-Ren Chen, Chen-Mou Cheng, Tanja Lange, and Bo-Yin Yang. Ecm on graphics cards. In Joux [Jou09a], pages 483–501.

[BCC+10] Charles Bouillaguet, Hsieh-Chung Chen, Chen-Mou Cheng, Tung Chou, Ruben Niederhagen, Adi Shamir, and Bo-Yin Yang. Fast exhaustive search for polynomial systems in $\mathbb{F}_2$. In Stefan Mangard and François-Xavier Standaert, editors, *CHES*, volume 6225 of *Lecture Notes in Computer Science*, pages 203–218. Springer, 2010.

[BCCG92] Thierry Baritaud, Mireille Campana, Pascal Chauvaud, and Henri Gilbert. On the security of the permuted kernel identification scheme. In Ernest F. Brickell, editor, *CRYPTO*, volume 740 of *Lecture Notes in Computer Science*, pages 305–311. Springer, 1992.

[BCCM+09] Emmanuel Bresson, Anne Canteaut, Benoit Chevallier-Mames, Christophe Clavier, Thomas Fuhr, Aline Gouget, Thomas Icart, Jean-Francois Misarsky, Maria Naya-Plasencia, Pascal Paillier, Thomas Pornin, Jean-Rene Reinhard, Celine Thuillet, and Marion Videau. Indifferentiability with Distinguishers: Why Shabal Does Not Require Ideal Ciphers. Cryptology ePrint Archive, Report 2009/199, 2009. `http://eprint.iacr.org/`.

[BCD08] John Baena, Crystal Clough, and Jintai Ding. Square-vinegar signature scheme. In *PQCrypto '08: Proceedings of the 2nd International Workshop on Post-Quantum Cryptography*, pages 17–30, Berlin, Heidelberg, 2008. Springer-Verlag.

[BCJ+05] Eli Biham, Rafi Chen, Antoine Joux, Patrick Carribault, Christophe Lemuet, and William Jalby. Collisions of SHA-0 and Reduced SHA-1. In Cramer [Cra05], pages 36–57.

[BCP97] Wieb Bosma, John J. Cannon, and Catherine Playoust. The Magma Algebra System I: The User Language. *J. Symb. Comput.*, 24(3/4):235–265, 1997.

[BCS05] John Black, Martin Cochran, and Thomas Shrimpton. On the impossibility of highly-efficient blockcipher-based hash functions. In Cramer [Cra05], pages 526–541.

[BD07] Eli Biham and Orr Dunkelman. A framework for iterative hash functions - haifa. Cryptology ePrint Archive, Report 2007/278, 2007. `http://eprint.iacr.org/`.

[BDD+10] Charles Bouillaguet, Patrick Derbez, Orr Dunkelman, Nathan Keller, and Pierre-Alain Fouque. Low Data Complexity Attacks on AES. Cryptology ePrint Archive, Report 2010/633, 2010. Submitted to IEEE IT. Available at `http://eprint.iacr.org/`.

[BDF11] Charles Bouillaguet, Patrick Derbez, and Pierre-Alain Fouque. Automatic Search of Attacks on Round-reduced AES and Applications. In Rogaway [Rog11], pages 169–187.

[BDFL10] Charles Bouillaguet, Orr Dunkelman, Pierre-Alain Fouque, and Gaëtan Leurent. Another Look at the Complementation Property. In Seokhie Hong and Tetsu Iwata, editors, *FSE '10*, Lecture Notes in Computer Science. Springer, 2010.

[BDK07] Eli Biham, Orr Dunkelman, and Nathan Keller. Improved slide attacks. In Biryukov [Bir07], pages 153–166.

[BDK+10] Alex Biryukov, Orr Dunkelman, Nathan Keller, Dmitry Khovratovich, and Adi Shamir. Key recovery attacks of practical complexity on aes-256 variants with up to 10 rounds. In Gilbert [Gil10], pages 299–319.

[BDLF10] Charles Bouillaguet, Orr Dunkelman, Gaëtan Leurent, and Pierre-Alain Fouque. Attacks on hash functions based on generalized feistel: Application to reduced-round Lesamnta and SHAvite-3_{512}. In Biryukov et al. [BGS11], pages 18–35.

[BDLF11] Charles Bouillaguet, Orr Dunkelman, Gaëtan Leurent, and Pierre-Alain Fouque. New insights on impossible differentials. In *Selected Areas in Cryptography*, August 2011.

[BDPA08] Guido Bertoni, Joan Daemen, Michael Peeters, and Gilles Van Assche. On the indifferentiability of the sponge construction. In Smart [Sma08], pages 181–197.

[BDPA09] Guido Bertoni, Joan Daemen, Michael Peeters, and Gilles Van Assche. Sufficient conditions for sound tree and sequential hashing modes. Cryptology ePrint Archive, Report 2009/210, 2009. `http://eprint.iacr.org/`.

[Bel10] Jordan Bell. A summary of euler's work on the pentagonal number theorem. *Archive for History of Exact Sciences*, 64:301–373, 2010. 10.1007/s00407-010-0057-y.

[Ben19] A. A. Bennett. Products of skew-symmetric matrices. *American M. S. Bull.*, 25:455–458, 1919.

[Ber08] Daniel J. Bernstein. Proving tight security for rabin-williams signatures. In Smart [Sma08], pages 70–87.

[BF05] Alexandra Boldyreva and Marc Fischlin. Analysis of random oracle instantiation scenarios for oaep and other practical schemes. In Shoup [Sho05], pages 412–429.

[BF08] Charles Bouillaguet and Pierre-Alain Fouque. Analysis of the collision resistance of radiogatún using algebraic techniques. In Avanzi et al. [AKS09], pages 245–261.

[BFFP11] Charles Bouillaguet, Jean-Charles Faugère, Pierre-Alain Fouque, and Ludovic Perret. Practical cryptanalysis of the identification scheme based on the isomorphism of polynomial with one secret problem. In Dario Catalano, Nelly Fazio, Rosario Gennaro, and Antonio Nicolosi, editors, *Public Key Cryptography*, volume 6571 of *Lecture Notes in Computer Science*, pages 473–493. Springer, 2011.

[BFJT09] Charles Bouillaguet, Pierre-Alain Fouque, Antoine Joux, and Joana Treger. A family of weak keys in hfe (and the corresponding practical key-recovery). Submitted to the Journal of Mathematical Cryptology., 2009. `http://eprint.iacr.org/2009/619`.

[BFL10] Charles Bouillaguet, Pierre-Alain Fouque, and Gaëtan Leurent. Security analysis of simd. In Biryukov et al. [BGS11], pages 351–368.

[BFMR11] Charles Bouillaguet, Pierre-Alain Fouque, and Gilles Macario-Rat. Practical key-recovery for all possible parameters of sflash. Submitted to Asiacrypt 2011., 2011. `http://eprint.iacr.org/2011/271`.

[BFP08] Luk Bettale, Jean-Charles Faugère, and Ludovic Perret. Cryptanalysis of the trms signature scheme of pkc'05. In Serge Vaudenay, editor, *AFRICACRYPT*, volume 5023 of *Lecture Notes in Computer Science*, pages 143–155. Springer, 2008.

[BFS99] Jonathan F. Buss, Gudmund Skovbjerg Frandsen, and Jeffrey Shallit. The Computational Complexity of Some Problems of Linear Algebra. *J. Comput. Syst. Sci.*, 58(3):572–596, 1999.

[BFSY05] Magali Bardet, Jean-Charles Faugère, Bruno Salvy, and Bo-Yin Yang. Asymptotic Behaviour of the Degree of Regularity of Semi-Regular Polynomial Systems. In *Proc. of MEGA 2005, Eighth International Symposium on Effective Methods in Algebraic Geometry*, 2005. Eighth International Symposium on Effective Methods in Algebraic Geometry, Porto Conte, Alghero, Sardinia (Italy), May 27th – June 1st.

[BFZ10] Charles Bouillaguet, Pierre-Alain Fouque, and Sébastien Zimmer. On the resistance of practical hash functions constructions to generic second preimage attacks. Submitted to the Information Processing Letters., 2010.

[BG75] E. A. Bender and J. R. Goldman. On the applications of mobius inversion in combinatorial analysis. *The American Mathematical Monthly*, 82(8):pp. 789–803, 1975.

[BG03] Olivier Billet and Henri Gilbert. A traceable block cipher. In Chi-Sung Laih, editor, *ASIACRYPT*, volume 2894 of *Lecture Notes in Computer Science*, pages 331–346. Springer, 2003.

[BGH07] Dan Boneh, Craig Gentry, and Michael Hamburg. Space-efficient identity based encryption without pairings. In *FOCS*, pages 647–657. IEEE Computer Society, 2007.

[BGP06] Côme Berbain, Henri Gilbert, and Jacques Patarin. QUAD: A practical stream cipher with provable security. In Dwork [Dwo06], pages 109–128.

[BGS11] Alex Biryukov, Guang Gong, and Douglas R. Stinson, editors. *Selected Areas in Cryptography - 17th International Workshop, SAC 2010, Waterloo, Ontario, Canada, August 12-13, 2010, Revised Selected Papers*, volume 6544 of *Lecture Notes in Computer Science*. Springer, 2011.

[Bir05] Alex Biryukov. A New 128-bit Key Stream Cipher LEX. ECRYPT stream cipher project report 2005/013, 2005. `http://www.ecrypt.eu.org/stream`.

[Bir06a] Alex Biryukov. The Design of a Stream Cipher LEX. In Biham and Youssef [BY07], pages 67–75.

[Bir06b] Alex Biryukov. The Tweak for LEX-128, LEX-192,LEX-256. ECRYPT stream cipher project report 2006/037, 2006. `http://www.ecrypt.eu.org/stream`.

[Bir07] Alex Biryukov, editor. *Fast Software Encryption, 14th International Workshop, FSE 2007, Luxembourg, Luxembourg, March 26-28, 2007, Revised Selected Papers*, volume 4593 of *Lecture Notes in Computer Science*. Springer, 2007.

[BK07] Alex Biryukov and Dmitry Khovratovich. Two New Techniques of Side-Channel Cryptanalysis. In Pascal Paillier and Ingrid Verbauwhede, editors, *CHES*, volume 4727 of *Lecture Notes in Computer Science*, pages 195–208. Springer, 2007.

[BK09] Alex Biryukov and Dmitry Khovratovich. Related-Key Cryptanalysis of the Full AES-192 and AES-256. In Matsui [Mat09], pages 1–18.

[BKN09] Alex Biryukov, Dmitry Khovratovich, and Ivica Nikolic. Distinguisher and Related-Key Attack on the Full AES-256. In Halevi [Hal09], pages 231–249.

[BKW93] T. Becker, H. Kredel, and V. Weispfenning. *Gröbner bases: a computational approach to commutative algebra*. Springer-Verlag, London, UK, 0 edition, 4 1993.

[BMR09] Olivier Billet and Gilles Macario-Rat. Cryptanalysis of the square cryptosystems. In Matsui [Mat09], pages 451–468.

[BN10] Alex Biryukov and Ivica Nikolic. Automatic Search for Related-Key Differential Characteristics in Byte-Oriented Block Ciphers: Application to AES, Camellia, Khazad and Others. In Gilbert [Gil10], pages 322–344.

[Bon03] Dan Boneh, editor. *Advances in Cryptology - CRYPTO 2003, 23rd Annual International Cryptology Conference, Santa Barbara, California, USA, August 17-21, 2003, Proceedings*, volume 2729 of *Lecture Notes in Computer Science*. Springer, 2003.

[BPW06] Johannes Buchmann, Andrei Pyshkin, and Ralf-Philipp Weinmann. A Zero-Dimensional Gröbner Basis for AES-128. In Robshaw [Rob06], pages 78–88.

[BR93] Mihir Bellare and Phillip Rogaway. Random Oracles are Practical: A Paradigm for Designing Efficient Protocols. In *ACM Conference on Computer and Communications Security*, pages 62–73, 1993.

[BR94] Mihir Bellare and Phillip Rogaway. Optimal Asymmetric Encryption. In Alfredo De Santis, editor, *EUROCRYPT*, volume 950 of *Lecture Notes in Computer Science*, pages 92–111. Springer, 1994.

[BR96] Mihir Bellare and Phillip Rogaway. The Exact Security of Digital Signatures - How to Sign with RSA and Rabin. In *EUROCRYPT*, pages 399–416, 1996.

[BR97] Mihir Bellare and Phillip Rogaway. Collision-Resistant Hashing: Towards Making UOWHFs Practical. In Jr. [Jr.97], pages 470–484.

[Bra90] Gilles Brassard, editor. *CRYPTO '89, Santa Barbara, California, USA, August0-24, 1989, Proceedings*, volume 435 of *Lecture Notes in Computer Science*. Springer, 1990.

[BRP07] Olivier Billet, Matthew J. B. Robshaw, and Thomas Peyrin. On building hash functions from multivariate quadratic equations. In Josef Pieprzyk, Hossein Ghodosi, and Ed Dawson, editors, *ACISP*, volume 4586 of *Lecture Notes in Computer Science*, pages 82–95. Springer, 2007.

[BRS02] John Black, Phillip Rogaway, and Thomas Shrimpton. Black-Box Analysis of the Block-Cipher-Based Hash-Function Constructions from PGV. In Yung [Yun02], pages 320–335.

[BRSS10] John Black, Phillip Rogaway, Thomas Shrimpton, and Martijn Stam. An analysis of the blockcipher-based hash functions from pgv. *J. Cryptology*, 23(4):519–545, 2010.

[BS00] Alex Biryukov and Adi Shamir. Cryptanalytic Time/Memory/Data Tradeoffs for Stream Ciphers. In Tatsuaki Okamoto, editor, *ASIACRYPT*, volume 1976 of *Lecture Notes in Computer Science*, pages 1–13. Springer, 2000.

[BSU11] Simon Blackburn, Douglas Stinson, and Jalaj Upadhyay. On the complexity of the herding attack and some related attacks on hash functions. *Designs, Codes and Cryptography*, pages 1–23, 2011. 10.1007/s10623-010-9481-x.

[Buc65] Bruno Buchberger. *Ein Algorithmus zum Auffinden der Basiselemente des Restklassenringes nach einem nulldimensionalen Polynomideal*. PhD thesis, University of Innsbruck, 1965.

[BW99] Alex Biryukov and David Wagner. Slide attacks. In Lars R. Knudsen, editor, *FSE*, volume 1636 of *Lecture Notes in Computer Science*, pages 245–259. Springer, 1999.

[BW00] Alex Biryukov and David Wagner. Advanced slide attacks. In Preneel [Pre00], pages 589–606.

[BWP05] An Braeken, Christopher Wolf, and Bart Preneel. A Study of the Security of Unbalanced Oil and Vinegar Signature Schemes. In Alfred Menezes, editor, *CT-RSA*, volume 3376 of *Lecture Notes in Computer Science*, pages 29–43. Springer, 2005.

[BY07] Eli Biham and Amr M. Youssef, editors. *Selected Areas in Cryptography, 13th International Workshop, SAC 2006, Montreal, Canada, August 17-18, 2006 Revised Selected Papers*, volume 4356 of *Lecture Notes in Computer Science*. Springer, 2007.

[Can97] Ran Canetti. Towards realizing random oracles: Hash functions that hide all partial information. In Jr. [Jr.97], pages 455–469.

[CB07] Nicolas Courtois and Gregory V. Bard. Algebraic cryptanalysis of the data encryption standard. In Steven D. Galbraith, editor, *IMA Int. Conf.*, volume 4887 of *Lecture Notes in Computer Science*, pages 152–169. Springer, 2007.

[CBD+09] Crystal Clough, John Baena, Jintai Ding, Bo-Yin Yang, and Ming-Shing Chen. Square, a new multivariate encryption scheme. In Fischlin [Fis09], pages 252–264.

[CBW08] Nicolas Courtois, Gregory V. Bard, and David Wagner. Algebraic and slide attacks on keeloq. In Nyberg [Nyb08], pages 97–115.

[CDMP05] Jean-Sébastien Coron, Yevgeniy Dodis, Cécile Malinaud, and Prashant Puniya. Merkle-Damgård Revisited: How to Construct a Hash Function. In *CRYPTO'05*, pages 430–448, 2005.

[CFPZ09] Céline Chevalier, Pierre-Alain Fouque, David Pointcheval, and Sébastien Zimmer. Optimal randomness extraction from a diffie-hellman element. In Joux [Jou09a], pages 572–589.

[CGH04] Ran Canetti, Oded Goldreich, and Shai Halevi. The random oracle methodology, revisited. *J. ACM*, 51(4):557–594, 2004.

[Che35] Claude Chevalley. Démonstration d'une hypothèse de M. Artin. *Abh. Math. Semin. Hamb. Univ.*, 11:73–75, 1935.

[Cid04] Carlos Cid. Some Algebraic Aspects of the Advanced Encryption Standard. In Dobbertin et al. [DRS05], pages 58–66.

[CJ98] Florent Chabaud and Antoine Joux. Differential Collisions in SHA-0. In Krawczyk [Kra98], pages 56–71.

[CKK+01] Jung Hee Cheon, MunJu Kim, Kwangjo Kim, Jung-Yeun Lee, and SungWoo Kang. Improved impossible differential cryptanalysis of rijndael and crypton. In Kim [Kim02], pages 39–49.

[CKM97] Stéphane Collart, Michael Kalkbrener, and Daniel Mall. Converting bases with the gröbner walk. *J. Symb. Comput.*, 24(3/4):465–469, 1997.

[CKPS00] Nicolas Courtois, Alexander Klimov, Jacques Patarin, and Adi Shamir. Efficient Algorithms for Solving Overdefined Systems of Multivariate Polynomial Equations. In *EUROCRYPT*, pages 392–407, 2000.

[CL05] Carlos Cid and Gaëtan Leurent. An analysis of the xsl algorithm. In Roy [Roy05], pages 333–352.

[CLO91] David A. Cox, John Little, and Donal O'Shea. *Ideals, Varieties, and Algorithms: An Introduction to Computational Algebraic Geometry and Commutative Algebra, (Undergraduate Texts in Mathematics)*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 1991.

[CM03] Nicolas Courtois and Willi Meier. Algebraic attacks on stream ciphers with linear feedback. In Eli Biham, editor, *EUROCRYPT*, volume 2656 of *Lecture Notes in Computer Science*, pages 345–359. Springer, 2003.

[CN08] Donghoon Chang and Mridul Nandi. Improved Indifferentiability Security Analysis of chopMD Hash Function. In Nyberg [Nyb08], pages 429–443.

[Cob72] Alan Cobham. Uniform Tag Seqences. *Mathematical Systems Theory*, 6(3):164–192, 1972.

[Coo00] C. Cooper. On the distribution of rank of a random matrix over a finite field. In *Proceedings of the ninth international conference on on Random structures and algorithms*, pages 197–212, New York, NY, USA, 2000. John Wiley & Sons, Inc.

[COQ09] Nicolas Courtois, Sean O'Neil, and Jean-Jacques Quisquater. Practical algebraic attacks on the hitag2 stream cipher. In Pierangela Samarati, Moti Yung, Fabio Martinelli, and Claudio Agostino Ardagna, editors, *ISC*, volume 5735 of *Lecture Notes in Computer Science*, pages 167–176. Springer, 2009.

[Cor02a] Jean-Sébastien Coron. Optimal security proofs for pss and other signature schemes. In Knudsen [Knu02], pages 272–287.

[Cor02b] Jean-Sébastien Coron. Security proof for partial-domain hash signature schemes. In Yung [Yun02], pages 613–626.

[Cou01] Nicolas Courtois. Efficient zero-knowledge authentication based on a linear algebra problem minrank. In Colin Boyd, editor, *ASIACRYPT*, volume 2248 of *Lecture Notes in Computer Science*, pages 402–421. Springer, 2001.

[Cou02] Nicolas Courtois. Higher order correlation attacks, xl algorithm and cryptanalysis of toyocrypt. In Pil Joong Lee and Chae Hoon Lim, editors, *ICISC*, volume 2587 of *Lecture Notes in Computer Science*, pages 182–199. Springer, 2002.

[Cou03] Nicolas Courtois. Fast algebraic attacks on stream ciphers with linear feedback. In Boneh [Bon03], pages 176–194.

[Cou04] Nicolas Courtois. General principles of algebraic attacks and new design criteria for cipher components. In Dobbertin et al. [DRS05], pages 67–83.

[CP02] Nicolas Courtois and Josef Pieprzyk. Cryptanalysis of block ciphers with overdefined systems of equations. In Zheng [Zhe02], pages 267–287.

[CR92] William Y. C. Chen and Gian-Carlo Rota. q-analogs of the inclusion- exclusion principle and permutations with restricted position. *Discrete Mathematics*, 104(1):7 – 22, 1992.

[CR06] Christophe De Cannière and Christian Rechberger. Finding sha-1 characteristics: General results and applications. In Lai and Chen [LC06], pages 1–20.

[Cra05] Ronald Cramer, editor. *Advances in Cryptology - EUROCRYPT 2005, 24th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Aarhus, Denmark, May 22-26, 2005, Proceedings*, volume 3494 of *Lecture Notes in Computer Science*. Springer, 2005.

[Cra08] Ronald Cramer, editor. *Public Key Cryptography - PKC 2008, 11th International Workshop on Practice and Theory in Public-Key Cryptography, Barcelona, Spain, March 9-12, 2008. Proceedings*, volume 4939 of *Lecture Notes in Computer Science*. Springer, 2008.

[DA99] T. Dierks and C. Allen. The MD5 message-digest algorithm. RFC2246, Januaryl 1999.

[Dam89] Ivan Damgård. A Design Principle for Hash Functions. In Brassard [Bra90], pages 416–427.

[Dav06] Timothy A. Davis. *Direct Methods for Sparse Linear Systems (Fundamentals of Algorithms 2)*. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 2006.

[dB61] N.G. de Bruijn. *Asymptotic methods in analysis. 2nd edition.* Bibliotheca Mathematica. Vol. 4. Groningen: P. Noordhoff Ltd. XII, 200 p. , 1961.

[dBB93] Bert den Boer and Antoon Bosselaers. Collisions for the Compression Function of MD5. In *EUROCRYPT*, pages 293–304, 1993.

[dCMR07] Christophe de Cannière, Florian Mendel, and Christian Rechberger. Collisions for 70-Step SHA-1: On the Full Cost of Collision Search. In Adams et al. [AMW07], pages 56–73.

[dCR06] Christophe de Cannière and Christian Rechberger. Finding SHA-1 Characteristics: General Results and Applications. In Lai and Chen [LC06], pages 1–20.

[dCR08] Christophe de Cannière and Christian Rechberger. Preimages for Reduced SHA-0 and SHA-1. In David Wagner, editor, *CRYPTO*, volume 5157 of *Lecture Notes in Computer Science*, pages 179–202. Springer, 2008.

[DDA11] Joan Daemen, Tony Dusenge, and Gilles Van Assche. Sufficient conditions for sound hashing using a truncated permutation. Cryptology ePrint Archive, Report 2011/459, 2011. `http://eprint.iacr.org/`.

[Dea99] Richard Drews Dean. *Formal Aspects of Mobile Code Security*. PhD thesis, Princeton University, January 1999.

[Der10] Patrick Derbez. Rapport de Stage. Technical report, Ecole Normale Supérieure, Paris, September 2010.

[DFS07] Vivien Dubois, Pierre-Alain Fouque, and Jacques Stern. Cryptanalysis of SFLASH with Slightly Modified Parameters. In *EUROCRYPT*, volume 4515, pages 264–275. Springer, 2007.

[DFSS07] Vivien Dubois, Pierre-Alain Fouque, Adi Shamir, and Jacques Stern. Practical Cryptanalysis of SFLASH. In *CRYPTO*, volume 4622, pages 1–12. Springer, 2007.

[DG10a] Jeffrey Dean and Sanjay Ghemawat. Mapreduce: a flexible data processing tool. *Commun. ACM*, 53(1):72–77, 2010.

[DG10b] Vivien Dubois and Nicolas Gama. The degree of regularity of hfe systems. In Abe [Abe10], pages 557–576.

[DGS06] Vivien Dubois, Louis Granboulan, and Jacques Stern. An efficient provable distinguisher for hfe. In Michele Bugliesi, Bart Preneel, Vladimiro Sassone, and Ingo Wegener, editors, *ICALP (2)*, volume 4052 of *Lecture Notes in Computer Science*, pages 156–167. Springer, 2006.

[Die04] Claus Diem. The xl-algorithm and a conjecture from commutative algebra. In Lee [Lee04], pages 323–337.

[DK08] Orr Dunkelman and Nathan Keller. A New Attack on the LEX Stream Cipher. In Pieprzyk [Pie08], pages 539–556.

[DK10a] Orr Dunkelman and Nathan Keller. Cryptanalysis of the Stream Cipher LEX, 2010. Available at `http://www.ma.huji.ac.il/~nkeller/Crypt-jour-LEX.pdf`.

[DK10b] Orr Dunkelman and Nathan Keller. The effects of the omission of last round's mixcolumns on aes. *Inf. Process. Lett.*, 110(8-9):304–308, 2010.

[DKR97] Joan Daemen, Lars R. Knudsen, and Vincent Rijmen. The block cipher square. In Eli Biham, editor, *FSE*, volume 1267 of *Lecture Notes in Computer Science*, pages 149–165. Springer, 1997.

[DKS10a] Orr Dunkelman, Nathan Keller, and Adi Shamir. Improved single-key attacks on 8-round aes-192 and aes-256. In Abe [Abe10], pages 158–176.

[DKS10b] Orr Dunkelman, Nathan Keller, and Adi Shamir. A practical-time related-key attack on the kasumi cryptosystem used in gsm and 3g telephony. In Tal Rabin, editor, *CRYPTO*, volume 6223 of *Lecture Notes in Computer Science*, pages 393–410. Springer, 2010.

[DKS11] Orr Dunkelman, Nathan Keller, and Adi Shamir. Alred blues: New attacks on aes-based mac's. Cryptology ePrint Archive, Report 2011/095, 2011. `http://eprint.iacr.org/`.

[DO68] Peter Dembowski and T. G. Ostrom. Planes of Order n with Collineation Groups of Order n^2. *Mathematische Zeitschrift*, 103(3):239–258, 1968.

[Dob96] Hand Dobbertin. Collision on the MD5 Compression Function. Rump Session of EUROCRYPT'96, 1996. `http://www-cse.ucsd.edu/users/bsy/dobbertin.ps`.

[Dob98] Hans Dobbertin. Cryptanalysis of MD4. *J. Cryptology*, 11(4):253–271, 1998.

[DOP05] Yevgeniy Dodis, Roberto Oliveira, and Krzysztof Pietrzak. On the generic insecurity of the full domain hash. In Shoup [Sho05], pages 449–466.

[DR05a] Joan Daemen and Vincent Rijmen. A New MAC Construction ALRED and a Specific Instance ALPHA-MAC. In Henri Gilbert and Helena Handschuh, editors, *FSE*, volume 3557 of *Lecture Notes in Computer Science*, pages 1–17. Springer, 2005.

[DR05b] Joan Daemen and Vincent Rijmen. The Pelican MAC Function. Cryptology ePrint Archive, Report 2005/088, 2005. `http://eprint.iacr.org/`.

[DRS05] Hans Dobbertin, Vincent Rijmen, and Aleksandra Sowa, editors. *Advanced Encryption Standard - AES, 4th International Conference, AES 2004, Bonn, Germany, May 10-12, 2004, Revised Selected and Invited Papers*, volume 3373 of *Lecture Notes in Computer Science*. Springer, 2005.

[DS08] Hüseyin Demirci and Ali Aydin Selçuk. A meet-in-the-middle attack on 8-round aes. In Nyberg [Nyb08], pages 116–126.

[DS09] Itai Dinur and Adi Shamir. Side channel cube attacks on block ciphers. Cryptology ePrint Archive, Report 2009/127, 2009. `http://eprint.iacr.org/`.

[DS10] Itai Dinur and Adi Shamir. An improved algebraic attack on hamsi-256. Cryptology ePrint Archive, Report 2010/602, 2010. `http://eprint.iacr.org/`, to appear at FSE'11.

[Dub07] Vivien Dubois. *Cryptanalyse de Schémas Multivariés*. PhD thesis, Ecole Normale Supérieure, Paris, France, 2007.

[Dun09] Orr Dunkelman, editor. *Fast Software Encryption, 16th International Workshop, FSE 2009, Leuven, Belgium, February 22-25, 2009, Revised Selected Papers*, volume 5665 of *Lecture Notes in Computer Science*. Springer, 2009.

[dVP03] Françoise Levy dit Vehel and Ludovic Perret. Polynomial Equivalence Problems and Applications to Multivariate Cryptosystems. In Thomas Johansson and Subhamoy Maitra, editors, *INDOCRYPT*, volume 2904 of *Lecture Notes in Computer Science*, pages 235–251. Springer, 2003.

[Dwo06] Cynthia Dwork, editor. *Advances in Cryptology - CRYPTO 2006, 26th Annual International Cryptology Conference, Santa Barbara, California, USA, August 20-24, 2006, Proceedings*, volume 4117 of *Lecture Notes in Computer Science*. Springer, 2006.

[DWY07] Jintai Ding, Christopher Wolf, and Bo-Yin Yang. -invertible cycles for multivariate quadratic public key cryptographyℓ. In Tatsuaki Okamoto and Xiaoyun Wang, editors, *Public Key Cryptography*, volume 4450 of *Lecture Notes in Computer Science*, pages 266–281. Springer, 2007.

[Eli06] Orr Dunkelman Eli Biham. A Framework for Iterative Hash Functions — HAIFA. Presented at the second NIST hash workshop, August 24–25 2006.

[ELR75] Andrzej Ehrenfeucht, K. P. Lee, and Grzegorz Rozenberg. Subword Complexities of Various Classes of Deterministic Developmental Languages without Interactions. *Theor. Comput. Sci.*, 1(1):59–75, 1975.

[Fau99] Jean-Charles Faugère. A new efficient algorithm for computing Gröbner bases (F4). *Journal of Pure and Applied Algebra*, 139(1-3):61–88, June 1999.

[Fau02] Jean-Charles Faugère. A New Efficient Algorithm for Computing Gröbner Bases Without Reduction to Zero (F5). In T. Mora, editor, *ISSAC '02: Proceedings of the 2002 International Symposium on Symbolic and Algebraic Computation*, pages 75–83, New York, NY, USA, July 2002. ACM Press. isbn: 1-58113-484-3.

[FD85] Harriet J. Fell and Whitfield Diffie. Analysis of a public key approach based on polynomial substitution. In Hugh C. Williams, editor, *CRYPTO*, volume 218 of *Lecture Notes in Computer Science*, pages 340–349. Springer, 1985.

[FDS11] Jean-Charles Faugère, Mohab Safey El Din, and Pierre-Jean Spaenlehauer. Gröbner bases of bihomogeneous ideals generated by polynomials of bidegree (1, 1): Algorithms and complexity. *J. Symb. Comput.*, 46(4):406–437, 2011.

[Fel71] W. Feller. *An Introduction to Probability Theory and Its Applications*, volume 1, chapter 12. John Wiley & Sons, 1971.

[FGLM93] Jean-Charles Faugère, Patrizia M. Gianni, Daniel Lazard, and Teo Mora. Efficient Computation of Zero-Dimensional Gröbner Bases by Change of Ordering. *Journal of Symbolic Computation*, 16(4):329–344, 1993.

[Fis09] Marc Fischlin, editor. *Topics in Cryptology - CT-RSA 2009, The Cryptographers' Track at the RSA Conference 2009, San Francisco, CA, USA, April 20-24, 2009. Proceedings*, volume 5473 of *Lecture Notes in Computer Science*. Springer, 2009.

[FJ03] Jean-Charles Faugère and Antoine Joux. Algebraic Cryptanalysis of Hidden Field Equation (HFE) Cryptosystems Using Gröbner Bases. In Boneh [Bon03], pages 44–60.

[FJPT10] Jean-Charles Faugère, Antoine Joux, Ludovic Perret, and Joana Treger. Cryptanalysis of the hidden matrix cryptosystem. In Michel Abdalla and Paulo S. L. M. Barreto, editors, *LATINCRYPT*, volume 6212 of *Lecture Notes in Computer Science*, pages 241–254. Springer, 2010.

[FK11] Jean-Christophe Filliâtre and K. Kalyanasundaram. Functory: A Distributed Computing Libraryfor Objective Caml. In *Trends in Functional Programming*, Madrid, Spain, May 2011.

[FKL+00] Niels Ferguson, John Kelsey, Stefan Lucks, Bruce Schneier, Michael Stay, David Wagner, and Doug Whiting. Improved cryptanalysis of rijndael. In Bruce Schneier, editor, *FSE*, volume 1978 of *Lecture Notes in Computer Science*, pages 213–230. Springer, 2000.

[FL07] Marc Fischlin and Anja Lehmann. Security-amplifying combiners for collision-resistant hash functions. In Menezes [Men07], pages 224–243.

[FLM10] Niels Ferguson, Stefan Lucks, and Kerry A. McKay. Symmetric States and their Structure: Improved Analysis of CubeHash. Cryptology ePrint Archive, Report 2010/273, 2010. `http://eprint.iacr.org/`.

[FLN07] Pierre-Alain Fouque, Gaëtan Leurent, and Phong Q. Nguyen. Full key-recovery attacks on hmac/nmac-md4 and nmac-md5. In Menezes [Men07], pages 13–30.

[FLPR99] Matteo Frigo, Charles E. Leiserson, Harald Prokop, and Sridhar Ramachandran. Cache-oblivious algorithms. In *FOCS*, pages 285–298, 1999.

[FMRPS08] Pierre-Alain Fouque, Gilles Macario-Rat, Ludovic Perret, and Jacques Stern. Total break of the ℓ-ic signature scheme. In Cramer [Cra08], pages 1–17.

[FMRS08] Pierre-Alain Fouque, Gilles Macario-Rat, and Jacques Stern. Key Recovery on Hidden Monomial Multivariate Schemes. In Smart [Sma08], pages 19–30.

[Fog10] Agner Fog. *Instruction Tables*. Copenhagen University, College of Engineering, Feb 2010. Lists of Instruction Latencies, Throughputs and micro-operation breakdowns for Intel, AMD, and VIA CPUs, `http://www.agner.org/optimize/instruction_tables.pdf`.

[For96] Scott Fortin. The graph isomorphism problem. Technical report, University of Alberta, 1996.

[FP06] Jean-Charles Faugère and Ludovic Perret. Polynomial Equivalence Problems: Algorithmic and Theoretical Aspects. In Vaudenay [Vau06], pages 30–47.

[FP09] Thomas Fuhr and Thomas Peyrin. Cryptanalysis of radiogatún. In Dunkelman [Dun09], pages 122–138.

[Frö85] Ralf Fröberg. An inequality for hilbert series of graded algebras. *Math. Scand.*, 56:117–144, 1985.

[FS86] Amos Fiat and Adi Shamir. How to prove yourself: Practical solutions to identification and signature problems. In Andrew M. Odlyzko, editor, *CRYPTO*, volume 263 of *Lecture Notes in Computer Science*, pages 186–194. Springer, 1986.

[Ful01] Jason Fulman. Random matrix theory over finite fields. *Bull. Amer. Math. Soc. (N.S*, 39:51–85, 2001.

[Fur01] Soichi Furuya. Slide attacks with a known-plaintext cryptanalysis. In Kim [Kim02], pages 214–225.

[Gam84] Taher El Gamal. A public key cryptosystem and a signature scheme based on discrete logarithms. In *CRYPTO*, pages 10–18, 1984.

[Gan59] F. R. Gantmacher. *The Theory of Matrices*. Chelsea Publishing Company, New York, NY, USA, 1959.

[Gei99] Jochen Geiger. Elementary new proofs of classical limit theorems for Galton-Watson processes. *J. Appl. Probab.*, 36(2):301–309, 1999.

[Geo92] J. Georgiades. Some remarks on the security of the identification scheme based on permuted kernels. *J. Cryptology*, 5(2):133–137, 1992.

[Gil10] Henri Gilbert, editor. *Advances in Cryptology - EUROCRYPT 2010, 29th Annual International Conference on the Theory and Applications of Cryptographic Techniques, French Riviera, May 30 - June 3, 2010. Proceedings*, volume 6110 of *Lecture Notes in Computer Science*. Springer, 2010.

[GJ79] M. R. Garey and D. S. Johnson. *Computers and Intractability, A Guide to the Theory of* NP-*Completeness*. Freeman, New-York, 1979.

[GJS06] Louis Granboulan, Antoine Joux, and Jacques Stern. Inverting HFE Is Quasipolynomial. In Dwork [Dwo06], pages 345–356.

[GM02] Henri Gilbert and Marine Minier. Cryptanalysis of SFLASH. In Knudsen [Knu02], pages 288–298.

[GMK08] Danilo Gligoroski, Smile Markovski, and Svein Johan Knapskog. Multivariate quadratic trapdoor functions based on multivariate quadratic quasigroups. In *Proceedings of the American Conference on Applied Mathematics*, pages 44–49, Stevens Point, Wisconsin, USA, 2008. World Scientific and Engineering Academy and Society (WSEAS).

[GMR85] Shafi Goldwasser, Silvio Micali, and Charles Rackoff. The knowledge complexity of interactive proof-systems (extended abstract). In *STOC*, pages 291–304. ACM, 1985.

[GMS03] Willi Geiselmann, Willi Meier, and Rainer Steinwandt. An Attack on the Isomorphisms of Polynomials Problem with One Secret. *Int. J. Inf. Sec.*, 2(1):59–64, 2003.

[GMW86] Oded Goldreich, Silvio Micali, and Avi Wigderson. Proofs that yield nothing but their validity and a methodology of cryptographic protocol design (extended abstract). In *FOCS*, pages 174–187. IEEE, 1986.

[GPV08] Craig Gentry, Chris Peikert, and Vinod Vaikuntanathan. Trapdoors for hard lattices and new cryptographic constructions. In Cynthia Dwork, editor, *STOC*, pages 197–206. ACM, 2008.

[GR69] J Goldman and G-C Rota. The number of subspaces of a vector space. In *in "Recent Progress in Combinatorics" (W. T. Tutte Ed*, pages 75–84. Academic Press, 1969.

[GSB01] Willi Geiselmann, Rainer Steinwandt, and Thomas Beth. Attacking the Affine Parts of SFLASH. In Bahram Honary, editor, *IMA Int. Conf.*, volume 2260 of *Lecture Notes in Computer Science*, pages 355–359. Springer, 2001.

[Hal09] Shai Halevi, editor. *Advances in Cryptology - CRYPTO 2009, 29th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 16-20, 2009. Proceedings*, volume 5677 of *Lecture Notes in Computer Science*. Springer, 2009.

[Hel80] Martin E. Hellman. A cryptanalytic time-memory trade off. In *IEEE Transactions on Information Theory*, volume IT-26, pages 401–406, 1980.

[HJ11] Martin Hell and Thomas Johansson. Breaking the stream ciphers f-fcsr-h and f-fcsr-16 in real time. *J. Cryptology*, 24(3):427–445, 2011.

[HS06] Jonathan J. Hoch and Adi Shamir. Breaking the ice - finding multicollisions in iterated concatenated and expanded (ice) hash functions. In Robshaw [Rob06], pages 179–194.

[HS08] Jonathan J. Hoch and Adi Shamir. On the Strength of the Concatenated Hash Combiner When All the Hash Functions Are Weak. In Aceto et al. [ADG+08], pages 616–630.

[Huy86] Dung T. Huynh. A superexponential lower bound for gröbner bases and church-rosser commutative thue systems. *Information and Control*, 68(1-3):196 – 206, 1986.

[IF09] Kh. Ikramov and H. Fassbender. On the product of two skew-hamiltonian or two skew-symmetric matrices. *Journal of Mathematical Sciences*, 157:697–700, 2009. 10.1007/s10958-009-9352-z.

[Iso11] Takanori Isobe. A single-key attack on the full gost block cipher. In Antoine Joux, editor, *FSE*, volume 19?? of *Lecture Notes in Computer Science*. Springer, 2011.

[JKJMR05] Antoine Joux, Sébastien Kunz-Jacques, Frédéric Muller, and Pierre-Michel Ricordel. Cryptanalysis of the tractable rational map cryptosystem. In Vaudenay [Vau05b], pages 258–274.

[JL09] Antoine Joux and Stefan Lucks. Improved Generic Algorithms for 3-Collisions. In Matsui [Mat09], pages 347–363.

[JLS04] Svante Janson, Stefano Lonardi, and Wojciech Szpankowski. On average sequence complexity. *Theor. Comput. Sci.*, 326(1-3):213–227, 2004.

[Jou04] Antoine Joux. Multicollisions in Iterated Hash Functions. Application to Cascaded Constructions. In Matthew K. Franklin, editor, *CRYPTO'04*, volume 3152 of *Lecture Notes in Computer Science*, pages 306–316. Springer, 2004.

[Jou09a] Antoine Joux, editor. *Advances in Cryptology - EUROCRYPT 2009, 28th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Cologne, Germany, April 26-30, 2009. Proceedings*, volume 5479 of *Lecture Notes in Computer Science*. Springer, 2009.

[Jou09b] Antoine Joux. *Algorithmic Cryptanalysis*. Chapman & Hall/CRC, 2009.

[JP07] Antoine Joux and Thomas Peyrin. Hash Functions and the (Amplified) Boomerang Attack. In Menezes [Men07], pages 244–263.

[Jr.97] Burton S. Kaliski Jr., editor. *Advances in Cryptology - CRYPTO '97, 17th Annual International Cryptology Conference, Santa Barbara, California, USA, August 17-21, 1997, Proceedings*, volume 1294 of *Lecture Notes in Computer Science*. Springer, 1997.

[KBC97] Hugo Krawczyk, Mihir Bellare, and Ran Canetti. HMAC: Keyed-Hashing for Message Authentication. RFC2104, February 1997.

[KBN09] Dmitry Khovratovich, Alex Biryukov, and Ivica Nikolić. Speeding up Collision Search for Byte-Oriented Hash Functions. In Fischlin [Fis09], pages 164–181.

[Kel04] Liam Keliher. Refined Analysis of Bounds Related to Linear and Differential Cryptanalysis for the AES. In Dobbertin et al. [DRS05], pages 42–57.

[Ker92] Veikko Keränen. Abelian Squares are Avoidable on 4 Letters. In Werner Kuich, editor, *ICALP*, volume 623 of *Lecture Notes in Computer Science*, pages 41–52. Springer, 1992.

[Kim02] Kwangjo Kim, editor. *Information Security and Cryptology - ICISC 2001, 4th International Conference Seoul, Korea, December 6-7, 2001, Proceedings*, volume 2288 of *Lecture Notes in Computer Science*. Springer, 2002.

[KJ07] Sébastien Kunz-Jacques. *Preuves de sécurité et problèmes diffciles en cryptologie : études de cas*. PhD thesis, Université de Paris VII, 2007.

[KK06] John Kelsey and Tadayoshi Kohno. Herding Hash Functions and the Nostradamus Attack. In Vaudenay [Vau06], pages 183–200.

[Kli06] Vlastimil Klima. Tunnels in Hash Functions: MD5 Collisions Within a Minute. Cryptology ePrint Archive, Report 2006/105, 2006. `http://eprint.iacr.org/`.

[KM96] Klaus Kühnle and Ernst W. Mayr. Exponential space computation of gröbner bases. In *ISSAC*, pages 63–71, 1996.

[KM99] Lars R. Knudsen and Willi Meier. Cryptanalysis of an identification scheme based on the permuted perceptron problem. In *EUROCRYPT*, pages 363–374, 1999.

[KMNP11] Simon Knellwolf, Willi Meier, and María Naya-Plasencia. Conditional differential cryptanalysis of trivium and katan. In *Selected Areas in Cryptography*, August 2011.

[KMT01a] Liam Keliher, Henk Meijer, and Stafford E. Tavares. Improving the Upper Bound on the Maximum Average Linear Hull Probability for Rijndael. In Serge Vaudenay and Amr M. Youssef, editors, *Selected Areas in Cryptography*, volume 2259 of *Lecture Notes in Computer Science*, pages 112–128. Springer, 2001.

[KMT01b] Liam Keliher, Henk Meijer, and Stafford E. Tavares. New Method for Upper Bounding the Maximum Average Linear Hull Probability for SPNs. In Pfitzmann [Pfi01], pages 420–436.

[Knu02] Lars R. Knudsen, editor. *Advances in Cryptology - EUROCRYPT 2002, International Conference on the Theory and Applications of Cryptographic Techniques, Amsterdam, The Netherlands, April 28 - May 2, 2002, Proceedings*, volume 2332 of *Lecture Notes in Computer Science*. Springer, 2002.

[KP09] Eike Kiltz and Krzysztof Pietrzak. On the security of padding-based encryption schemes - or - why we cannot prove oaep secure in the standard model. In Joux [Jou09a], pages 389–406.

[KPG99] Aviad Kipnis, Jacques Patarin, and Louis Goubin. Unbalanced Oil and Vinegar Signature Schemes. In *EUROCRYPT*, pages 206–222, 1999.

[KR07] Lars R. Knudsen and Vincent Rijmen. Known-key distinguishers for some block ciphers. In Kurosawa [Kur07], pages 315–324.

[Kra98] Hugo Krawczyk, editor. *Advances in Cryptology - CRYPTO '98, 18th Annual International Cryptology Conference, Santa Barbara, California, USA, August 23-27, 1998, Proceedings*, volume 1462 of *Lecture Notes in Computer Science*. Springer, 1998.

[KS98] Aviad Kipnis and Adi Shamir. Cryptanalysis of the oil & vinegar signature scheme. In Krawczyk [Kra98], pages 257–266.

[KS99] Aviad Kipnis and Adi Shamir. Cryptanalysis of the HFE Public Key Cryptosystem by Relinearization. In Michael J. Wiener, editor, *CRYPTO*, volume 1666 of *Lecture Notes in Computer Science*, pages 19–30. Springer, 1999.

[KS05] John Kelsey and Bruce Schneier. Second Preimages on n-Bit Hash Functions for Much Less than 2^n Work. In Cramer [Cra05], pages 474–490.

[Kur07] Kaoru Kurosawa, editor. *Advances in Cryptology - ASIACRYPT 2007, 13th International Conference on the Theory and Application of Cryptology and Information Security, Kuching, Malaysia, December 2-6, 2007, Proceedings*, volume 4833 of *Lecture Notes in Computer Science*. Springer, 2007.

[KW03] Jonathan Katz and Nan Wang. Efficiency improvements for signature schemes with tight security reductions. In Sushil Jajodia, Vijayalakshmi Atluri, and Trent Jaeger, editors, *ACM Conference on Computer and Communications Security*, pages 155–164. ACM, 2003.

[Laz83] Daniel Lazard. Gröbner-bases, gaussian elimination and resolution of systems of algebraic equations. In J. A. van Hulzen, editor, *EUROCAL*, volume 162 of *Lecture Notes in Computer Science*, pages 146–156. Springer, 1983.

[LBF08] Gaëtan Leurent, Charles Bouillaguet, and Pierre-Alain Fouque. SIMD Is a Message Digest. Submission to NIST, 2008.

[LC06] Xuejia Lai and Kefei Chen, editors. *Advances in Cryptology - ASIACRYPT 2006, 12th International Conference on the Theory and Application of Cryptology and Information Security, Shanghai, China, December 3-7, 2006, Proceedings*, volume 4284 of *Lecture Notes in Computer Science*. Springer, 2006.

[LDKK08] Jiqiang Lu, Orr Dunkelman, Nathan Keller, and Jongsung Kim. New impossible differential attacks on aes. In Dipanwita Roy Chowdhury, Vincent Rijmen, and Abhijit Das, editors, *INDOCRYPT*, volume 5365 of *Lecture Notes in Computer Science*, pages 279–293. Springer, 2008.

[Lee04] Pil Joong Lee, editor. *Advances in Cryptology - ASIACRYPT 2004, 10th International Conference on the Theory and Application of Cryptology and Information Security, Jeju Island, Korea, December 5-9, 2004, Proceedings*, volume 3329 of *Lecture Notes in Computer Science*. Springer, 2004.

[Leu08a] Gaëtan Leurent. Md4 is not one-way. In Nyberg [Nyb08], pages 412–428.

[Leu08b] Gaëtan Leurent. Practical key-recovery attack against APOP, an MD5-based challenge-response authentication. *IJACT*, 1(1):32–46, 2008.

[Lis06] Moses Liskov. Constructing an ideal hash function from weak ideal compression functions. In Biham and Youssef [BY07], pages 358–375.

[LLMP93] A. Lenstra, H. Lenstra, M. Manasse, and J. Pollard. The number field sieve. In Arjen Lenstra and Hendrik Lenstra, editors, *The development of the number field sieve*, volume 1554 of *Lecture Notes in Mathematics*, pages 11–42. Springer Berlin / Heidelberg, 1993.

[LN97] Rudolf Lidl and Harald Niederreiter. *Finite fields*. Cambridge University Press, New York, NY, USA, 1997.

[LN09] Gaëtan Leurent and Phong Q. Nguyen. How risky is the random-oracle model? In Halevi [Hal09], pages 445–464.

[Luc05] Stefan Lucks. A Failure-Friendly Design Principle for Hash Functions. In Roy [Roy05], pages 474–494.

[Lyu08] Vadim Lyubashevsky. Lattice-based identification schemes secure under active attacks. In Cramer [Cra08], pages 162–179.

[Mac69] Jessie MacWilliams. Orthogonal matrices over finite fields. *The American Mathematical Monthly*, 76(2):152–164, 1969.

[Mat93] Mitsuru Matsui. Linear cryptoanalysis method for des cipher. In *EUROCRYPT*, pages 386–397, 1993.

[Mat09] Mitsuru Matsui, editor. *Advances in Cryptology - ASIACRYPT 2009, 15th International Conference on the Theory and Application of Cryptology and Information Security, Tokyo, Japan, December 6-10, 2009. Proceedings*, volume 5912 of *Lecture Notes in Computer Science*. Springer, 2009.

[May89] Ernst W. Mayr. Membership in plynomial ideals over q is exponential space complete. In Burkhard Monien and Robert Cori, editors, *STACS*, volume 349 of *Lecture Notes in Computer Science*, pages 400–406. Springer, 1989.

[May95] Ernst W. Mayr. On polynomial ideals, their complexity, and applications. In Horst Reichel, editor, *FCT*, volume 965 of *Lecture Notes in Computer Science*, pages 89–105. Springer, 1995.

[McE78] Robert McEliece. A Public-Key Cryptosystem Based on Algebraic Coding Theory, 1978. DSN Progress Report 42-44.

[MDBW09] Mohamed Mohamed, Jintai Ding, Johannes Buchmann, and Fabian Werner. Algebraic attack on the mqq public key cryptosystem. In Juan Garay, Atsuko Miyaji, and Akira Otsuka, editors, *Cryptology and Network Security*, volume 5888 of *Lecture Notes in Computer Science*, pages 392–401. Springer Berlin / Heidelberg, 2009.

[Men07] Alfred Menezes, editor. *Advances in Cryptology - CRYPTO 2007, 27th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 19-23, 2007, Proceedings*, volume 4622 of *Lecture Notes in Computer Science*. Springer, 2007.

[Mer89] Ralph C. Merkle. One Way Hash Functions and DES. In Brassard [Bra90], pages 428–446.

[MGH+05] Michael B. Monagan, Keith O. Geddes, K. Michael Heal, George Labahn, Stefan M. Vorkoetter, James McCarron, and Paul DeMarco. *Maple 10 Programming Guide*. Maplesoft, Waterloo ON, Canada, 2005.

[MI88] Tsutomu Matsumoto and Hideki Imai. Public Quadratic Polynominal-Tuples for Efficient Signature-Verification and Message-Encryption. In *EUROCRYPT*, pages 419–453, 1988.

[Mir01] Ilya Mironov. Hash functions: From merkle-damgård to shoup. In Pfitzmann [Pfi01], pages 166–181.

[MM82] Ernst W. Mayr and Albert R. Meyer. The complexity of the word problems for commutative semigroups and polynomial ideals. *Advances in Mathematics*, 46(3):305 – 329, 1982.

[MP08] Stéphane Manuel and Thomas Peyrin. Collisions on sha-0 in one hour. In Nyberg [Nyb08], pages 16–35.

[MR02] Sean Murphy and Matthew J. B. Robshaw. Essential Algebraic Structure within the AES. In Yung [Yun02], pages 1–16.

[MR10] Gilles Macario-Rat. *Cryptanalyse de schémas multivariés et résolution du problème Isomorphisme de Polynômes*. PhD thesis, Université Paris Diderot — Paris 7, June 2010.

[MRH04] Ueli M. Maurer, Renato Renner, and Clemens Holenstein. Indifferentiability, Impossibility Results on Reductions, and Applications to the Random Oracle Methodology. In Moni Naor, editor, *TCC*, volume 2951 of *Lecture Notes in Computer Science*, pages 21–39. Springer, 2004.

[MRV99] Silvio Micali, Michael O. Rabin, and Salil P. Vadhan. Verifiable random functions. In *FOCS*, pages 120–130, 1999.

[MV04] Jean Monnerat and Serge Vaudenay. On Some Weak Extensions of AES and BES. In Javier Lopez, Sihan Qing, and Eiji Okamoto, editors, *ICICS*, volume 3269 of *Lecture Notes in Computer Science*, pages 414–426. Springer, 2004.

[MvOV] A. Menezes, P van Oorschot, and S. Vanstone. Handbook of Applied Cryptography.

[Nac01] David Naccache, editor. *Topics in Cryptology - CT-RSA 2001, The Cryptographer's Track at RSA Conference 2001, San Francisco, CA, USA, April 8-12, 2001, Proceedings*, volume 2020 of *Lecture Notes in Computer Science*. Springer, 2001.

[Nao07] Moni Naor, editor. *Advances in Cryptology - EUROCRYPT 2007, 26th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Barcelona, Spain, May 20-24, 2007, Proceedings*, volume 4515 of *Lecture Notes in Computer Science*. Springer, 2007.

[Nie02] Jesper Buus Nielsen. Separating random oracle proofs from complexity theoretic proofs: The non-committing encryption case. In Yung [Yun02], pages 111–126.

[NIS01] NIST. Advanced Encryption Standard (AES), FIPS 197. Technical report, NIST, November 2001.

[Niv04] Gabriel Nivasch. Cycle detection using a stack. *Inf. Process. Lett.*, 90(3):135–140, 2004.

[NS07] Mridul Nandi and Douglas R. Stinson. Multicollision attacks on some generalized sequential hash functions. *IEEE Transactions on Information Theory*, 53(2):759–767, 2007.

[NY89] Moni Naor and Moti Yung. Universal One-Way Hash Functions and their Cryptographic Applications. In *STOC*, pages 33–43. ACM, 1989.

[Nyb08] Kaisa Nyberg, editor. *Fast Software Encryption, 15th International Workshop, FSE 2008, Lausanne, Switzerland, February 10-13, 2008, Revised Selected Papers*, volume 5086 of *Lecture Notes in Computer Science*. Springer, 2008.

[OK09] Özgul Küçük. The Hash Function Hamsi. Submission to NIST, 2009.

[Ore34] O. Ore. Contributions to The Theory of Finite Fields. *Transactions A. M. S.*, 36:243–274, 1934.

[OSS84a] H. Ong, Claus-Peter Schnorr, and Adi Shamir. An efficient signature scheme based on quadratic equations. In *STOC*, pages 208–216. ACM, 1984.

[OSS84b] H. Ong, Claus-Peter Schnorr, and Adi Shamir. Efficient signature schemes based on polynomial equations. In *CRYPTO*, pages 37–46, 1984.

[Pak71] A. G. Pakes. Some limit theorems for the total progeny of a branching process. *Advances in Applied Probability*, 3(1):176–192, 1971.

[Pan84] J.-J. Pansiot. Complexité des facteurs des mots infinis engendrés par morphismes itérés. In Jan Paredaens, editor, *11th ICALP, Antwerpen*, volume 172 of *LNCS*, pages 380–389. Springer, july 1984.

[Pat95] Jacques Patarin. Cryptoanalysis of the Matsumoto and Imai Public Key Scheme of Eurocrypt'88. In Don Coppersmith, editor, *CRYPTO*, volume 963 of *Lecture Notes in Computer Science*, pages 248–261. Springer, 1995.

[Pat96a] Jacques Patarin. Asymmetric cryptography with a hidden monomial. In Neal Koblitz, editor, *CRYPTO*, volume 1109 of *Lecture Notes in Computer Science*, pages 45–60. Springer, 1996.

[Pat96b] Jacques Patarin. Hidden fields equations (hfe) and isomorphisms of polynomials (ip): Two new families of asymmetric algorithms. In *EUROCRYPT*, pages 33–48, 1996.

[Pat96c] Jacques Patarin. Hidden Fields Equations (HFE) and Isomorphisms of Polynomials (IP): Two New Families of Asymmetric Algorithms. In *EUROCRYPT*, pages 33–48, 1996. Etended version available on `http://www.minrank.org/hfe.pdf`.

[Pat97] Jacques Patarin. The Oil and Vinegar signature scheme. presented at the Dagstuhl Workshop on Cryptography, 1997.

[PC93] Jacques Patarin and Pascal Chauvaud. Improved algorithms for the permuted kernel problem. In Stinson [Sti94], pages 391–402.

[PCG01a] Jacques Patarin, Nicolas Courtois, and Louis Goubin. Flash, a fast multivariate signature algorithm. In Naccache [Nac01], pages 298–307.

[PCG01b] Jacques Patarin, Nicolas Courtois, and Louis Goubin. QUARTZ, 128-Bit Long Digital Signatures. In Naccache [Nac01], pages 282–297.

[Per05] Ludovic Perret. A Fast Cryptanalysis of the Isomorphism of Polynomials with One Secret Problem. In Cramer [Cra05], pages 354–370.

[Pey07] Thomas Peyrin. Cryptanalysis of grindahl. In Kurosawa [Kur07], pages 551–567.

[Pfi01] Birgit Pfitzmann, editor. *Advances in Cryptology - EUROCRYPT 2001, International Conference on the Theory and Application of Cryptographic Techniques, Innsbruck, Austria, May 6-10, 2001, Proceeding*, volume 2045 of *Lecture Notes in Computer Science*. Springer, 2001.

[PG97] Jacques Patarin and Louis Goubin. Asymmetric cryptography with s-boxes. In Yongfei Han, Tatsuaki Okamoto, and Sihan Qing, editors, *ICICS*, volume 1334 of *Lecture Notes in Computer Science*, pages 369–380. Springer, 1997.

[PGC98a] Jacques Patarin, Louis Goubin, and Nicolas Courtois. c^* _-+ and hm: Variations around two schemes of t. matsumoto and h. imai. In Kazuo Ohta and Dingyi Pei, editors, *ASIACRYPT*, volume 1514 of *Lecture Notes in Computer Science*, pages 35–49. Springer, 1998.

[PGC98b] Jacques Patarin, Louis Goubin, and Nicolas Courtois. Improved Algorithms for Isomorphisms of Polynomials. In *EUROCRYPT*, pages 184–200, 1998.

[PGV93] Bart Preneel, René Govaerts, and Joos Vandewalle. Hash functions based on block ciphers: A synthetic approach. In Stinson [Sti94], pages 368–378.

[Pie07] Krzysztof Pietrzak. Non-trivial black-box combiners for collision-resistant hash-functions don't exist. In Naor [Nao07], pages 23–33.

[Pie08] Josef Pieprzyk, editor. *Advances in Cryptology - ASIACRYPT 2008, 14th International Conference on the Theory and Application of Cryptology and Information Security, Melbourne, Australia, December 7-11, 2008. Proceedings*, volume 5350 of *Lecture Notes in Computer Science*. Springer, 2008.

[Ple70] P. A. Pleasants. Non-repetitive sequences. *Mat. Proc. Camb. Phil. Soc.*, 68:267–274, 1970.

[Poi95] David Pointcheval. A new identification scheme based on the perceptrons problem. In *EUROCRYPT*, pages 319–328, 1995.

[PQ03] Gilles Piret and Jean-Jacques Quisquater. A Differential Fault Attack Technique against SPN Structures, with Application to the AES and KHAZAD. In Colin D. Walter, Çetin Kaya Koç, and Christof Paar, editors, *CHES*, volume 2779 of *Lecture Notes in Computer Science*, pages 77–88. Springer, 2003.

[Pre93] B. Preneel. *Analysis and design of cryptographic hash functions*. PhD thesis, Katholieke Universiteit Leuven, 1993.

[Pre00] Bart Preneel, editor. *Advances in Cryptology - EUROCRYPT 2000, International Conference on the Theory and Application of Cryptographic Techniques, Bruges, Belgium, May 14-18, 2000, Proceeding*, volume 1807 of *Lecture Notes in Computer Science*. Springer, 2000.

[PSC+02] Sangwoo Park, Soo Hak Sung, Seongtaek Chee, E-Joong Yoon, and Jongin Lim. On the security of rijndael-like structures against differential and linear cryptanalysis. In Zheng [Zhe02], pages 176–191.

[PSLL03] Sangwoo Park, Soo Hak Sung, Sangjin Lee, and Jongin Lim. Improving the Upper Bound on the Maximum Differential and the Maximum Linear Hull Probability for SPN Structures and AES. In Thomas Johansson, editor, *FSE*, volume 2887 of *Lecture Notes in Computer Science*, pages 247–260. Springer, 2003.

[PV05] Pascal Paillier and Damien Vergnaud. Discrete-log-based signatures may not be equivalent to discrete log. In Roy [Roy05], pages 1–20.

[Rab78] Michael O. Rabin. Digitalized signatures. In Richard A. DeMillo, Richard J. Lipton, David P. Dobkin, and Anita K. Jones, editors, *Foundations of Secure Computation*, pages 155–168, New York, USA, 1978. Academic Press, Inc.

[Rad07] Håvard Raddum. Mrhs equation systems. In Adams et al. [AMW07], pages 232–245.

[Rit22] J. F. Ritt. Prime and Composite Polynomials. *American M. S. Trans.*, 23:51–66, 1922.

[Riv05] Ronald L. Rivest. Abelian Square-Free Dithering for Iterated Hash Functions. *Presented at ECrypt Hash Function Workshop, June 21, 2005, Cracow, and at the Cryptographic Hash workshop, November 1, 2005, Gaithersburg, Maryland*, August 2005.

[Rob06] Matthew J. B. Robshaw, editor. *Fast Software Encryption, 13th International Workshop, FSE 2006, Graz, Austria, March 15-17, 2006, Revised Selected Papers*, volume 4047 of *Lecture Notes in Computer Science*. Springer, 2006.

[Rog06] Phillip Rogaway. Formalizing human ignorance. In Phong Q. Nguyen, editor, *VIETCRYPT*, volume 4341 of *Lecture Notes in Computer Science*, pages 211–228. Springer, 2006.

[Rog11] Phillip Rogaway, editor. *Advances in Cryptology - CRYPTO 2011, 31th Annual Cryptology Conference, Santa Barbara, CA, USA, August 14-18, 2011. Proceedings*, volume 6841 of *Lecture Notes in Computer Science*. Springer, 2011.

[Roy05] Bimal K. Roy, editor. *Advances in Cryptology - ASIACRYPT 2005, 11th International Conference on the Theory and Application of Cryptology and Information Security, Chennai, India, December 4-8, 2005, Proceedings*, volume 3788 of *Lecture Notes in Computer Science*. Springer, 2005.

[RS04] Phillip Rogaway and Thomas Shrimpton. Cryptographic Hash-Function Basics: Definitions, Implications, and Separations for Preimage Resistance, Second-Preimage Resistance, and Collision Resistance. In Bimal K. Roy and Willi Meier, editors, *FSE*, volume 3017 of *Lecture Notes in Computer Science*, pages 371–388. Springer, 2004.

[RSA78] Ronald L. Rivest, Adi Shamir, and Leonard M. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Commun. ACM*, 21(2):120–126, 1978.

[RSS11] Thomas Ristenpart, Hovav Shacham, and Thomas Shrimpton. Careful with composition: Limitations of the indifferentiability framework. In Kenneth G. Paterson, editor, *EUROCRYPT*, volume 6632 of *Lecture Notes in Computer Science*, pages 487–506. Springer, 2011.

[SA09] Yu Sasaki and Kazumaro Aoki. Finding preimages in full md5 faster than exhaustive search. In Joux [Jou09a], pages 134–152.

[Sem07] Igor Semaev. On solving sparse algebraic equations over finite fields (part ii). Cryptology ePrint Archive, Report 2007/280, 2007. `http://eprint.iacr.org/`.

[SG03] Andrey V. Sidorenko and Ernst M. Gabidulin. The Weak Keys For HFE. In *7th International Symposium on Communication Theory and Applications*, pages 239–244, 2003.

[Sha49] Claude E. Shannon. Communication theory of secrecy systems. *Bell System Technical Journal*, 28:656–715, 1949.

[Sha89] Adi Shamir. An efficient identification scheme based on permuted kernels (extended abstract). In Brassard [Bra90], pages 606–609.

[SHJ09] Paul Stankovski, Martin Hell, and Thomas Johansson. An efficient state recovery attack on x-fcsr-256. In Dunkelman [Dun09], pages 23–37.

[Sho97] Peter W. Shor. Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. *SIAM J. Comput.*, 26(5):1484–1509, 1997.

[Sho00a] Victor Shoup. A Composition Theorem for Universal One-Way Hash Functions. In Preneel [Pre00], pages 445–452.

[Sho00b] Victor Shoup. Using hash functions as a hedge against chosen ciphertext attack. In Preneel [Pre00], pages 275–288.

[Sho05] Victor Shoup, editor. *Advances in Cryptology - CRYPTO 2005: 25th Annual International Cryptology Conference, Santa Barbara, California, USA, August 14-18, 2005, Proceedings*, volume 3621 of *Lecture Notes in Computer Science*. Springer, 2005.

[Sho09] Victor Shoup. *A Computational Introduction to Number Theory and Algebra*. Cambridge University Press, New York, NY, USA, 2009.

[SKPI07] Makoto Sugita, Mitsuru Kawazoe, Ludovic Perret, and Hideki Imai. Algebraic cryptanalysis of 58-round sha-1. In Biryukov [Bir07], pages 349–365.

[SLdW07] Marc Stevens, Arjen K. Lenstra, and Benne de Weger. Chosen-Prefix Collisions for MD5 and Colliding X.509 Certificates for Different Identities. In Naor [Nao07], pages 1–22.

[Sma08] Nigel P. Smart, editor. *Advances in Cryptology - EUROCRYPT 2008, 27th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Istanbul, Turkey, April 13-17, 2008. Proceedings*, volume 4965 of *Lecture Notes in Computer Science*. Springer, 2008.

[SS08] Thomas Shrimpton and Martijn Stam. Building a collision-resistant compression function from non-compressing primitives. In Aceto et al. [ADG+08], pages 643–654.

[SSA+09] Marc Stevens, Alexander Sotirov, Jacob Appelbaum, Arjen K. Lenstra, David Molnar, Dag Arne Osvik, and Benne de Weger. Short chosen-prefix collisions for md5 and the creation of a rogue ca certificate. In Halevi [Hal09], pages 55–69.

[SSH11] Koichi Sakumoto, Taizo Shirai, and Harunaga Hiwatari. Public-key identification schemes based on multivariate quadratic polynomials. In Rogaway [Rog11], pages 706–723.

[Sta86] R P Stanley. *Enumerative combinatorics*. Wadsworth Publ. Co., Belmont, CA, USA, 1986.

[Ste22] H. Stenzen. Über die darstellbarkeit einer matrix als produkt von zwei symmetrischer matrizen, als produkt von zwei alternierenden matrizen und als produkt von einer symmetrischen und einer alternierenden matrix. *Math. Z.*, 15:1–25, 1922.

[Ste93] Jacques Stern. A new identification scheme based on syndrome decoding. In Stinson [Sti94], pages 13–21.

[Ste94] Jacques Stern. Designing identification schemes with keys of short size. In Yvo Desmedt, editor, *CRYPTO*, volume 839 of *Lecture Notes in Computer Science*, pages 164–173. Springer, 1994.

[Sti94] Douglas R. Stinson, editor. *Advances in Cryptology - CRYPTO '93, 13th Annual International Cryptology Conference, Santa Barbara, California, USA, August 22-26, 1993, Proceedings*, volume 773 of *Lecture Notes in Computer Science*. Springer, 1994.

[TC86] Ren Ji Tao and Shi Hua Chen. Two varieties of finite automaton public key cryptosystem and digital signatures. *Journal of computer science and technology*, 1(1):9–18, 1986.

[TCHP08] Marc Tedder, Derek G. Corneil, Michel Habib, and Christophe Paul. Simpler linear-time modular decomposition via recursive factorizing permutations. In Luca Aceto, Ivan Damgård, Leslie Ann Goldberg, Magnús M. Halldórsson, Anna Ingólfsdóttir, and Igor Walukiewicz, editors, *ICALP (1)*, volume 5125 of *Lecture Notes in Computer Science*, pages 634–645. Springer, 2008.

[Vau05a] Serge Vaudenay. *A Classical Introduction to Cryptography: Applications for Communications Security*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2005.

[Vau05b] Serge Vaudenay, editor. *Public Key Cryptography - PKC 2005, 8th International Workshop on Theory and Practice in Public Key Cryptography, Les Diablerets, Switzerland, January 23-26, 2005, Proceedings*, volume 3386 of *Lecture Notes in Computer Science*. Springer, 2005.

[Vau06] Serge Vaudenay, editor. *Advances in Cryptology - EUROCRYPT 2006, 25th Annual International Conference on the Theory and Applications of Cryptographic Techniques, St. Petersburg, Russia, May 28 - June 1, 2006, Proceedings*, volume 4004 of *Lecture Notes in Computer Science*. Springer, 2006.

[vL00] Charles F. van Loan. The ubiquitous kronecker product. *J. Comput. Appl. Math.*, 123:85–100, November 2000.

[vRS06] Håvard Raddum and Igor Semaev. New technique for solving sparse equation systems. Cryptology ePrint Archive, Report 2006/475, 2006. `http://eprint.iacr.org/`.

[vzG90a] Joachim von zur Gathen. Functional Decomposition of Polynomials: The Tame Case. *J. Symb. Comput.*, 9(3):281–299, 1990.

[vzG90b] Joachim von zur Gathen. Functional Decomposition of Polynomials: The Wild Case. *J. Symb. Comput.*, 10(5):437–452, 1990.

[War35] Ewald Warning. Bemerkung zur vorstehenden Arbeit von Herrn Chevalley. *Abh. Math. Semin. Hamb. Univ.*, 11:76–83, 1935.

[Wei] Eric W. Weisstein. "q-analog". From MathWorld–A Wolfram Web Resource. "`http://mathworld.wolfram.com/q-Analog.html`".

[WHL+05] Lih-Chung Wang, Yuh-Hua Hu, Feipei Lai, Chun yen Chou, and Bo-Yin Yang. Tractable rational map signature. In Vaudenay [Vau05b], pages 244–257.

[Wik11] Wikipedia. Hash function — wikipedia, the free encyclopedia, 2011. [Online; accessed 20-May-2011].

[Wil80] Hugh C. Williams. A modification of the rsa public-key encryption procedure. *IEEE Transactions on Information Theory*, 26:726–729, 1980.

[Win84] Robert S. Winternitz. A Secure One-Way Hash Function Built from DES. In *IEEE Symposium on Security and Privacy*, pages 88–90, 1984.

[WLF+05] Xiaoyun Wang, Xuejia Lai, Dengguo Feng, Hui Chen, and Xiuyuan Yu. Cryptanalysis of the Hash Functions MD4 and RIPEMD. In Cramer [Cra05], pages 1–18.

[WM97] David Wolpert and William G. Macready. No Free Lunch Theorems for Optimization. *IEEE Trans. Evolutionary Computation*, 1(1):67–82, 1997.

[WP05a] Christopher Wolf and Bart Preneel. Large Superfluous Keys in Multivariate Quadratic Asymmetric Systems. In Vaudenay [Vau05b], pages 275–287.

[WP05b] Christopher Wolf and Bart Preneel. Taxonomy of Public Key Schemes Based on the Problem of Multivariate Quadratic Equations. Cryptology ePrint Archive, Report 2005/077, 2005.

[WY05] Xiaoyun Wang and Hongbo Yu. How to Break MD5 and Other Hash Functions. In Cramer [Cra05], pages 19–35.

[WYY05a] Xiaoyun Wang, Yiqun Lisa Yin, and Hongbo Yu. Finding Collisions in the Full SHA-1. In Shoup [Sho05], pages 17–36.

[WYY05b] Xiaoyun Wang, Hongbo Yu, and Yiqun Lisa Yin. Efficient Collision Search Attacks on SHA-0. In Shoup [Sho05], pages 1–16.

[WZ92] Herbert Wilf and Doron Zeilberger. An algorithmic proof theory for hypergeometric (ordinary and "q") multisum/integral identities. *Inventiones Mathematicae*, 108:575–633, 1992. 10.1007/BF02100618.

[Yas08] Kan Yasuda. How to fill up merkle-damgård hash functions. In Pieprzyk [Pie08], pages 272–289.

[Yun02] Moti Yung, editor. *Advances in Cryptology - CRYPTO 2002, 22nd Annual International Cryptology Conference, Santa Barbara, California, USA, August 18-22, 2002, Proceedings*, volume 2442 of *Lecture Notes in Computer Science*. Springer, 2002.

[YWJ+09] Zheng Yuan, Wei Wang, Keting Jia, Guangwu Xu, and Xiaoyun Wang. New Birthday Attacks on Some MACs Based on Block Ciphers. In Halevi [Hal09], pages 209–230.

[Zhe02] Yuliang Zheng, editor. *Advances in Cryptology - ASIACRYPT 2002, 8th International Conference on the Theory and Application of Cryptology and Information Security, Queenstown, New Zealand, December 1-5, 2002, Proceedings*, volume 2501 of *Lecture Notes in Computer Science*. Springer, 2002.

Résumé

La cryptanalyse, anciennement l'*art de déchiffrer les codes secrets*, est aujourd'hui comprise dans un sens plus large, qui consiste à trouver toute sorte de défauts, des plus légers jusqu'au plus graves, dans des constructions cryptographiques. Cette thèse s'articule en trois partie, chacune consacrée à une famille de primitives cryptographiques.

Les fonctions de hachage sont généralement des constructions itérées, où une brique de base est utilisée de façon répétée, conformément à mode opératoire. En 2004, 2005 et 2006, l'apparition d'attaques génériques, visant spécialement le mode opératoire de Merkle-Damgård, le plus répandu, a suscité des propositions de modes opératoires alternatifs. Les attaques génériques exploitent principalement le fait que l'état interne des fonctions de hachage a la même taille que leur sortie, ce qui permet de trouver et d'exploiter des collisions internes. Nos résultats tendent à montrer que cette difficulté est difficile à lever, car nous parvenons à monter des attaques en seconde préimage contre des tentatives, parfois exotiques, de "réparer" la construction de Merkle-Damgård.

Dans une deuxième partie, nous nous intéressons à la cryptanalyse du système de chiffrement par bloc le plus répandu (l'AES). Nous nous plaçons dans un modèle assez restrictif où la quantité de paires clair/chiffré disponible pour l'attaquant est très faible. Cela nous condamne à étudier des versions fortement réduites de l'AES, mais cela rend les attaques obtenues plus facilement réutilisables dans d'autres contextes. Nous avons construit des outils automatiques pour nous assister dans la recherche d'attaques par test d'hypothèses partielles et d'attaques par le milieu. Ces outils ont découverts des attaques surprenantes, qui sont plus efficaces que celles qui avaient été trouvées manuellement par d'autres chercheurs. Nous obtenons ainsi la meilleure attaque connue contre la fonction d'authentification Pelican-MAC et contre le système de chiffrement par flot LEX.

La dernière partie de cette thèse est consacrée à la cryptanalyse de schémas multivariés, c'est-à-dire reposant explicitement sur la difficulté de résoudre des systèmes polynomiaux en plusieurs variables. Certains de ces schémas reposent aussi sur une autre hypothèse algorithmiques, la difficulté problème de l'Equivalence Linéaire de Polynômes (PLE). En combinant des outils de l'algèbre linéaire et de la géométrie algébrique avec des techniques statistiques et combinatoires, nous construisons de nouveaux algorithmes plus efficace pour PLE. Ceci montre qu'un schéma dont la sécurité reposerait sur PLE ne peut pas exhiber un niveau de sécurité optimal. Ces algorithmes permettent également de casser en pratique la variante "sous-corps" de HFE.

Abstract

Cryptanalysis, formerly known as the art of deciphering secret codes, is now understood in a broader sense: it consists in finding flaws of all kinds, either harmless or severe, in cryptographic constructions. This thesis is made of three parts, devoted to the study of different families of cryptographic primitives.

Hash functions are usually iterated constructions, where a building block is used in a repeated way, as specified by a mode of operation. In 2004, 2005 and 2006, the discovery of several generic attacks, targeting the ubiquitous Merkle-Damgård mode of operation, prompted researchers to design alternative modes of operations. Generic attacks exploit the fact that the internal state of the hash functions has the same size as the digests. This allows the attackers to find internal collisions and exploit them. Our results tend to show that this problem is difficult to avoid in general: in the first part of this thesis, we describe several generic second preimage attacks against proposals to repair or patch the Merkle-Damgård construction in order to precisely avoid generic attacks.

In the second part, we focus on the cryptanalysis of the AES, the most popular block cipher. We consider a somewhat restrictive attack model where the attacker only has access to very few plaintext/ciphertext pairs. We are thus condemned to attack only highly reduced version of the full cipher, but the attacks we find in this model can sometimes be reused in other contexts. We have built software tools to assist us in finding guess-and-determine as well as meet-in-the-middle attacks. These tools found surprising attacks that are more efficient than those found manually by other cryptanalysts. For instance, we find the best known attacks against the Message Authentication Code Pelican-MAC, and against the stream cipher LEX.

The last part of this thesis is devoted to the cryptanalysis of multivariate schemes. This label covers all the schemes whose security explicitly relies on the hardness of solving systems of polynomial equations in several unknowns. Some multivariate scheme also rely on another hardness assumption, namely the hardness of the Polynomial Linear Equivalence (PLE) problem. We build new algorithms to solve PLE problems by combining tools from linear algebra and algebraic geometry with combinatorial and statistical techniques. Our algorithms show that a multivariate scheme relying on the hardness of PLE cannot exhibit an optimal security level. These algorithms also allow to break in practice the "subfield" variant of HFE.