ANNÉE 2016

UNIVERSITÉ DE RENNES 1

UNIVERSITE BRETAGNE LOIRE

THÈSE / UNIVERSITÉ DE RENNES 1
sous le sceau de l'Université de Bretagne Loire

pour le grade de

DOCTEUR DE L'UNIVERSITÉ DE RENNES 1

Mention : informatique

Ecole doctorale Matisse

présentée par

Brice Minaud

préparée à l'Institut de Recherche en Informatique et Systèmes Aléatoires (IRISA), UMR 6074

Analyse de primitives cryptographiques récentes

Thèse soutenue à Rennes
le 7 octobre 2016

devant le jury composé de :

Henri GILBERT
Chercheur associé, ANSSI/UVSQ – rapporteur

Louis GOUBIN
Professeur, UVSQ – rapporteur

Anne CANTEAUT
Directrice de recherche, Inria – examinatrice

Jean-Sébastien CORON
Professeur adjoint, U. Luxembourg – examinateur

Antoine JOUX
Professeur, UPMC – examinateur

Reynald LERCIER
Chercheur associé, DGA/IRMAR – examinateur

David POINTCHEVAL
DR, CNRS/ENS/Inria – examinateur

Pierre-Alain FOUQUE
Professeur, U. Rennes 1 – directeur de thèse

Remerciements

En premier lieu je remercie Pierre-Alain et l'ensemble de l'équipe EMSEC pour deux années de thèse qui ont été à la fois productives et agréables. Pierre-Alain est toujours présent, et en plus de ses qualités scientifiques, a une patience et une gentillesse qui contribuent largement à l'atmosphère de l'équipe. Je remercie aussi les membres de l'équipe, en particulier mes confrères du bureau F413, Pierre, Pierre et Raphaël ; les maîtres du badminton, Benjamin, Cyrille et Patrick ; les piliers de la pause thé, Baptiste, Florent, Paul et Pauline, mais aussi Alban et Benoît quand ils sont de passage, et Benjamin H. et Jean-Christophe avant qu'ils ne repartent ; et enfin Adeline, Barbara, Cécile, Cristina et Gildas, dont la présence est très appréciée nonobstant qu'ils ne rentrent pas dans une des catégories précédentes.

Je suis aussi très reconnaissant envers les membre du laboratoire de cryptologie de l'ANSSI, en particulier Henri et Thomas, qui ont accepté d'adopter un logicien pour faire de la cryptanalyse, puis réussi à créer un poste temporaire dans une conjoncture difficile ; sans oublier mes voisins de bureau Jean-Pierre, Jérôme et Yannick, qui ne m'ont jamais reproché d'avoir remplacé la machine à café.

Je remercie aussi Mireille Fouquet, qui a organisé un master de cryptologie impeccable ; David Pointcheval, qui a accepté que je vienne suivre son cours de preuve hors cursus ; les responsables des projets BLOC et BRUTUS, qui ont financé plusieurs trajets ; et Louis Granboulan, qui a été tour à tour mon tuteur à l'ENS, un directeur de projet, et plusieurs fois mon enseignant : c'est par lui que j'ai entendu pour la première fois parler de cryptographie et que j'ai voulu en savoir plus.

Je suis également reconnaissant aux collègues et amis avec qui j'ai partagé ma thèse précédente, c'est-à-dire les membre effectifs ou honorifiques du fameux bureau 5C6, qui était devenu une institution au point que sa fermeture donne lieu à une cérémonie : Ana, Avenilde, David, Fares, Laura, Luis, Nicole, Pablo, Rémi et Yann.

Enfin, je remercie vivement Henri Gilbert et Louis Goubin pour avoir accepté de rapporter ma thèse, et tous les membres du jury pour avoir accepté de venir.

Contents

Introduction générale

1 Cryptographie symétrique et asymétrique

L'objet premier de la cryptographie est la conception et l'analyse de communications sécurisées. Au sens moderne, il s'agit de communications informatiques. Cependant on distingue généralement la cryptographie de la sécurité informatique, qui joue un rôle complémentaire.

Une communication informatique peut se décomposer en un certain nombre de niveaux, depuis les couches dites de bas niveau, comme les couches matérielles ou les différentes couches réseau, jusqu'aux couches de haut niveau, qui portent la charge utile de la communication. Chaque nouvelle couche a tendance à abstraire les couches inférieures, c'est-à-dire à modéliser et supposer de manière plus ou moins explicite leur bon fonctionnement. La cryptographie s'attache aux couches supérieures de cette décomposition : elle traite de la sécurité dans un monde partiellement idéalisé, en faisant le plus souvent abstraction des failles de sécurité dûes au contexte d'exécution : bogues d'implémentation, failles matérielles, virus, etc.

Dans ce cadre, l'objectif le plus simple de la cryptographie, et sans doute l'un des premiers, est la transmission de messages confidentiels. Ces messages ne doivent être intelligibles que pour l'émetteur et le récepteur, ou plus généralement tout parti en possession d'un secret qui permet le déchiffrement. Le terme de confidentialité évoque naturellement une application militaire. Cependant, avec le déploiement global de réseaux informatiques et d'internet depuis la fin du siècle dernier, des techniques assurant la confidentialité sont nécessaires à grande échelle pour la population civile. En effet le commerce par internet ou l'accès aux comptes bancaires en ligne nécessitent de transmettre des informations privées, qui transitent sur un réseau public.

La première garantie offerte par la cryptographie est donc la confidentialité : un message chiffré ne donne aucune information sur son contenu, sauf si l'on est en possession de la clef secrète de déchiffrement. Cependant dans de nombreuses applications cryptographiques, d'autres garanties classiques de la cryptographie sont nécessaires, comme l'authenticité et l'intégrité. L'authenticité permet au destinataire de s'assurer que le message provienne bien de l'émetteur prétendu ; l'intégrité lui permet de vérifier que le message n'a pas été modifié depuis son écriture.

Les trois notions de confidentialité, authenticité et intégrité appartiennent au domaine de la cryptographie symétrique. Le terme symétrique est employé pour exprimer le fait qu'il existe un unique secret commun aux partis qui communiquent : ce même secret permet à la fois le chiffrement et le déchiffrement, l'authentification et sa vérification. La situation entre les partis possédant le secret est donc symétrique. On peut remarquer notamment que l'authenticité permet seulement de garantir que le message provient d'un des partis en possession du secret, plutôt que d'un parti spécifique.

Au contraire la cryptographie asymétrique englobe toutes les situations où les partis qui communiquent ne partagent pas la même information secrète. Jusqu'à présent nous avons défini les trois notions de confidentialité, authenticité et intégrité pour la cryptographie symétrique. La cryptographie asymétrique permet elle aussi de satisfaire des notions de sécurité variées–en

fait en très grand nombre. Nous nous contentons ici de mentionner deux notions parmi les plus courantes.

La plus simple est peut-être la notion de signature électronique : comme une signature physique, une signature électronique permet de prouver l'identité du parti qui signe un message, de manière vérifiable par tous. Ainsi il n'est plus nécessaire d'être en possession d'une information secrète pour la vérification, comme dans le cas (symétrique) de l'authentification : tout le monde peut vérifier l'exactitude d'une signature ; mais en principe, une seule personne est capable de la créer, prouvant par là son identité.

Il y a donc une clef privée, propre au signataire, qui lui confère sa capacité à signer des messages ; et une clef publique, commune à tous, qui permet de vérifier les signatures. Pour cette raison on appelle aussi la cryptographie asymétrique cryptographie à clef publique. On peut définir de la même façon un chiffrement à clef publique, où une clef publique commune permet de chiffrer des messages, mais ceux-ci ne pourront être déchiffrés qu'à l'aide d'une clef privée, secrète, associée à la clef publique. Nous rencontrerons par la suite d'autres exemples de notions asymétriques, comme les échanges de clef multipartis.

En général les primitives de cryptographie symétrique remplissent des fonctions relativement simples, comme l'authenticité et la confidentialité, mais sont très rapides. Elles peuvent être utilisées pour chiffrer de gros volumes de données. Au contraire les primitives asymétriques fournissent des fonctions plus riches, mais sont relativement lentes. Elles sont utilisées ponctuellement pour assurer des propriétés de sécurité critiques, tandis que la masse des communications est chiffrée de manière symétrique.

2 Notions de cryptographie

Dans cette partie nous allons décrire quelques bases de la cryptographie. L'exposé qui suit est bref et incomplet : il se concentre sur les notions nécessaires pour comprendre les travaux qui suivent, ou au moins leur donner un contexte.

2.1 Primitives, modes et protocoles

En premier lieu, on peut remarquer que les applications cryptographiques déjà mentionnées, comme le paiement en ligne, consistent en fait en un échange d'information entre plusieurs entités. La description précise d'un tel échange est appelée un protocole. Ainsi, sans entrer dans les détails, lorsqu'un utilisateur se connecte sur un site sécurisé suivant le protocole HTTPS, la confidentialité de la communication et l'identité des partis sont assurées par le protocole TLS, qui spécifie très précisément la forme des échanges autorisés.

Un protocole sécurisé s'appuie lui-même sur des éléments cryptographiques plus fondamentaux, que sont par exemple le chiffrement ou la signature, déjà évoqués. Dans le cas du chiffrement, il peut reposer à son tour sur une brique encore plus fondamentale, comme un chiffrement par bloc. Dans ce cas, l'utilisation du chiffrement par bloc pour créer un chiffrement est spécifiée par ce qu'on appelle un mode opératoire. Dans cet empilement de couches, les constructions les plus fondamentales, qui ne reposent sur aucune autre, s'appellent les *primitives* cryptographiques. La sécurité de ces primitives constitue en quelque sorte les axiomes de la cryptographie.

Dans toute la suite, nous nous intéresserons presque exclusivement aux primitives cryptographiques, et non aux modes ni aux protocoles.

2.2 Modèles et preuves

Comme les autres sciences dures, la cryptographie repose au moins partiellement sur des bases mathématiques. En particulier les notions de sécurités ébauchées dans l'introduction, comme la confidentialité ou la sécurité des signatures, peuvent être définies formellement. Ces définitions sont souvent appelées des modèles, parce qu'elles interprètent mathématiquement des idées à l'origine informelles. De plus une même notion de sécurité, comme celle d'une signature électronique, peut être exprimée par des modèles différents, selon la propriété exacte que l'on souhaite.

Une fois le modèle de sécurité fixé, une construction visant à satisfaire ce modèle (par exemple une construction de signature électronique) peut être soit prouvée vis-à-vis du modèle, soit simplement conjecturée sûre dans le modèle, si aucune preuve n'est connue. Les primitives symétriques et asymétriques diffèrent sur ce point.

Les primitives de cryptographie symétrique ne sont (presque) jamais prouvées. En effet elles reposent sur des techniques efficaces, mais qui se prêtent (volontairement) peu à l'analyse mathématique. Au contraire les primitives asymétriques s'appuient généralement sur une forte structure mathématique, nécessaire pour fournir des propriétés d'utilisation plus riches. Cette structure mathématique peut être exploitée par des attaques, et impose aux primitives asymétriques une relative lenteur. Par contre grâce aussi aux structures mathématiques sous-jacentes, les primitives asymétriques classiques, comme les signatures et le chiffrement à clef publique, sont généralement prouvées.

Il faut cependant qualifier le terme de preuve. On ne prouve pas qu'il soit impossible pour un attaquant de contredire le modèle de sécurité, parce qu'un attaquant non borné peut toujours essayer toutes les clefs privées possibles, en quantité a priori bornée. Même pour un attaquant borné, on ne prouve pas non plus de manière absolue qu'il soit difficile pour lui d'attaquer le chiffrement, parce que de telles preuves sont hors de portée aujourd'hui[1]. Les preuves de sécurité consistent à montrer que si un attaquant sait efficacement attaquer la construction dans un certain modèle, alors il sait résoudre efficacement un problème réputé difficile, comme la factorisation de grands entiers. Une preuve fournit ainsi une garantie de sécurité significative, mais pas absolue.

Il faut encore mentionner d'autres primitives asymétriques moins classiques, comme les applications multilinéaires, qu'on ne sait pas prouver au sens précédent, et dont on ne sait pas même encore de manière certaine si elles existent. Suivant un article fameux d'Impagliazzo [Imp95], on peut décrire un monde cryptographique où ces primitives existent, un autre où elles n'existent pas, sans savoir encore dans lequel on vit.

2.3 Paramètre de sécurité

Une construction répond à une définition de sécurité définie par un modèle, comme expliqué plus haut. Ce modèle contient un ou plusieurs paramètres de sécurité, qui expriment la difficulté pour un adversaire d'attaquer le chiffrement. Dans le cas le plus simple, qui est aussi un des plus courants, le modèle contient un unique paramètre de sécurité, noté λ. Ce paramètre signifie qu'un attaquant contre la construction cryptographique ne pourra réussir qu'avec une puissance de calcul 2^λ, mesurée suivant les cas en nombre d'opérations élémentaires ou de chiffrements. Le paramètre λ est généralement fixé de manière à ce que la puissance de calcul nécessaire à un attaquant soit irréalisable avec les moyens de l'informatique actuelle, y compris dans un futur

[1] L'exemple canonique est la non résolution du problème $P \neq NP$. Mais plus généralement, prouver des bornes inférieures absolues (non réductionnistes) dans des modèles de complexité standards est un problème que l'on ne semble pas savoir résoudre actuellement, en dehors de bornes « simples » provenant de la théorie de l'information.

proche[2].

Par ailleurs, pour les constructions asymétriques en particulier, une construction cryptographique est souvent vue comme une famille de constructions, dont λ est l'un des paramètres. On peut alors s'intéresser à la complexité asymptotique des attaques, qui fournissent une approximation simple de leur complexité pratique. En particulier on impose en général qu'attaquer un schéma soit exponentiellement plus coûteux que de l'utiliser.

2.4 Cryptanalyse

La cryptanalyse est le sous-domaine de la cryptographie qui cherche à évaluer la sécurité des constructions, par opposition notamment à leur conception[3]. Cela revient essentiellement à rechercher des attaques. En effet le choix des paramètres d'une construction doit être tel qu'il rend toute attaque irréalisable. Ici il faut séparer plusieurs cas, selon que la construction est prouvée ou non.

Si une construction est non prouvée, la confiance en sa sécurité n'existe généralement que si la construction fait partie d'un type de construction bien étudié (comme les chiffrements par bloc), pour lesquels les principaux vecteurs d'attaques sont considérés comme connus et bien maîtrisés (attaques linéaires et différentielles, par exemple). Dans ce cas, il faut évaluer l'efficacité de ces attaques, puis fixer les paramètres de la construction de manière à la placer hors de portée des attaques. Bien sûr la construction peut être conçue de manière à faciliter son évaluation. D'autre part des attaques dédiées peuvent persister, et une analyse approfondie de la part de cryptanalystes indépendants, sur un long intervalle de temps, est inévitable.

Si la construction est prouvée, il faut encore séparer deux cas. Soit sa preuve fournit une réduction efficace vers un problème difficile, auquel cas attaquer la construction revient à attaquer le problème difficile : la cryptanalyse devient un problème appartenant autant à l'algorithmique générale qu'à la cryptologie en particulier. C'est le cas des attaques sur le logarithme discret ou la factorisation, par exemple. Sinon, si la preuve ne fournit pas de réduction efficace, la situation n'est pas si différente du cas non prouvé : il faut évaluer la sécurité concrète des meilleures attaques, en tenant compte des fossés éventuels séparant la construction du problème difficile sous-jacent. C'est encore le cas si le modèle de sécurité n'englobe pas toutes les propriétés de sécurité que l'on pourrait souhaiter.

Ainsi le travail de la cryptanalyse consiste à attaquer des constructions cryptographiques. Mais indirectement, sa contribution majeure est aussi de dresser un tableau des attaques pertinentes pour un certain type de construction ou de problème. Si une construction résiste à ces attaques, et à un effort cryptanalytique important et durable, elle peut être considérée comme sûre. Puisqu'il n'y a (presque) jamais de borne inférieure absolue prouvée sur l'attaquant, c'est en fin de compte sur cette base que repose la confiance en la sécurité d'un chiffrement.

3 Définitions utiles

Dans cette section, nous donnons quelques définitions usuelles, qui servent à la fois à préciser certains énoncés informels qui précèdent, et qui seront utiles pour présenter nos travaux dans la section suivante.

[2]Pour λ suffisamment élevé, il est aussi possible d'argumenter que la puissance de calcul nécessaire est physiquement impossible.

[3]En français, on réserve parfois le terme de cryptographie pour la conception. Dans ce cas l'ensemble formé par la cryptographie et la cryptanalyse est appelé cryptologie. Ici nous utilisons la convention anglaise et internationale, pour laquelle cryptographie désigne la discipline dans son ensemble.

3.1 Chiffrement par bloc

Un chiffrement par bloc est une primitive symétrique définie de la manière suivante.

Definition 1 (Chiffrement par bloc)**.** *Étant donné un espace de clef $\mathcal{K}$, un espace de message clair $\mathcal{P}$ et un espace de message chiffré $\mathcal{C}$, un* chiffrement par bloc *est une fonction $E : \mathcal{K} \times \mathcal{P} \to \mathcal{C}$ telle que pour tout $K \in \mathcal{K}$, $x \mapsto E(K, x)$ est inversible.*

Si $\mathcal{P}$ et $\mathcal{C}$ sont en bijection, ils peuvent être assimilés. Dans ce cas un chiffrement par bloc peut être vu comme une application de K dans les permutations de $\mathcal{P}$. En général $\mathcal{P} = \mathcal{C} = \{0,1\}^n$ pour un certain entier n, la taille de bloc, typiquement $n = 128$. De même $\mathcal{K} = \{0,1\}^k$ pour un certain entier k, la taille de clef.

Informellement, le modèle standard de sécurité associé à un chiffrement par bloc est le suivant. Aucun adversaire efficace ne sait distinguer un oracle donnant accès à $E(K, \cdot)$ et son inverse, pour K tiré uniformément, d'un oracle donnant accès à une permutation uniformément aléatoire et son inverse. Du point de vue d'un adversaire ne possédant pas la clef secrète, un bon chiffrement par bloc se comporte donc comme une permutation aléatoire.

D'autres modèles de sécurité sont possibles. En particulier, le modèle à clef liée, qui sera brièvement mentionné dans le premier chapitre, impose (au minimum) qu'aucun adversaire efficace ne sache distinguer la donnée conjointe de $E(K, \cdot)$ et $E(f(K), \cdot)$ et leurs inverses, pour f appartenant à une certaine classe de fonctions (typiquement, les translations par XOR), de deux permutations uniformes indépendantes et leurs inverses.

Dans le même chapitre, nous mentionnerons aussi les chiffrements par bloc avec tweak (*tweakable bloc cipher*), définis comme suit.

Definition 2 (Chiffrement par bloc avec tweak)**.** *Étant donné un espace de clef $\mathcal{K}$, un espace de tweak $\mathcal{T}$, un espace de message clair $\mathcal{P}$ et un espace de message chiffré $\mathcal{C}$, un* chiffrement par bloc avec tweak *est une fonction $E : \mathcal{K} \times \mathcal{T} \times \mathcal{P} \to \mathcal{C}$ telle que pour tout $K \in \mathcal{K}, T \in \mathcal{T}$, $x \mapsto E(K, T, x)$ est inversible.*

La définition ci-dessus est équivalente à un chiffrement par bloc avec espace de clef $\mathcal{K} \times \mathcal{T}$. La distinction entre la clef et le tweak apparaît dans le modèle de sécurité : la clef est tirée uniformément indépendamment d'un adversaire potentiel, tandis que le tweak peut généralement être non seulement connu, mais choisi par l'adversaire. La définition standard de la sécurité d'un chiffrement par bloc avec tweak exige que pour K uniforme fixé, la famille paramétrée par T des permutations $E(K, T, \cdot)$ avec leur inverse soit indistinguable pour un adversaire efficace d'une famille de permutations uniformes et indépendantes avec leur inverse.

3.2 Chiffrement à clef publique

Dans les chapitres 2 et 3, il sera question de chiffrement à clef publique. En fait, il sera plus précisément question de permutation à trappe, une primitive à partir de laquelle un chiffrement à clef publique peut être ensuite construit de manière standard. Nous donnons ici une définition de cette primitive. L'idée informelle est qu'une permutation à trappe est une famille de permutations auxquelles sont associées des « trappes ». Pour chaque permutation, calculer la permutation doit être facile, mais l'inverser difficile, sauf si l'on connaît la trappe.

Definition 3 (Permutation à trappe)**.** *Pour un espace de clef $\mathcal{K}$, un espace de message $\mathcal{P}_K$, un espace de chiffré $\mathcal{C}_K$ (généralement dépendants de la clef) et un espace de trappe $\mathcal{T}$, une* permutation à trappe *est une famille de permutations $E_K : \mathcal{P}_K \to \mathcal{C}_K$, paramétrée par une clef $K \in \mathcal{K}$, satisfaisant les propriétés suivantes :*

1. *Il existe un algorithme efficace permettant de tirer une clef K, avec une trappe $T(K) \in \mathcal{T}$ associée.*

2. *Il existe des algorithmes efficaces permettant de tirer des éléments de $\mathcal{P}_K$, pour tout K, et de calculer E.*

3. *Pour tout $K \in \mathcal{K}$ et $x \in \mathcal{P}_K$, étant donnés $T(K)$ et $E_K(x)$, il existe un algorithme efficace retrouvant x. Autrement dit, la connaissance d'une trappe permet d'inverser efficacement.*

4. *Pour tout $K \in \mathcal{K}$, aucun algorithme efficace ne peut calculer x à partir de la seule donnée de K et $E_K(x)$ avec probabilité de succès non négligeable. Autrement dit, sans connaissance de la trappe il est difficile d'inverser.*

La définition ci-dessus n'est pas entièrement formelle : les permutations et les éléments des domaines sont implicitement tirés suivant des distributions associées ; et négligeable fait implicitement référence à un paramètre de sécurité. D'autre part la définition fait appel à la notion d'algorithme « efficace ». Cela peut être exprimé par la notion d'algorithme probabiliste polynomial, qui est une manière (pertinente mais assez théorique) de formaliser la notion d'algorithme efficace.

En ce qui concerne la sécurité pratique, tout ce qui est efficace dans la définition ci-dessus doit pouvoir être calculé « rapidement », tandis que ce qui n'est calculable par aucune algorithme efficace ne doit pas pouvoir être calculé plus rapidement qu'une limite imposée par un paramètre de sécurité, au sens de la Section 2.3.

3.3 Chiffrement en boîte blanche

Le chiffrement en boîte blanche englobe plusieurs modèles de sécurité, moins élémentaires que les modèles présentés jusqu'ici. Nous nous contentons ici d'évoquer rapidement certains modèles, en particulier celui de l'incompressibilité, qui sera développé dans le chapitre 3.

De manière idéale, la cryptographie en boîte blanche cherche à protéger une implémentation contre un attaquant qui la connaît entièrement, avec des garanties comparables à celles que procure un composant matériel sécurisé. De manière concrète, on ne connaît pas aujourd'hui de technique qui atteigne pleinement ce but. Plusieurs compromis sont possibles, aussi bien dans le domaine de l'ingénierie logicielle que de la cryptographie.

Du côté cryptographique, qui nous intéresse, cela se traduit par différents modèles de sécurité. Le plus fort est celui de l'obfuscation au sens cryptographique, mais il reste hors de portée en pratique aujourd'hui. Un modèle moins ambitieux est l'irréversibilité : un attaquant ayant accès à l'implémentation d'un chiffrement ne sait pas l'utiliser pour déchiffrer. Cela revient à définir une permutation à trappe, telle qu'on l'a vue plus haut.

Si l'on souhaite une performance plus proche de la cryptographie symétrique, on peut considérer un modèle plus modeste, l'incompressibilité : un adversaire ayant accès à l'implémentation d'un chiffrement ne peut pas produire une implémentation significativement plus compacte du même chiffrement. Ce modèle implique qu'il est impossible pour l'attaquant d'extraire une clef maître courte, qui est l'intention originelle des constructions en boîte blanche. D'autre part, il implique qu'il est nécessaire pour un attaquant souhaitant produire un outil de déchiffrement illicite d'extraire la quasi-totalité du code, et non une clef ponctuelle, ce qui se prête mieux à des techniques d'obfuscation logicielle.

Résumé en français

1 Organisation du manuscrit

Les travaux que j'ai effectués pendant ma thèse portent sur des sujets assez variés. Dans ce manuscrit, j'en ai sélectionné quatre, à la fois pour leur intérêt propre, et parce qu'ils forment une suite relativement cohérente depuis la cryptanalyse symétrique jusqu'à la cryptanalyse asymétrique, en passant par la cryptographie en boîte blanche, qui est à certains égards intermédiaire.

Le premier chapitre porte sur la cryptanalyse, par auto-similarité et espaces invariants, des chiffrements symétriques Robin, iSCREAM et Zorro. Le second chapitre présente la cryptanalyse structurelle d'une construction nommée ASASA, dont les instances concrètes incluent à la fois des constructions symétriques et multivariées, dans le contexte de la cryptographie en boîte blanche. Le chapitre suivant montre comment réaliser de manière prouvée un des modèles de boîte blanche du chapitre précédent. Enfin du côté purement asymétrique, le dernier chapitre présente une cryptanalyse d'application multilinéaire. Les chapitres sont en ordre chronologique, sauf les deux derniers, pour des raisons de cohérence et facilité d'exposition.

Dans le résumé en français qui suit, chacun des chapitres est présenté brièvement. La dernière section mentionne mes autres travaux, suivis d'une liste des publications.

2 Cryptanalyse par auto-similarité des LS-Designs et Zorro

Les LS-Designs sont une famille de chiffrements par bloc présentée par Grosso, Leurent, Standaert et Varici à FSE 2014 [GLSV14]. La structure de ces chiffrements est simple : l'état interne est vu comme une matrice rectangulaire de bits. La fonction de tour se décompose en deux étapes L et S. L'étape L consiste à appliquer une même fonction linéaire à chaque ligne de la matrice. L'étape S consiste à appliquer une même boîte S à chaque colonne. L'intérêt de cette structure est qu'elle peut s'implémenter très facilement en bitslice (pour peu que ce soit le cas de la boîte S), et se prête naturellement à une implémentation masquée.

L'article d'origine inclut aussi deux instances concrètes, Robin et Fantomas, dont la première utilise des composants (fonction linéaire L et boîte S) involutifs. Par ailleurs des variantes avec tweak de ces deux chiffrements, nommées SCREAM et iSCREAM [GLS+14b], ont été proposées comme candidates à la compétition de chiffrement authentifié CAESAR [Com13].

Dans un travail effectué (en fait, fusionné) avec Gregor Leander et Sondre Rønjom [LMR15], dans un premier temps, nous montrons que la structure très forte provenant du LS-Design et des composants involutifs confère des propriétés surprenantes à Robin et iSCREAM : il existe une permutation des colonnes de l'état interne qui commute avec la fonction de tour.

On appelle « reliés » deux états internes liés par cette permutation, et « faible » un état interne relié à lui-même ; de même pour les messages, les chiffrés et les clefs, qui sont assimilés à des états internes. Alors deux messages reliés chiffrés avec deux clefs reliées produisent deux chiffrés

reliés. En particulier, un message faible chiffré avec une clef faible produit une chiffré faible : une clef faible peut être détectée avec forte probabilité à l'aide d'un seul message choisi. Ainsi dans un modèle à clef faible (ou reliée), l'indistinguabilité du chiffrement est cassée instantanément avec essentiellement un message choisi. Dans le cas de Robin, la densité de clefs faibles est de 2^{-32}, ce qui rend l'attaque significative en pratique.

L'attaque précédente ne recouvre pas la clef secrète, mais nous montrons aussi que si la clef est faible, alors le chiffrement inclut un sous-chiffrement : pour une clef faible, une certaine projection de la valeur du chiffré ne dépend que de la même projection des valeurs du message et de la clef. On en déduit une attaque qui retrouve la clef secrète en temps 2^{64} (au lieu de 2^{96} de manière générique pour une clef faible).

L'attaque ci-dessus repose sur la structure spécifique des LS-Designs—et aussi sur le choix des composants involutifs, et des constantes de tour. Mais il s'avère que la même attaque s'applique à un chiffrement très différent, Zorro [GGNPS13], également conçu pour avoir un faible surcoût en masquage. Contrairement aux LS-Designs, d'autres attaques classiques avaient déjà cassé Zorro. L'intérêt de notre contribution sur ce point est de montrer une autre instance de notre attaque, sur un chiffrement complètement différent. En effet Zorro est une variante d'AES, basé sur des octets et non des bits, dont l'innovation principale est une couche non-linéaire partielle. Cependant la même attaque fonctionne : on peut exhiber, non plus une permutation, mais une fonction linéaire plus générale sur $\mathbb{F}_{2^8}$ qui commute avec la fonction de tour. Les conséquences sont exactement les mêmes, en termes de clefs faibles, clefs reliées et recouvrement de clef. Il se trouve par ailleurs que la densité de clefs faibles est la même (2^{-32}).

Dans une second temps, on peut remarquer que si l'on s'intéresse à la variante de l'attaque avec clef faible sur des messages faibles, alors on est en présence d'une attaque par sous-espace invariant, c'est-à-dire qu'un certain sous-espace (affine) des clefs envoie un certain sous-espace des messages sur un certain sous-espace des chiffrés. Ce type d'attaque avait été introduit pour la cryptanalyse de PrintCIPHER en 2011 [LAAZ11], et n'avait pas été développé depuis. Puisque nous montrons deux nouvelles attaques de ce type, il est légitime de se demander comment les analyser ou les détecter. Nous proposons un algorithme générique qui recherche automatiquement les attaques par sous-espace invariant à partir d'une implémentation de la fonction de tour d'un chiffrement, avec une complexité directement proportionnelle à (l'inverse de) la densité du sous-espace invariant. En effet l'algorithme procède simplement par clôture, en devinant un élément du sous-espace invariant, et en clôturant l'espace qu'il génère avec les constantes de tour par va-et-vient à travers la fonction de tour. Une implémentation de cet algorithme générique a été rendue publique, et permet de détecter automatiquement les attaques précédentes (sous leur forme de sous-espace invariant).

3 Cryptanalyse structurelle d'ASASA

Le schéma ASASA a été introduit par Biryukov, Bouillaguet et Khovratovich à Asiacrypt 2014 pour concevoir des chiffrements en boîte blanche [BBK14]. Le constat de départ est que les constructions en boîte blanche originelles et leurs variantes immédiates cherchaient globalement à masquer des couches non-linéaires, notamment des boîtes S (couche S), par des couches affines aléatoires (couches A) composées en entrée et en sortie. On obtient alors une structure de la forme ASA, qui se prête trop facilement à la cryptanalyse. Par ailleurs, si les couches non-linéaires sont composées de boîtes S, alors un résultat de Biryukov et Shamir montre que la structure SASAS peut être cryptanalysée de manière structurelle [BS01], c'est-à-dire qu'un attaquant pouvant interroger la construction en entrée et en sortie peut retrouver efficacement tous les composants

internes (boîtes S et couches linéaires A). Par contre aucune attaque n'était connue sur ASASA.

D'autre part, la cryptographie multivariée, qui peut aussi être utilisée pour la construction de chiffrements en boîte blanche, tente également de masquer des composants non-linéaires structurés à l'aide de couches linéaires aléatoires externes. De ce point de vue aussi, une construction ASASA permettrait de gêner les attaques connues[4]. En conséquence, Biryukov, Bouillaguet et Khovratovich proposent des constructions ASASA pour la cryptographie en boîte blanche, avec des instances aussi bien symétriques que multivariées.

Dans un travail avec Patrick Derbez, Pierre Karpman et Pierre-Alain Fouque, nous proposons une cryptanalyse structurelle d'ASASA [MDFK15]. L'attaque est structurelle au même sens que la cryptanalyse structurelle de SASAS citée plus haut : c'est-à-dire qu'elle recouvre les composants internes de la structure ASASA, même s'ils sont uniformément aléatoires. Le terme structurel est employé par opposition aux attaques traditionnelles sur les primitives symétriques, où tous les composants sont connus de l'adversaire excepté la clef. Ici seule la *structure* des composants est connue (boîtes S, couches linéaires, etc.).

En particulier, notre attaque permet de casser de manière quasi-uniforme les instances dites en boîte noire, en boîte blanche faible, et une des deux instances multivariées de [BBK14], avec une complexité estimée de l'ordre de 2^{56} dans le pire des cas, pour un paramètre de sécurité de 128 bits. L'instance multivariée que nous n'attaquons pas est celle qui avait été déjà cryptanalysée par Gilbert, Plût et Treger à Crypto 2015 [GPT15], à l'aide d'une attaque par décomposition.

Notre attaque recouvre chaque couche d'ASASA successivement. La première couche est naturellement le point difficile. Notre technique exploite un défaut de degré observable si l'on devine correctement une partie de la dernière couche linéaire. Ce défaut de degré peut être traduit en équations quadratiques reliant certains éléments de la couche linéaire. On aboutit à un système qui peut se résoudre par relinéarisation, et permet de retrouver la dernière couche linéaire. Il est intéressant de noter que ce défaut de degré est exploitable aussi bien dans l'instance multivariée que dans les instances symétriques. En fait la même technique reste applicable tant que la construction n'est pas de degré algébrique maximal, et suite à notre attaque, une note de Biryukov et Khovratovich montre qu'elle permet encore d'attaquer ASASASA ou même SASASASAS avec des paramètres raisonnables [BC13].

Par ailleurs, dans le cas de l'instance multivariée, nous proposons aussi une seconde attaque indépendante, qui réduit le problème à une instance du problème « Learning Parity with Noise » (LPN) avec des paramètres faibles. Un algorithme de type BKW permet ensuite de casser le chiffrement [BKW03], légèrement moins efficacement qu'avec l'attaque précédente. L'idée naturelle pour faire apparaître LPN dans ce type de construction est d'assimiler heuristiquement les termes non-linéaires à une forme de bruit. Dans le cas de la construction multivariée que nous attaquons, pour se ramener à LPN, on calcule la dérivée seconde du chiffrement par rapport à deux vecteurs arbitraires, et le problème de retrouver un élément de la dernière couche linéaire se présente comme une variante de LPN. Cependant cette attaque est nettement plus spécifique que la précédente.

[4] En fait cette idée avait déjà été proposée par Patarin sous le nom de 2R, mais avait été victime d'attaques par décomposition. En résumé, celles-ci permettent de retrouver deux fonctions $f, g : \mathbb{F}_2^n \to \mathbb{F}_2^n$ à partir de leur composée $g \circ f$. Ici cette attaque est évitée à l'aide de polynômes quartiques aléatoires mélangés linéairement à la sortie du chiffrement.

4 Construction de primitives prouvables en boîte blanche

Nous avons mentionné plus haut le modèle d'incompressibilité en boîte blanche. Plusieurs constructions ont déjà été proposées dans ce modèle, ou de proches variantes [DLPR13, BBK14, BI15]. C'est le cas d'une des constructions ASASA de la section précédente. Cependant ces constructions offrent des garanties purement heuristiques.

Dans un travail avec Pierre-Alain Fouque, Pierre Karpman et Paul Kirchner [FKKM16], nous partons de l'idée qu'en s'autorisant AES comme sous-composant, on peut obtenir d'une part des garanties prouvables d'incompressibilité reposant uniquement sur la sécurité de l'AES, et d'autre part des implémentations logicielles très performantes en présence d'instructions matérielles AES (l'implémentation logicielle étant la préoccupation originelle de la boîte blanche). On réalise ainsi des constructions prouvables, avec des performances comparables aux solutions existantes, non prouvées.

En fait nous définissons deux modèles d'incompressibilité. Le modèle dit faible correspond essentiellement aux notions définies précédemment dans la littérature : il est impossible pour un attaquant ayant accès à l'implémentation d'un chiffrement d'en produire une autre significativement plus compacte, fonctionnellement équivalente sur la plupart des entrées. Le modèle fort, que nous introduisons, ajoute qu'un attaquant ayant accès à l'implémentation compressée ne peut même pas l'utiliser pour distinguer, avec probabilité de succès significative, le chiffrement de départ d'une fonction ou permutation aléatoire.

Notre travail part aussi du constat, déjà observé dans la littérature, dès même l'article fondateur de Chow et al., que les constructions en boîte blanche peuvent être utilisées pour l'encapsulation de clef plutôt que pour chiffrer chaque bloc de données [CEJO02a]. Cela amortit le coût de la boîte blanche, de la même manière que le chiffrement hybride pour le chiffrement à clef publique. Dans ce cas (et d'ailleurs d'une manière plus générale), il n'est pas nécessaire de se limiter au chiffrement par bloc comme unique primitive en boîte blanche.

Suite à cette observation, nous proposons deux primitives en boîte blanche. La première est un chiffrement par bloc en boîte blanche, classique dans ses garanties de sécurité. Ce chiffrement est prouvé, avec une hypothèse heuristique, dans le modèle d'incompressibilité faible, c'est-à-dire celui des constructions non prouvées existantes. Il permet donc une comparaison directe. La seconde construction est un générateur de clef, prouvable dans le modèle fort (et dans le faible). Nos implémentations montrent que ces deux primitives ont des performances comparables ou meilleures que les constructions précédentes.

L'idée générale de notre chiffrement par bloc est d'alterner des appels à AES avec une couche d'appels à une large table pseudo-aléatoire, également générée avec AES. Grossièrement, les appels alternés à AES rendent « aléatoires » les appels à la table, de sorte que si un adversaire ne retient pas une partie significative de la table, il est incapable de chiffrer la plupart des messages. La preuve repose sur des considérations combinatoires.

Le générateur de clef parallélise la même idée : l'entrée est utilisée pour générer une suite de blocs aléatoires en utilisant AES en mode compteur. Chaque bloc passe à travers une large table pseudo-aléatoire. Enfin cette nouvelle série de bloc est traitée par un extracteur d'entropie. En effet l'observation centrale est qu'un générateur de clef en boîte blanche correspond à un extracteur local d'entropie : si une implémentation compacte créée par un adversaire est significativement plus petite que la table pseudo-aléatoire d'origine, alors la table a une grande (min-)entropie conditionnée à la connaissance de l'implémentation compacte. Pour avoir une sortie uniforme aux yeux de l'adversaire, il suffit donc d'extraire cette entropie au sens d'un extracteur classique. Par ailleurs on souhaite que l'extracteur soit local, c'est-à-dire qu'il n'utilise qu'un faible nombre d'appels à la table, pour des raisons de performance. Dans le domaine des extracteurs locaux, un

article fondamental de Vadhan démontre un théorème modulaire qui permet de prouver notre construction dans le modèle fort [Vad04]. Nous donnons aussi une preuve combinatoire directe dans le modèle faible, dont les bornes permettent de choisir des paramètres plus performants.

5 Cryptanalyse d'application multilinéaire sur les entiers

Les applications multilinéaires sont une primitive asymétrique expressive et polyvalente, qui permet de réaliser une grande variété de constructions cryptographiques. Informellement, on appelle parfois cette primitive « crypto-complète » dans le sens où la plupart des constructions cryptographiques connues peuvent êtres réalisées à partir d'applications multilinéaires. Cela inclut aussi des constructions qu'on ne sait pas aujourd'hui réaliser autrement, comme les échanges de clefs multipartis sans interaction, ou, suite à un article fondateur de Garg, Gentry, Halevi, Raykova, Sahai et Waters, l'obfuscation générale de programme [GGH+13b]. Cette dernière application en particulier a suscité un grand intérêt de la communauté envers les applications multilinéaires.

Paradoxalement, à côté de cette riche variété de constructions fondées sur les applications multilinéaires, il existe très peu de constructions effectives. En particulier, aucune construction n'admet de réduction de sécurité vers un problème difficile standard. La première construction d'application multilinéaire, par Garg, Gentry et Halevi (GGH13), utilise des réseaux idéaux [GGH13a]. Elle a été suivie de près par une seconde construction sur les entiers par Coron, Lepoint et Tibouchi (CL13) [CLT13]. Ces constructions ont toutes deux été cassées en temps polynomial pour certaines applications, dont l'échange de clef multiparti sans interaction.

Cela a conduit Coron, Lepoint et Tibouchi à présenter une nouvelle version de leur construction à Crypto 2015 (CLT15), conçue pour résister aux attaques précédentes [CLT15]. Dans un travail commun avec Pierre-Alain Fouque [MF15], puis fusionné avec Jung Hee Cheon, Changmin Lee et Hansol Ryu [CFL+16], nous montrons que cette nouvelle construction peut elle aussi être attaquée en temps polynomial. En fait notre technique consiste à retrouver un paramètre secret de la construction, qui permet de ramener la sécurité de la nouvelle construction à celle de la version originale. De plus, notre attaque utilise uniquement les données essentielles de la clef publique, et fonctionne pour toutes les applications possibles de CLT15. Ainsi, tandis que CLT13 n'a pas d'attaque connue pour certaines applications, CLT15 peut être écarté entièrement.

Comme toutes les attaques sur les applications multilinéaires jusqu'ici, notre technique d'attaque évite les problèmes difficiles sur lesquels les constructions proposées voudraient heuristiquement reposer, typiquement des problèmes de réseaux. Dans notre cas l'observation cruciale est qu'une fonction bien choisie est $\mathbb{Z}$-linéaire sur un sous-domaine pertinent. L'attaque se conclut ensuite avec des techniques d'algèbre linéaire. L'attaque est polynomiale, et d'ailleurs instantanée pour une version optimisée de CLT15 proposée dans l'article d'origine. Nous proposons aussi une seconde attaque, probabiliste.

En prenant en compte l'attaque récente du schéma GGH15 par Coron, Lee, Lepoint et Tibouchi [CLLT16], toutes les propositions majeures d'application multilinéaire ont finalement été cassées pour l'échange de clef multiparti, qui est leur application la plus directe. Le statut d'autres applications, notamment l'obfuscation générale de programme, reste à déterminer dans la plupart des cas.

6 Autres travaux

Lors du séjour que j'ai effectué à l'ANSSI, j'ai publié deux articles de cryptanalyse symétrique. Le premier, en commun avec Thomas Fuhr, met au point une technique de rencontre au milieu avec

un nouveau type de précalcul sur la phase centrale de l'attaque [FM14]. Cela permet d'améliorer les attaques existantes sur KATAN, un chiffrement par bloc atypique utilisant des registres à décalage non-linéaires.

Un second article exhibe de faibles biais linéaires sur AEGIS, un candidat de chiffrement authentifié de la compétition CAESAR, avec des performances extrêmement élevées en software [Min14]. L'attaque permet en particulier de recouvrer des bits de message clair avec 2^{220} données sur la version à 256 bits de sécurité d'AEGIS, et surcoût en temps négligeable par rapport à la lecture des données. C'est une attaque entièrement théorique au vu de la quantité de données nécessaire. Il n'y a pas aujourd'hui d'autre attaque connue sur AEGIS, ni d'ailleurs sur les chiffrements qui s'en inspirent directement, MORUS et Tiaoxin.

Enfin dans un autre domaine, un travail en commun avec Yannick Seurin montre que l'avantage d'un distingueur entre une permutation aléatoire et le carré (ou l'autocomposition un nombre fixé de fois) d'une permutation aléatoire est en $\Theta(q/N)$, où q est le nombre de requêtes et N la taille de l'espace d'entrée [MS15]. Montrer $O(q^2/N)$ est bien sûr immédiat par le paradoxe des anniversaires, le but étant de démontrer une borne asymptotique linéaire en q. On en déduit assez directement que la composition de deux chiffrements identiques avec la même clef n'induit pas de perte de sécurité significative au point de vue de la sécurité prouvable. Ce travail a été suscité par une question ouverte dans un travail précédent de Chen, Lampe, Lee, Seurin et Steinberger.

7 Liste des publications

- Thomas Fuhr and Brice Minaud. Match box meet-in-the-middle attack against KATAN. In *Fast Software Encryption*, pages 61–81. Springer, 2014.
- Brice Minaud. Linear biases in AEGIS keystream. In *Selected Areas in Cryptography–SAC 2014*, pages 290–305. Springer, 2014.
- Gregor Leander, Brice Minaud, and Sondre Rønjom. A generic approach to invariant subspace attacks: Cryptanalysis of Robin, iSCREAM and Zorro. In *Advances in Cryptology–EUROCRYPT 2015*, pages 254–283. Springer, 2015.
- Brice Minaud and Yannick Seurin. The iterated random permutation problem with applications to cascade encryption. In *Advances in Cryptology–CRYPTO 2015*, pages 351–367. Springer, 2015.
- Brice Minaud, Patrick Derbez, Pierre-Alain Fouque, and Pierre Karpman. Key-recovery attacks on ASASA. In *Advances in Cryptology–ASIACRYPT 2015*, pages 3–27. Springer, 2015. Invited to the Journal of Cryptology.
- Jung Hee Cheon, Pierre-Alain Fouque, Changmin Lee, Brice Minaud, and Hansol Ryu. Cryptanalysis of the new CLT multilinear map over the integers. In *Advances in Cryptology–EUROCRYPT 2016*, pages 509–536. Springer, 2016.
  (Fusion de [MF15] et [CLR15])
- (En soumission) Pierre-Alain Fouque, Pierre Karpman, Paul Kirchner, and Brice Minaud. Efficient and provable white-box primitives. To appear in the proceedings of Asiacrypt 2016, 2016.

General Introduction

1 Symmetric and Asymmetric Cryptography

The primary object of cryptography is the design and analysis of secure communications. Modern cryptography is mainly concerned with electronic communication, and belongs to the field of computer science. Nevertheless it is distinct from computer security, which plays a complementary role.

An electronic communication, such as occurs on a computer network, may be decomposed into several layers. Low-level layers include physical and various network layers, while high-level layers bear the payload of the communication. Conceptually, each new layer is built on top of the underlying layers, often modeling their behavior, and assuming they function correctly. Cryptography studies the higher levels of this hierarchy. As such it deals with security in a partially idealized world, and generally disregards security flaws that stem from the execution context, such as implementation bugs, hardware failures, or viruses.

In this context, one the simplest and earliest goals of cryptography is confidentiality. Confidential messages must only be readable by their sender and receiver, or more generally any party in possession of a secret key. This definition naturally suggests military applications. However with the ever more widespread use of computer networks and internet, confidentiality has become a common requirement. It is necessary for applications such as e-commerce or online banking, where private information travels on a public network.

Confidentiality ensures that an encrypted message leaks no information about its content, except when the secret decryption key is known. However in many applications of cryptography, other standard cryptographic guarantees are required, such as integrity and authenticity. Authenticity allows the receiver to verify that the message was indeed sent by its purported sender. Integrity allows the receiver to verify that the message was not tampered with during transit.

The three notions of confidentiality, authenticity and integrity belong to the realm of symmetric cryptography. The term symmetric is used to reflect the fact that a unique secret is shared among the communicating parties. This one secret enables both encryption and decryption, authentication and verification. The situation is thus entirely symmetric between parties that share the secret. In particular, one may observe that authenticity only ensures that the message was written by one of the parties in possession of the secret, rather than any one specific party.

By contrast, asymmetric cryptography encompasses all cases where communicating parties do not share a common secret. So far we have seen the three notions of confidentiality, authenticity and integrity for symmetric cryptography. Asymmetric cryptography also allows to fulfill various security notions—in fact, a large number of them. For the purpose of this introduction we will only consider two common definitions.

The most basic asymmetric notion is perhaps that of an electronic signature. Similar to a physical signature, an electronic signature proves the identity of the signing party, in a way that anyone can verify. It is no longer necessary to know any secret information in order to verify a

signature, contrary to the (symmetric) case of authenticity. Anyone can verify a signature, but only one party can generate it, thus proving its identity.

The signature is generated using a private key, which only the signing party knows. Meanwhile a public key, generally accessible to anyone, allows to verify a signature. For this reason, asymmetric cryptography is also called public-key cryptography. In the same way, one can define public-key encryption schemes. In such a scheme, anyone can encrypt messages using a public key. Meanwhile, decryption requires a private key, secretly associated to the public key. We will later encounter other examples of asymmetric schemes, such as multipartite key exchanges.

In general, symmetric primitives fulfill relatively simple purposes, such as authenticity and confidentiality, but do so very efficiently. They can be used to encrypt large amounts of data. On the other hand, asymmetric primitives provide richer functionality sets, but are comparatively slow. They are used at key points to achieve critical security properties, while the bulk of communications is encrypted using symmetric primitives.

2 Cryptographic Notions

In this section we set out to describe some basic notions of cryptography. Our presentation shall be brief and incomplete: we will focus on those notions that are necessary to understand our work, or at least provide some context for it.

2.1 Primitives, Modes and Protocols

Many applications of cryptography, such as online payments, require communication between several entities over a number of messages. The precise description of the authorized sequence of messages is called a protocol. For example, when a user connects to a secure website using the HTTPS protocol, security properties such as authentication are enforced by the TLS protocol. The TLS protocol defines in a rigorous manner what type of message is authorized at each step of the protocol.

A security protocol such as TLS is built on top of more fundamental cryptographic elements, such as encryption schemes and signatures. An encryption scheme may in turn be built from an even more basic cryptographic brick, such as a block cipher. In that case, a mode of operation specifies how the block cipher is used to derive an encryption scheme. At the bottom of this multilayered composition of cryptographic elements, the most fundamental constructions, which do not rely on any other, are called cryptographic *primitives*. In the remainder, we will almost exclusively be interested in primitives, as opposed to modes or protocols.

2.2 Models and Proofs

Modern cryptography is (at least partially) underpinned by mathematics. Security notions that we have alluded to so far, such as signatures, can be formally defined. These definitions are often called models, because they interpret informal notions in a mathematical way. Moreover a given security notion may be captured by distinct models, depending on the required properties.

For a given security model, a cryptographic scheme aiming to satisfy the model may either be proven in that model, or simply be conjectured secure, if no proof is known. Symmetric and asymmetric primitives differ in this regard.

Symmetric primitives are (almost) never proven. They rely on efficient techniques, but do not lend themselves well to mathematical analysis (in some sense this is by design). On the contrary, asymmetric primitives rely on a strong mathematical structure in order to provide

richer usage properties. This structure may be exploited by attacks, and makes asymmetric primitives comparatively slow. On the other hand, thanks to this underlying structure, standard asymmetric primitives, such as public-key encryption schemes, are typically proven.

However the meaning of proof should be qualified. A security proof does not show that an arbitrary adversary is unable to attack the scheme, because an unbounded adversary can exhaustively try every private key. Even when considering bounded adversaries, it is not proven that it is impossible for the adversary to break the scheme, because such proofs seem out of reach at the moment[5]. Instead, security proofs show that if an adversary can efficiently break the scheme, then she can efficiently solve a reputedly hard problem, such as factoring large integers. The security guarantees afforded by a proof are meaningful, but not absolute.

In the case of some less common asymmetric primitives, such as multilinear maps, not only is there no known security proof, but it is not yet clear whether the primitives exist. In such a case, following a famous article by Impagliazzo [Imp95], one may describe and study a hypothetical world where these primitives exist, and one where they do not, without knowing which one we live in.

2.3 Security Parameter

A cryptographic scheme targets a security definition provided by a model, as explained earlier. The security model may include one or several security parameters, which express the hardness of breaking the scheme. In the simplest case, which is also the most common, the model contains a single security parameter, denoted by λ. This parameter means that an adversary against the scheme cannot succeed with less than 2^λ computing power, tallied either in number of encryptions or basic operations. The parameter λ is usually chosen in such a way that the required computing power is infeasible with current technology, including in the near future[6].

Cryptographic schemes, especially asymmetric ones, are often viewed as a family of constructions, one of whose parameters is λ. In that case, one may be interested in the asymptotic complexity of an attack, which provides a simple approximation of its practical complexity. In this context, it is generally expected that attacking a scheme is exponentially more expensive than using it.

2.4 Cryptanalysis

Cryptanalysis is the branch of cryptography that seeks to evaluate the security of schemes. This mostly amounts to finding attacks. Indeed, the choice of parameters of a cryptographic scheme must be such that all attacks are infeasible. However the nature of attacks is somewhat different depending on whether a security proof is available.

If no security proof is known, confidence in the security of a scheme generally only exists if the scheme belongs to a well-studied type of construction (such as block ciphers), for which the main attack vectors are believed to be known (linear and differential attacks, for example). In that case, the efficiency of these attacks is assessed, and the parameters of the scheme are chosen such that it is comfortably out of reach of attacks. The scheme itself may be designed in a way that facilitates its security assessment. Nevertheless dedicated attacks may still exist, and an in-depth analysis by independent experts, over a long period of time, is unavoidable.

[5] A canonical example of this limitation is the inability so far to solve the problem $P \neq NP$. But more generally, proving absolute (non-reductionist) lower bounds in a standard complexity model seems currently intractable, outside of "simple" bounds arising from information theory.

[6] For higher values of λ, it is also possible to argue that the required computing power is physically impossible.

If a security proof is available, two cases should be considered. Either the proof provides a tight reduction to a hard problem. In that case, attacking the scheme amounts to attacking the hard problem, and cryptanalysis belongs to general algorithmic as much as it does to cryptography. Examples include attacks on discrete logarithm or factorization. Otherwise, if the proof does not provide a tight reduction, the situation is similar to the unproven case: the concrete efficiency of the best attacks must be assessed, taking into account possible gaps between the scheme and the underlying hard problem. This is also the case should the security model fail to capture all desirable security properties.

Cryptanalysis consists in attacking cryptographic schemes. But indirectly, one of its main contributions is to draw a picture of the most relevant attacks against a given type of scheme or problem. If a scheme is able to resist these attacks, as well as a extensive cryptanalytic effort, then it may be regarded as secure. Since absolute lower bounds on adversaries (almost) never exist, this is what confidence in the security of a scheme is ultimately based on.

3 Useful Definitions

In this section, we provide a few relevant definitions. This is intended to make some informal statements from the previous sections more precise, as well as to provide a basis for the next chapters.

3.1 Block Ciphers

A block cipher is a symmetric primitive defined as follows.

Definition 4 (Block Cipher)**.** *Given key space $\mathcal{K}$, plaintext space $\mathcal{P}$, and ciphertext $\mathcal{C}$, a* block cipher *is a mapping $E : \mathcal{K} \times \mathcal{P} \to \mathcal{C}$ such that for all $K \in \mathcal{K}$, $x \mapsto E(K, x)$ is invertible.*

In most cases $\mathcal{P}$ and $\mathcal{C}$ have the same cardinality, and can be identified. A block cipher can then be regarded as a mapping from $\mathcal{K}$ into the permutations of $\mathcal{P}$. Typically, $\mathcal{P} = \mathcal{C} = \{0,1\}^n$ for some integer n, denoting the block size, *e.g.* $n = 128$. Similarly $\mathcal{K} = \{0,1\}^k$ for some integer k, denoting the key size.

Informally, the usual security model associated with a block cipher is as follows. No efficient adversary should be able to distinguish an oracle giving access to $E(K, \cdot)$ and its inverse, for uniform K, from an oracle giving access to a uniformly random permutation and its inverse. From the point of view of an adversary with no knowledge of the secret key, a good block cipher looks like a uniformly random permutation.

Other security models are possible. The related-key model, which shall be briefly mentioned in the first chapter, asks (at the very least) that no efficient adversary is able to distinguish $E(K, \cdot)$, $E(f(K), \cdot)$, and their inverses, for f belonging to some class of functions (typically constant additions) from two uniformly random permutations and their inverses.

In the same chapter, we will also mention tweakable block ciphers.

Definition 5 (Tweakable Block Cipher)**.** *Given key space $\mathcal{K}$, tweak space $\mathcal{T}$, plaintext space $\mathcal{P}$, and ciphertext space $\mathcal{C}$, a* tweakable block cipher *is a mapping $E : \mathcal{K} \times \mathcal{T} \times \mathcal{P} \to \mathcal{C}$ such that for all $K \in \mathcal{K}, T \in \mathcal{T}$, $x \mapsto E(K, T, x)$ is invertible.*

The above definition is equivalent to a block cipher with key space $\mathcal{K} \times \mathcal{T}$. The difference between key and tweak lies in the security model: the key is drawn uniformly independently of any adversary, whereas the tweak can in general be controlled by the adversary. The standard

security definition of a tweakable block cipher asks that for uniform K, the family parametrized by T of permutations $E(K, T, \cdot)$ and their inverse is indistinguishable for an efficient adversary from a family of uniformly random permutations and their inverse.

3.2 Public-Key Encryption Scheme

Chapters 2 and 3 will mention public-key encryption, and more precisely, trapdoor permutations. Public-key encryption schemes can be built from trapdoor permutations in a standard manner. We now define this primitive. Informally, a trapdoor permutation is a family of permutations with associated *traps*. Each permutation should be easy to compute, but difficult to invert, unless the associated trap is know.

Definition 6 (Trapdoor Permutation)**.** *Given key space $\mathcal{K}$, plaintext space $\mathcal{P}_K$, ciphertext space $\mathcal{C}_K$ (which may depend on the key), and trap space $\mathcal{T}$, a* trapdoor permutation *is a family of permutations $E_K : \mathcal{P}_K \to \mathcal{C}_K$, parametrized by key $K \in \mathcal{K}$, with the following properties :*

1. *There exists an efficient algorithm sampling K, together with associated trap $T(K) \in \mathcal{T}$.*
2. *There exist efficient algorithms sampling from $\mathcal{P}_K$, for all K, and computing E.*
3. *For all $K \in \mathcal{K}$ and all $x \in \mathcal{P}_K$, given $T(K)$ et $E_K(x)$, there exists an efficient algorithm recovering x. In other words, inverting is easy whenever the trap is known.*
4. *For all $K \in \mathcal{K}$, no efficient algorithm is able to compute x from knowledge only of K and $E_K(x)$, except with negligible probability of success. In other words, inverting is difficult without knowledge of the trap.*

The above definition is not entirely formal: permutations and elements are implicitly drawn according to some associated distributions; and "negligible" implicitly refers to a security parameter. The definition also uses the notion of an efficient algorithm. This can be captured by the notion of probabilistic polynomial time algorithm, which is a (meaningful but rather theoretical) way of formalizing the notion of efficient algorithm.

With regards to practical security, every "efficient" algorithm in the above definition should be "fast" in practice; meanwhile, whatever is not computable by any efficient algorithm should not be computable faster than some limit imposed by a security parameter, in the sense of Section 2.3.

4 Layout of the Thesis

During my thesis, I have worked on a variety of subjects. In this manuscript, I have selected four, based on their individual interest, and also because they form a relatively coherent sequence from symmetric cryptanalysis to asymmetric cryptanalysis, via white-box cryptography, which in some respects may be regarded as a middle ground between the two.

The first chapter presents self-similarity and invariant subspace cryptanalyses of symmetric ciphers Robin, iSCREAM and Zorro. The second chapter moves on to the structural cryptanalysis of the so-called ASASA construction, whose concrete instances include both symmetric and multivariate schemes, in the context of white-box cryptography. The third chapter shows how to provably achieve one of the white-box models of the previous chapter. Finally on the purely asymmetric side, the last chapter proposes a polynomial-time cryptanalysis of a recent multilinear map candidate construction. The chapters are presented in chronological order, except for the last two, for the sake of coherence and ease of exposition.

Notation

The following notation is used throughout this work.

$\triangleq$:	equality by definition.
$\mathbb{Z}_n$:	ring $\mathbb{Z}/n\mathbb{Z}$ of integers modulo n.
$\mathbb{F}_q$:	finite field of size q.
$\lvert S\rvert$:	cardinality of the set S.
$\binom{n}{k}$:	binomial for k elements among n.
log:	logarithm in base 2.
ln:	natural logarithm.
E_K or Enc_K:	encryption function with key K.

Chapter 1

Self-Similarity and Invariant Subspace Attacks

1.1 Introduction

In this chapter we show the existence of self-similarity attacks in some recent block cipher designs: the lightweight cipher Robin introduced at FSE 2014 as a concrete instance of the LS-design framework [GLSV14], the closely related CAESAR [Com13] candidate iSCREAM [GLS+14b][7], as well as the lightweight cipher Zorro presented at CHES 2013 [GGNPS13]. The self-similarity property we consider, where a linear map commutes with the round function, also entails the existence of invariant subspaces in the same ciphers, and leads to practical attacks in all cases.

Self-similarity properties were first formally defined in [BB02] to study alternative descriptions of AES, and later used in [BDLF10] to cryptanalyze the SHA-3 candidate *Lesamnta* and the lightweight cipher XTEA. Meanwhile invariant subspace attacks were introduced for the cryptanalysis of PRINTCIPHER [LAAZ11], and made no other appearance prior to our work. Since then, a few more cases have arisen [GJN+15, Røn16, GRR16].

Both of these attacks are unusual in that they rely on an unexpected form of symmetry in the round function, and yield attacks that are independent of the number of rounds. It is notable that neither attack surfaced until recently, nor have they been successfully applied to older designs[8]. This may be regarded as a consequence of the recent profusion of designs aiming to fulfill strong performance requirements, such as low hardware footprint (e.g. PRESENT [BKL+07], LED [GPPR11], KATAN [CDK09]), low memory consumption on small embedded processors (e.g. ITUBee [KDH13], SPECK [BSS+], PRIDE [ADK+14]), low latency (e.g. PRINCE [BCG+12]) or ease of side-channel protection (e.g. Zorro [GGNPS13], LS-Designs [GLSV14]). Ciphers aiming to fulfill these requirements tend to feature innovative designs: they may rely on simpler round functions, or minimal key schedules. While in most cases, guarantees against traditional linear or differential attacks are still offered, the simpler structure of many of these ciphers may lend itself to new attacks, such as self-similarity and invariant subspace attacks.

We now give a brief overview of the core property at play in our attack. The self-similary we exploit is the existence of a linear map commuting with the round function. Of course the existence of such a map is quite suprising in itself, but it also leads to a number of powerful attacks. Define *related* inner states (resp. plaintexts, ciphertexts or keys) as pairs of states (resp. plaintexts, ciphertexts or keys) linked by the linear map. Define *self-related* inner states

[7] Due to our attacks, iSCREAM has now been dismissed from the CAESAR competition.
[8] However a generalization of invariant subspaces was recently applied to AES [GRR16].

(resp. plaintexts, ciphertexsts or keys) as those that are related to themselves. *Weak* keys are self-related keys. Then related keys map related plaintexts to related ciphertexts. In particular, weak keys map self-related plaintexts to self-related ciphertexts. As a result in a weak (or related) key setting, indistiguishability can be broken using essentially a single chosen plaintext. In all ciphers we target, the density of weak keys turns out to be 2^{-32}, making the attack quite relevant in practice.

Moreover in all ciphers considered, as a consequence of the special type of self-similarity property involved, our analysis also shows that whenever a weak key is used, the ciphers contain an embedded subcipher. That is, a certain projection of the ciphertext only depends on a projection of the plaintext and key. This property has been called a "linear factor" in an older line of work [RM85, CE85, Eve87], which aimed to disprove the existence of such a property in DES. To the best of our knowledge, our attacks are the first actual occurrence of this phenomenon within "serious" block cipher designs, although only in a weak key setting. For all ciphers considered, the embedded subcipher allows key recovery in 2^{64} operations whenever a weak key is used, with minimal data.

In the case of LS-designs, we furthermore present a second attack, based on S-box-dependent invariant subspaces, without an underlying self-similarity. We obtain a new set of weak keys with the same properties as above, although much less dense. This yields a weaker attack. However it also uncovers a different security caveat of LS-Designs, which is nicely symmetric with the main attack, as it swaps the roles of the linear and nonlinear layers in the attack.

Since our attack applies to block cipher designs as different as iSCREAM and Zorro, a natural question is whether it is possible to detect this type of attack in a generic, reusable manner. Note that finding invariant subspaces is enough, as (most) commuting linear maps of the type we consider will give rise to a non-trivial invariant subspace. In the last part of the chapter, we present a generic algorithm that is able to detect invariant subspaces. The running time of this algorithm depends on the block size of the primitive and the density of the existing invariant subspaces. In particular, it is especially efficient if relatively large invariant subspaces exist. As the impact of an invariant subspace increases with its dimension, this can be seen as detecting stronger attacks significantly faster than minor attacks. The algorithm is able to detect the invariant subspaces from our previous attacks in less than a day on a standard desktop computer[9].

Layout of the Chapter.

In Section 1.2, we provide a description of LS-designs, including our targets Robin and iSCREAM. In Section 1.3, we define self-similarity and invariant subspace attacks, then give the outline of our attack. In Sections 1.4, 1.5 and 1.6, we develop our attacks against LS-designs, introduce a particular self-similarity property, the resulting invariant subspaces, and finally describe a different invariant subspace attack not underpinned by self-similarity. In Section 1.7, we apply the same attacks to Zorro. In Section 1.8, we present our generic algorithm for detecting invariant subspaces. Finally, in Section 1.9, we conclude with a discussion of our results and outline interesting open problems.

[9]The source code of our tool is available at `http://invariant-space.gforge.inria.fr`.

1.2 Description of LS-Designs, Robin, and iSCREAM

1.2.1 LS-Designs

LS-designs were introduced by Grosso, Leurent, Standaert and Varici at FSE 2014 [GLSV14]. We refer the interested reader to their article for a detailed presentation of LS-designs and their design rationale. For our purpose, a brief technical description suffices.

An LS-design is a block cipher encrypting n-bit plaintext blocks using a n-bit key. The inner state of the cipher, as well as the plaintext, ciphertext, and key, are all represented as an $r \times c$ bit array, with r the number of rows and c the number of columns. A concrete LS-design is parametrized by the following components:

- A choice of r and c. The size of the key and message blocks is $n = r \cdot c$.
- An r-bit S-box s.
- A bijective linear map ℓ on c-bit vectors, called the L-box.
- A number of rounds t.
- A choice of k-bit round constants $C(i)$ for $1 \leq i \leq t$.

In order to encrypt a given n-bit plaintext block, the plaintext is first loaded into the inner state of the cipher, and the master key is added in (all additions are bitwise `XOR`s). Then a round function is applied successively for rounds 1 to t. At that point the cipherext is equal to the inner state. The round function at round i proceeds as follows:

1. The round constant $C(i)$ is added to the inner state.
2. The S-box s is applied to each column of the state.
3. The L-box ℓ is applied to each row of the state.
4. The n-bit master key K is added to the state.

1.2.2 Notation

When dealing with LS-designs, we will always use the previous notation; that is:

r	the number of rows of the state.
c	the number of columns.
n	the size of the state; that is, $n = r \cdot c$.
s	the r-bit S-box.
S	the S-box step; that is, the application of s on each column of the state.
ℓ	the $c \times c$ binary matrix representing the linear layer, identified with the corresponding linear map on $\mathbb{F}_2^c$.
L	the L-box step; that is, application of ℓ on each row of the state.

1.2.3 Robin

In [GLSV14], two concrete LS-designs are proposed, Robin and Fantomas. The idea behind Robin is that both the S-box and L-box are involutive. This allows the same circuitry to be reused when computing these components and their inverse operation, i.e. when encrypting and decrypting. This saves valuable space on embedded devices when both encryption and

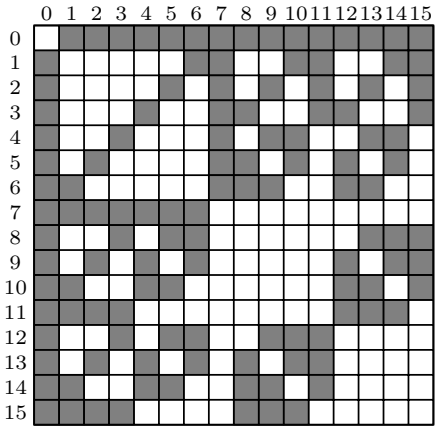

Figure 1.1: Matrix representing the L-box of Robin and iSCREAM. Dark cells stand for 1's and white cells for 0's.

decryption capabilities are required. The trade-off is that involutive components have more structure, resulting in a slightly higher number of rounds to reach the same security level as an LS-design based on non-involutive components.

Robin strictly fits within the LS-design framework recalled in the previous section. As such it can be fully described by the following parameters:

- The inner state of Robin has 8 rows and 16 columns, resulting in 128-bit blocks and a 128-bit key.
- The 8-bit involutive S-box is given in Appendix 1.
- The 16-bit involutive L-box is depicted as a 16×16 binary matrix on Fig. 1.1.
- The number of rounds is 16.
- At round i (starting from 1), the round constant $C(i)$ is zero outside of the first row, where it is equal to $\ell(i)$, with ℓ the L-box matrix.

1.2.4 iSCREAM

SCREAM and iSCREAM [GLS+14b] are two authenticated ciphers closely related to LS-designs. In fact iSCREAM is essentially a tweaked version of Robin, together with a Tweakable Authenticated Encryption (TAE) mode of operation [LRW02]. Meanwhile SCREAM is similar to Fantomas, with a different linear layer. The TAE mode of operation requires a tweakable block cipher [LRW02]. Accordingly, the difference between the block cipher underlying iSCREAM and Robin stems from the introduction of a 128-bit tweak T into the (previously non-existent) key schedule.

In the remainder we focus on weaknesses of the block cipher on which iSCREAM is built, independently of the mode of operation. We may abuse notations and write iSCREAM to mean its underlying block cipher.

This block cipher can be described as an LS-design, except for the fact that during the key addition phase, instead of adding in K every round: at odd rounds, $K + T$ is added; while at even rounds, $T \lll_c 1$ is added, where $T \lll_c 1$ denotes a circular shift of the columns of T by one column towards the left. The combination of two rounds is called a step. Beside that, iSCREAM can be described by the following parameters:

- The inner state of iSCREAM has 8 rows and 16 columns, resulting in 128-bit blocks and a 128-bit key.
- The S-box and L-box are those of Robin.
- The number of rounds depends on the required security level. The original article lists six variants. However the primary recommendation for iSCREAM as per CAESAR requirements is 12 steps (24 rounds) [GLS+14a]. A secondary recommendation claiming related-key security has 14 steps (28 rounds). Since our attacks are essentially independent of the number of rounds, we omit other variants.
- At round i (starting from 1), the round constant $C(i)$ is zero outside of the first row, where it is equal to $27 \cdot i$ modulo 256 (affecting only the first 8 bits of the row).

1.3 Self-Similarity, Commuting Maps and Invariant Subspaces

We start by recalling the concept of self-similarity, instantiate it with commuting linear maps, and relate the result to invariant subspaces. We then explain how these properties can be used in an S-box-independent manner. In that case, we also show that involutive commuting maps imply the existence of an embedded subcipher, enabling key recovery attacks. These techniques will later be applied to LS-designs in general and to Robin and iSCREAM in particular (Section 1.4), then to Zorro (Section 1.7).

1.3.1 Self-Similarity Properties and Linear Commutant

In [BB02] and [BDLF10], self-similarity in general is defined as:

Definition 7 (Self-similarity in a block cipher)**.** *For a fixed block cipher E, let $E_K(x)$ denote the ciphertext block resulting from the encryption of plaintext block x under key K. A self-similarity relation is given by invertible and efficiently computable mappings ϕ, ψ, θ such that:*

$$\forall K, x: \quad \theta(E_K(x)) = E_{\psi(K)}(\phi(x))$$

What we are interested in is the case where $M = \phi = \psi = \theta$ is a linear map. This situation will arise if the cipher follows a generalized Even-Mansour structure where key-independent round functions F_i alternate with the addition of a fixed key K (i.e. no key schedule); and M commutes with the round functions F_i. This last condition is very demanding; but this is precisely what happens in both Robin and Zorro, despite their difference in structure[10]. The following lemma sums up the attack.

Lemma 1. *Consider a block cipher composed of round functions F_i separated by addition of a fixed key K. Suppose there exists a linear map M such that M commutes with the F_i's. Then:*

$$\forall x: \quad M(E_K(x)) = E_{M(K)}(M(x))$$

In particular, if $K = M(K)$:

$$\forall x: \quad M(E_K(x)) = E_K(M(x))$$

[10] We expand on why this might be the case in Section 1.9.

In our applications, M will be involutive, so we focus on the case $i = 1$. In the remainder, whenever two plaintext blocks (or ciphertext blocks, or inner states, or keys) satisfy $x_2 = M(x_1)$, we say that they are *related*. If a plaintext block (or ciphertext block, or inner state, or key) is related to itself, we say that it is *self-related*. A *weak key* is a self-related key. In short, our attack states that weak keys map self-related plaintexts to self-related ciphertexts; while related keys map pairs of related plaintexts to pairs of related ciphertexts.

1.3.2 Invariant Subspace Attacks

Invariant subspace attacks were introduced and applied to PRINTCipher in [LAAZ11]. We recall the basic principle here.

Consider a n-bit block cipher with round function F_K consisting of a key addition and a SP layer $F : \mathbb{F}_2^n \to \mathbb{F}_2^n$. That is, F_K is defined by $F_K(x) = F(x + K)$. Assume the SP-layer F is such there exists a subspace $A \subseteq \mathbb{F}_2^n$ and two constants $u, v \in \mathbb{F}_2^n$ with the property:

$$F(u + A) = v + A$$

Then, given a (round) key $K \in u - v + A$, i.e. $K = K_A + u - v$ with $K_A \in A$, the following holds:

$$F_K(v + A) = F(v + A + u - v) = F(u + A) = v + A$$

i.e. the round function maps the affine subspace $v + A$ onto itself. If all round keys are in $u - v + A$, in particular if identical round keys are used as in LS-designs and Zorro, then this property is iterative over an arbitrary number of rounds.

In the case where an identical key is added in every round (there is no key schedule), a key is said to be weak iff it belongs to $u - v + A$. Whenever a key is weak, plaintexts in $v + A$ are mapped to ciphertexts in $v + A$, breaking plaintext confidentiality. The number of weak keys is the cardinality of A.

In order to detect whether an unknown key is weak, it is enough to encrypt one plaintext in $v + A$, and test whether the resulting ciphertext is in the same space. Indeed, over the set of all keys, false positives will occur with the same frequency as true positives, and can be discarded with a second chosen plaintext.

The commuting linear map M from the previous section can be interpreted from the invariant subspace perspective. Indeed, if we let $A = \ker(M^i + Id)$ for any i, A is an invariant subspace[11]. Of course self-similarity is a stronger property stemming from a stronger requirement on the cipher.

1.3.3 S-box-Independent Setting

The previous attack can be specialized to the case where the cipher is a substitution-permutation network (SPN), whose round function F_i consists of an S-box layer with identical S-boxes, a linear map L, addition of a round constant $C(i)$, and addition of a fixed key K. From the invariant subspace (resp. self-similarity) perspective, we are interested in subspaces (resp. linear maps) that traverse (resp. commute with) each of these components.

It is quite apparent that the main roadblock is the nonlinear S-box layer. However even in a generic setting where we do not take into account a particular choice of S-box, any permutation of the S-box inputs will commute with the S-box layer (due to S-boxes being identical). We shall

[11] It may be that a non-trivial commuting matrix leads only to trivial invariant subspaces, as evidenced by the 2×2 binary matrix with rows [01] and [11]. However if M is involutive, $\ker(M + Id)$ is at least half of the space.

say that we are in the S-box-independent setting iff the linear map acts as a permutation on S-box inputs. Note the we were careful in our phrasing to encompass the case where the S-box layer may be only partial, *i.e.* S-box inputs do not cover the whole state, in order to account for Zorro later on. In that case the linear map is required to act as a permutation on S-box inputs, but is subject to no special requirement on the rest of the state.

In terms of invariant subspaces, this setting corresponds to subspaces containing those vectors whose coordinates belonging to the same cycle in the permutation are equal; that is, subspaces that only require S-box inputs to be equal to some other input, or independent. We call such spaces *equality spaces*. Note that these are vector subspaces and no longer affine subspaces. Our strongest attacks actually occur in this setting.

As for constant and key addition, asking that their addition commutes with M amounts to asking that they belong to $\ker(M + Id)$. Now it remains to find permutations that commute with the linear layer. An efficient algorithm to do so is provided in [LMR15, Appendix G]. The invariant subspace variant seems more difficult, as we do not know an algorithm able to efficiently enumerate equality spaces that traverse a linear map.

1.3.4 Embedded Subcipher

By themselves, self-similarity and invariant subspace properties stemming from a commuting linear map break plaintext confidentiality. In addition, if the commuting map M is involutive and is S-box-independent as per the previous section (which will be the case in our applications), efficient key recovery is possible. That is because the cipher embeds a subcipher, which is nontrivial as long as $\ker(M + id)$ is not the full space. Indeed, if M is a permutation, the projection of the ciphertext on the fixed points of M only depends on the projection of the plaintext and key on the same space. More generally if M is not a permutation, the role of the fixed points of M in the previous statement is played by a supplementary space of $\mathrm{Im}(M + id)$.

We now formalize and prove the above statement. We want to encompass the case where the S-box layer is only partial. We ask that the linear map commuting with the round function present itself as a block matrix, where the first block covers all S-box inputs, and is only allowed to permute them as before; while the second block is a general bijective linear map with no further restriction (if the S-box layer is not partial, this second block is simply nonexistent).

Let us then denote $S = \ker(M + Id)$ (the self-related space), and define $I = \mathrm{Im}(M + Id)$. Observe that because M is involutive, I is a subspace of S (indeed, $(M + Id)(M(x) + x) = M(M(x) + x) + M(x) + x = M(M(x)) + x = 0$). Our claim is the following.

Lemma 2. *Consider an SPN whose round function is composed of:*

- *A potentially partial S-box layer;*
- *A linear layer L;*
- *Addition of a fixed key K;*
- *Addition of rounds constants.*

We assume the existence of an involutive *linear map M commuting with all of these components. Furthermore M is a block matrix, where the first block covers all S-box inputs, and only permutes them; while the second block is a general bijective linear map with no further restriction.*

Let $S = \ker(M + Id)$ and $I = \mathrm{Im}(M + Id)$. We already know that $x \in S$ and $K \in S$ implies $E_K(x) \in S$. But in addition:

$$\forall K_1, K_2, x_1, x_2 \in S : \quad \left.\begin{array}{c} x_1 + x_2 \in I \\ K_1 + K_2 \in I \end{array}\right\} \Rightarrow E_{K_1}(x_1) + E_{K_2}(x_2) \in I$$

Proof. We show that if two self-related inner states s_1 and s_2 satisfy $s_1 + s_2 \in I$, this remains true after every component of the round function:

- S-box layer: let us restrict our attention to the part E of the state affected by S-boxes. We view E as a subspace of the states, and we group bits corresponding to the same S-box input together to view E as a space on $\mathbb{F}_2^b$, where b is the bit size of an S-box. We know that M acts as a permutation P on E. Since P is involutive, E can be decomposed into 3 disjoint subspaces $E = F + A + B$, where F are the fixed points of the permutation, and A and B are mapped to each other by P. Let Z be the space of vectors whose value is zero on F in the previous decomposition.

  We claim that $I = S \cap Z$. Indeed $I \subseteq S$ because P is involutive; and $I \subseteq Z$ because $P(x) + x$ equals zero on fixed points of P; so $I \subseteq S \cap Z$. Conversely choose $x \in S \cap Z$; then let x' be the projection of x on A (parallel to $B + Z$); then $x = x' + P(x')$. To see this, use the decomposition along $Z + A + B$: x and $x' + P(x')$ are both zero on Z; they are equal on A, as x and x' are equal on A by definition and $P(x')$ is zero; and finally they are equal on B because they are equal on the rest and both sides are self-related. Thus we have shown $I \supseteq S \cap Z$; so $I = S \cap Z$.

  Now observe that if s_1 and s_2 satisfy $s_1 + s_2 \in Z$, this remains true after the S-box layer, since belonging to Z for the sum of two states only means that the columns of these two states in F are equal. We already know that belonging to S is preserved by the S-box layer; so in the end $s_1 + s_2 \in S \cap Z = I$ implies that this is still true after the S-box layer.

- Linear layer: assume $s_1 + s_2 \in I$; then $L(s_1) + L(s_2) \in L(I)$. However $L(I) = I$, because L commutes with $M + Id$, so the property of belonging to the image of $M + Id$ is stable by L. Indeed, $\forall x : \quad x \in I \Leftrightarrow x = P(y) + y \Leftrightarrow L(x) = P(L(y)) + L(y) \Leftrightarrow L(x) \in I$.

- Constant addition: this step does not affect the value of $s_1 + s_2$.

- Key addition: it is assumed that $K_1 + K_2 \in I$, so $s_1 + s_2 \in I$ implies $(s_1 + K_1) + (s_2 + K_2) \in I$. □

As a consequence, provided the key is in S (i.e. weak), and we encrypt self-related plaintexts, the value of the ciphertext on any supplementary space F of I in S only depends on the value of the plaintext and key on the same space (we denote this supplementary space by F, as in the case where M is a permutation on the full state, the fixed points of M is a valid choice). This allows us to try and guess the value of the key on F independently of the rest of the key, by encrypting any self-related plaintext.

The number of key bits we are thus allowed to guess independently of the rest is the dimension of F, which is $\dim(S) - \dim(I) = 2 \cdot \dim(S) - n$ (this number is positive because M is an involution). Another viewpoint is that the cipher contains an embedded subcipher operating on F with a $\dim(F)$-bit key: we can project self-related plaintexts, ciphertexts and keys on F parallel to I and obtain a well-defined new cipher. Note that this embedded subcipher may lend itself to further attacks; this is a direction we have not investigated, as we believe ciphers are sufficiently broken at that point.

1.4 Invariant Permutation Attack on LS-Designs

Notation. In the S-box-independent setting of Section 1.3.3, for an LS-design, a permutation of S-box inputs is simply a permutation of the columns of the state. Let us write P for such a permutation. We always denote by the lowercase p its effect on a single row. Thus, P is the application of p on each row of the state. We identify p with the corresponding $c \times c$ permutation matrix. We adopt notations from Section 1.2.2.

The particular structure of LS-designs means that P commutes with L iff p commutes with ℓ. This is still a strong requirement, but we expect the L-box of an LS-design to have some structure in order to provide a good branch number, especially if it is involutive. In the case of Robin for instance, the linear layer is built from a Reed-Muller code and provides plenty of structure. Applied to LS-designs, Lemma 1 becomes:

Lemma 3. *For an LS-design, assume there exists a permutation P with the following properties:*

- *P commutes with L.*
- *$P(C(i)) = C(i)$ for all round indices i.*

Then for any plaintext message m:

$$\mathrm{Enc}_{P(K)}(P(m)) = P(\mathrm{Enc}_K(m))$$

In particular, if $K = P(K)$:

$$\mathrm{Enc}_K(P(m)) = P(\mathrm{Enc}_K(m))$$

Note that the identity permutation trivially satisfies the above requirements. Hereafter we always assume P is non-trivial. If $\mathrm{ncycles}(p)$ is the number of cycles of p, weak keys form a proportion $2^{-r\cdot(c-\mathrm{ncycles}(p))}$ of all keys (namely, those keys whose columns are equal on each cycle of p).

Key Recovery

The previous attack breaks plaintext confidentiality. In addition, when P is involutive, efficient key recovery is possible, as shown in Section 1.3.4. We propose a simpler statement and proof below, dedicated to LS-designs.

Lemma 4. *Consider an LS-design, and assume there exists a permutation P with the same requirements as in Lemma 3. Also assume that P is an involution. Consider a weak key $K = P(K)$. Denote by F the set of fixed points of P.*

Take any self-related plaintext $m = P(m)$. Then the value of the ciphertext $\mathrm{Enc}_K(m)$ on the columns in F only depends on the value of m and K on the same columns.

Proof. Since P is an involution, all of its cycles have length 1 or 2. Hence we can partition the columns of the state into three subsets F, A, B, such that P is the identity on F, and maps A and B into each other. Take any self-related message m that is zero on F. Then the linear layer maps m to a self-related state $L(m)$ that is also zero on F. To see this, write $m = m_A + m_B$, where m_A is equal to m on A, and zero elsewhere, and likewise m_B is equal to m on B and zero elsewhere. Then $P(m_A) = m_B$, hence $P(L(m_A)) = L(m_B)$ by commutativity of P and L. Since P is the identity on F, this implies that $L(m_A) + L(m_B)$ is zero on F, so $L(m)$ is zero on F.

Thus, if $m = P(m)$ is zero on F, so is $L(m)$. By linearity, this implies that if m_1 and m_2 are self-related and equal on F, then so are $L(m_1)$ and $L(m_2)$. Thus, the property that two

self-related states are equal on F goes through the linear layer. This property automatically goes through the S-box layer since it is column-wise. Since the same key and round constants are added to both sides, they have no impact. Hence this property goes through the whole cipher. □

For clarity, we emphasize that this is only a special case of Section 1.3.4.

Permutation Characteristic

Instead of considering only permutations P commuting with L, we can naturally look for pairs of permutations (P, Q) such that $L \cdot P = Q \cdot L$. We denote this by $P \to Q$, representing the fact that if two inner states are related by P before the linear layer, then after the linear layer they are related by Q.

From there we can hope to build a form of characteristic $P_0 \to P_1 \to P_2 \to P_3 \to \dots$ The commutative case in the previous section corresponds to $P \to P$. Note that the set of permutations P such that $Q = L \cdot P \cdot L^{-1}$ is a permutation forms a group. Also note that if L is involutive, $P \to Q$ is equivalent to $Q \to P$: indeed $L \cdot P = Q \cdot L$ implies $P^{-1} \cdot L = L \cdot Q^{-1}$, implies $L \cdot Q = P \cdot L$: hence any transition $P \to Q$ yields an iterative characteristic of length at most 2.

A particularly interesting case occurs whenever $P \to P^\alpha$ for some $\alpha \neq 0$. Indeed, in that case we automatically have a cyclic characteristic $P \to P^\alpha \to P^{\alpha^2} \to \dots \to P^{\alpha^i} = P$. Moreover the attack from Lemma 3 goes through with exactly the same requirements on the key and round constants (namely they are self-related by P).

Application to Robin

Applying our attack to Robin amounts to finding a permutation p commuting with the matrix ℓ in Fig. 1.1, such that P leaves all round constants $C(i)$ invariant. More generally, as pointed out just above, we can actually look at transitions $P \to Q$, i.e. permutations p, q such that $\ell \cdot p = q \cdot \ell$. It turns out there are 720 such transitions, and all of them are of the form $P \to P^{-1}$. Moreover 76 of these permutations are involutive, and hence commute with L.

Recall that the round constants of Robin are defined as $C(i) = \ell(i)$ on the first row, and zero on the others, for $1 \leq i \leq t$. Hence we want $p(\ell(i)) = \ell(i)$, which amounts to $p(i) = i$ by commutativity. Since i ranges from 1 to 16, what we are looking for is simply permutations leaving the first 5 columns fixed. It turns out there exists exactly one such permutation, namely the involutive permutation P_0 switching columns 8, 9, 10, 11 respectively with columns 12, 13, 14, 15. Looking at Fig. 1.1, one can indeed see that permuting the rows and columns of the matrix of ℓ by p_0 leaves the matrix invariant, which is the same as saying p_0 commutes with ℓ.

With P_0, weak keys are simply keys whose last four columns are equal to the previous four. In particular the proportion of weak keys is 2^{-32}. Furthermore P_0 leaves the first 8 columns fixed, so Lemma 4 shows that for self-related plaintexts, the first 8 columns of ciphertexts only depend on the first 8 columns of plaintext and key. This makes it possible to guess the value of the master key on the first 8 columns independently of the rest of the key. This means 64 bits of the key can be guessed separately; then the remaining 64 bits are symmetric through P_0, so only 32 bits remain to be guessed. Thus the full key can be recovered in time complexity 2^{64} by encrypting any self-related message. This may yield a few solutions, which can be checked against any other plaintext/ciphertext pair.

Beside P_0, two other permutations P_1 and P_2 commuting with L leave the round constants invariant up to the very last round (cf. Table 1.1). This means related plaintexts are mapped

Table 1.1: Permutations p_0, p_1 and p_2. Fixed points are omitted.

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
p_0									12	13	14	15	8	9	10	11
p_1					8	9	10	15	4	5	6					7
p_2					12	13	14	11				7	4	5	6	

to related inner states after 15 encryption rounds; followed by the final constant addition, S-box layer, L-box layer, and key addition. The final linear layer can be reversed, and the resulting states will agree on pairs of columns transposed by P on which $C(16)$ is equal. In both cases, there is one such pair, so self-related keys with respect to P_1 and P_2 can still be detected easily by encrypting a few self-related plaintexts, reversing the last linear layer, and checking that these two columns agree.

Permutations P_1 and P_2 both leave 8 columns fixed and hence yield an attack with essentially the same properties as P_0. Actually some key bits can be recovered faster than with P_0 thanks to the one-round differential at the end, but this involves the symmetric part of the key (that is, outside the fixed points of the permutation) and thus the overall key recovery time is still 2^{64}.

Application to iSCREAM

Recall that iSCREAM and Robin share the same linear layer. Round constants only affect the first eight columns of the state, and so we are looking for permutations commuting with L and leaving the first eight columns unchanged. As a matter of fact, there exists exactly one such permutation, namely the same permutation P_0 as above, which switches the last four columns of the state with the previous four.

Another difference between Robin and iSCREAM is the number of rounds, but that is actually irrelevant for our attack. The last difference is the presence of a tweak in the key schedule. Recall that at odd rounds, $T + K$ is added, while at even rounds, $T \lll_c 1$ is added, where T is a 128-bit tweak. In a chosen-tweak scenario, we can simply set T to zero, or any other value such that T and $T \lll_c 1$ are invariant by P. Then the attack against Robin from the previous section applies to iSCREAM essentially unchanged, with the same consequences.

A small variant of our attack is also possible when using P_0 as the commuting permutation. What we truly want is that $K + T$ and $T \lll_c 1$ should be self-related. This amounts to asking that columns 8, 9, 10, 11 should be equal to columns 12, 13, 14, 15. Since $T \lll_c 1$ is a column-wise shift of T by one column towards the left, this means that columns 9, 10, 11, 12 of T should be equal to columns 13, 14, 15, 0. Note that there is no condition on column 8 of T. As a consequence, for $K + T$ to be self-related for some choice of T, it is enough to ask that columns 9, 10, 11 of K should be equal to columns 13, 14, 15. Indeed in that case, we can fix T to be all-zero, except for column 8 which can take any value: exactly one such choice of T will satisfy that $K + T$ is self-related. Thus we obtain a larger set of weak keys (with ratio 2^{-24}), at the cost of requiring 2^8 chosen-tweak messages in order to detect whether a fixed unknown key is weak.

In addition, some variants of iSCREAM claim related-key security. If two keys are related by P_0, then our attack applies immediately without any weak key requirement, following the first consequence of Lemma 3. That is, related plaintexts are mapped by the related keys to related ciphertexts. Thus it is easy to check whether a pair of keys is related, and the cipher is broken in a strong sense.

Generalizations of the Permutation Attack

There appears to be a few simple ways in which our attack could be generalized. We discuss them briefly here.

We could consider a probabilistic version of the attack. Instead of requiring $L \cdot P = P \cdot L$, we could consider P's such that the kernel of $L \cdot P - P \cdot L$ is almost the full space. In the case of Robin or iSCREAM, this would incur a cost at least 2^{-8} per round.

Another natural extension is to consider cases where all round constants are P-invariant except for the last few rounds (or first few rounds). Then our attack goes through most of the encryption process, and eventually yields a differential attack on the remaining rounds. When encrypting self-related plaintexts, this differential attack turns into an inner differential.

1.5 Invariant Equality Space Attack

In this section we study invariant subspaces for LS-designs following the S-box-independent setting of Section 1.3.3. We begin by defining equality spaces, and then present our results on Robin and iSCREAM. On the way, we will relate it to the commuting permutations from the previous section.

1.5.1 Equality Spaces

As always, we use notations from Section 1.2.2. We always view n-bit vectors as an $r \times c$ matrix. In Section 1.3.3, we defined equality spaces in general terms for an SPN; we now provide a more specific definition suited to LS-designs.

Definition 8. *A subspace E of $\{0,1\}^c$ is an* equality space *iff there exists a partition of $\{0,\ldots,c-1\}$ such that E is the set of vectors whose values on coordinates belonging to the same class in the partition are equal.*

The dimension of E is the number of classes of the partition. By E^r we denote the set of n-bit states whose columns belonging to the same class in the partition underlying E are equal. Equivalently, this means that every row of the state belongs to E, hence the notation E^r. By extension we also call E^r an equality space. The point of this definition is that equality spaces are preserved by the S-box layer. The question is to determine which equality spaces are also preserved by the linear layer. That is, we are looking for equality spaces $E \subset \{0,1\}^c$ such that $\ell(E) = E$.

As pointed out in Section 1.3.1, when a permutation P commutes with L, the equality space defined by the cycles of P is preserved by the linear layer. The idea is that equality spaces preserved by the linear layer do not necessarily stem from a commuting permutation. Conversely, commuting permutations are an interesting special case, since they lead to a stronger property: indeed, when considering equality spaces rather than permutations, we are looking at a property of a single state, and there is no equivalent to the property that distinct related plaintexts are mapped to related ciphertexts; there is also no equivalent to Lemma 4. Meanwhile, Lemma 3 becomes:

Lemma 5. *For an LS-design, assume there exists an equality space E such that:*

- $\ell(E) = E$.
- $C(i) \in E^r$ *for all round indexes i.*

Then for any key K and plaintext message m:

$$\textit{If } K \in E^r \textit{ and } m \in E^r \textit{ then } \mathrm{Enc}_K(m) \in E^r$$

The lemma trivially holds if E is the full space $\{0,1\}^c$; hereafter we assume this is not the case. Then we have an attack in the weak key setting, where weak keys are keys in E^r. Hence the proportion of weak keys is $2^{-r\cdot(c-\dim(E))}$.

1.5.2 Variants of the Attack

Essentially the same extensions as in Section 1.4 apply to equality spaces.

Characteristics: if the image $F = L(E)$ of an equality space E is also an equality space, we write $E \to F$. As with permutations, we can aim to build a characteristic $E_0 \to E_1 \to \dots$ over several rounds. Note that the set of equality spaces is closed under intersection, and as a direct consequence, the set of equality spaces E such that $L(E)$ is an equality space is also closed under intersection. If L is involutive, $E \to F$ is equivalent to $F \to E$, so characteristics are automatically cyclic.

Probabilistic attack: Instead of asking $F = L(E)$, we can require the dimension of the quotient space $F/L(E)$ to be small.

Differential ending: If all round constants are in the required equality spaces except for the last few (or first few) rounds, it may be possible to cover the remaining rounds with an inner differential characteristic. Indeed in the case $E \to E$, the equality space attack may be seen as an all-zero inner differential attack.

Differential attack: the entire attack itself may be transposed into the differential world, at the expense of becoming probabilistic. Consider a state difference living in E^r with $L(E) = E$. Then at each round, require that the S-box layer preserves this equality; that is, the output of some S-boxes which receive equal input, should remain equal. Note that if E stems from an involutive permutation commuting with L, the columns corresponding to fixed points of the permutation can be set to a zero difference: this will be preserved by the linear layer (cf. the proof of Lemma 4). This attack avoids key and round constant requirements, at the cost of much lower probability, and hence high data requirements. In practice this would lead to a weaker attack against Robin than truncated differential product trails in the original article [GLSV14], because the branch number is 8 and the non-fixed points of P_0 involve 8 S-boxes.

1.5.3 Application to Robin and iSCREAM

Since Robin and iSCREAM share the same linear layer L, we consider them together. We enumerated all equality spaces E such that $L(E)$ is an equality space (there are around 2^{33} partitions of 16 elements, so this is feasible), and analyzed the results.

Our first observation is that there are many more *well-behaved* equality spaces E (in the sense that $L(E)$ is also an equality space), than *well-behaved* permutations P (in the sense that $Q = L \cdot P \cdot L^{-1}$ is also a permutation). Namely, there are 720 well-behaved permutations for L, while there are 30162 well-behaved spaces of dimension 8 or more. Even if we remove from this list spaces that are an intersection of larger well-behaved spaces (and thus could have a chance of indirectly resulting from well-behaved permutations), 7746 well-behaved spaces remain.

Recall that L is involutive, so any transition $E \to F$ (i.e. $L(E) = F$ with E and F two equality spaces) yields a cyclic characteristic $E \to F \to E$. Hence all well-behaved spaces belong to cycles of length 1 or 2. The aforementioned 7746 intersection-reduced well-behaved spaces of dimension at least 8 form 2506 cycles of length 1 (that is, $E \to E$) and 2620 cycles of length 2

(that is $E \to F \to E$). Thus equality spaces offer considerably more potential attacks, depending on round constants.

However, all equality spaces compatible with actual round constants for Robin minus the last round, and hence directly usable in an attack, stem from commuting permutations. There exist four such spaces: three of them correspond to permutations P_0, P_1 and P_2 from Table 1.1, and the last one is a space of dimension 8 resulting from the composition of any two of the previous permutations (any combination yields the same permutation or its inverse). As for iSCREAM, the only well-behaved space compatible with round constants is the one resulting from P_0. Thus, our previous attack is not improved. Moreover, the largest well-behaved spaces have dimension 12 and all stem from involutive permutations (there are 15 of them). The largest well-behaved equality spaces not stemming from a well-behaved permutation have dimension 10. This may be interpreted to mean that the strongest phenomenon is due to commuting permutations.

Thus for both Robin and iSCREAM, the equality space induced by P_0 is the only equality space that goes through the whole cipher, including the last round. This space has dimension 96 over $\mathbb{F}_2$.

1.5.4 A note on Fantomas and SCREAM

The matrix L of Fantomas is a permutation of the lines and columns of the matrix of Robin. As a consequence, they have the same number of well-behaved permutations and spaces. However we found no cycle among well-behaved spaces of Fantomas of dimension 6 or more (lower dimensions would yield very weak attacks); and no characteristic of length more than 2. Hence Fantomas seems safe from this attack.

The same is true for SCREAM. However, it is worth noting that there exists no well-behaved permutation for the matrix of SCREAM, while we found 5404 well-behaved spaces of dimension 8 or more.

1.6 A Second Invariant Subspace Attack on LS-Designs

In this section we present a different invariant subspace attack on LS-designs, which may be regarded as a form of dual of the previous attack. This attack does not stem from an underlying permutation; nor does it have an equivalent for Zorro. Thus, this section is specific to LS-designs, and takes advantage of their particular structure: namely, the fact that LS-designs not only rely on a layer of identical S-boxes, but also on a layer of identical L-boxes.

Due to the S-box-indenpendent setting, the invariant space we have uncovered is automatically preserved by the S-box layer. It is natural to ask if, conversely, something similar can be done with the L-box layer. That is, we are now going to look for subspaces that are automatically preserved by the L-box layer.

This gives us more freedom, since we can leverage linearity. Essentially, if all columns of the state live in the same linear subspace, this will remain true after the linear layer (in [LMR15, Appendix D], we show that this is in fact the most general property generically preserved by the linear layer); whereas in the previous case, we were limited to equality spaces. Beside this difference, the attack is essentially a dual version of the previous one, reversing the roles of the L-box and S-box layers.

1.6.1 Description of the Attack

In the previous attack, we searched for equality spaces $E \subset \{0,1\}^c$ on the rows of the state such that $\ell(E) = E$. Instead, we are now interested in general linear subspaces $A \subset \{0,1\}^r$ on the columns of the state such that $s(A) = A$. Once again, if A is a linear space on the columns (or one of its cosets), we denote by A^c the set of states whose columns all belong to A.

The core of the attack is the following: assume $s(A) = A$ for some linear space A. If the inner state lies in A^c, this will remain true after the S-box layer. Moreover, this property is automatically preserved by the linear layer. Indeed, the linear layer of an LS-design is not truly "line-wise": precisely because the same linear map is applied to each row, the linear layer may be seen as directly adding together column vectors. From this point of view, it becomes clear that if all columns lie in the same linear space A, this remains true after the linear layer.

Thus we are still within the invariant subspace framework, and follow the corresponding strategy: we choose A such that all round constants belong to A^c, and we consider a weak key scenario by requiring that the key also lie in A^c. If these requirements are fulfilled, plaintexts in A^c are mapped to ciphertexts in A^c.

More generally, we can consider cosets of linear spaces (i.e. affine spaces) rather than just linear spaces: indeed, as long as each coordinate at the output of ℓ is the sum of an odd number of coordinates at the input, the linear layer still preserves the property that all columns belong to a fixed coset. The following lemma sums up the attack.

Lemma 6. *Let u, v, w be r-bit vectors, and A be a linear subspace of r-bit vectors. Assume the following conditions hold:*

- *The S-box s maps all vectors in $u + A$ to vectors in $v + A$.*
- *Either $v = 0$ or all rows of the matrix of ℓ have an odd number of 1's.*
- *The columns of all round constants are in $w + A$.*
- *The columns of the key are in $(u + v + w) + A$.*

Then any plaintext in $(u+w)+A$ is encrypted into a ciphertext in $(u+w)+A$ (and conversely).

Weak keys are keys in $(u + v + w) + E$. This means a proportion $2^{-c\cdot(r-\dim(A))}$ of keys is weak.

1.6.2 Application to Robin and iSCREAM

In the case of Robin, the second condition in Lemma 6 is automatically true. In order to satisfy the third condition (round constants), since round constants only affect the first row of the state, we require that the r-bit vector denoted by 1, with 1 on the first row and 0 elsewhere, belongs to E. To instantiate the attack, it remains to look for affine spaces whose direction contains the vector 1, that are mapped by the S-box to affine spaces with the same direction.

It turns out the largest such spaces have dimension 3, and are mapped into themselves. We list all six choices in Table 1.2. Since these spaces have dimension 3, and the state has 8 rows and 16 columns, a proportion $2^{-16\cdot 5} = 2^{-80}$ of keys are weak. This means our attack is considerably weaker than the first one against Robin. By comparison, a generic multi-target time-memory trade-off with 2^{48} memory would lead to key recovery for the same proportion of keys. Of course our attack requires no memory or table lookup.

Table 1.2: Six affine spaces of dimension 3 invariant through s.

Values in A								Dir(A)		
00	01	26	27	84	85	a2	a3	01	26	84
18	19	7c	7d	9e	9f	fa	fb	01	64	86
28	29	32	33	8a	8b	90	91	01	1a	a2
3c	3d	5e	5f	b2	b3	d0	d1	01	62	8e
44	45	66	67	c8	c9	ea	eb	01	22	8c
4e	4f	54	55	6c	6d	76	77	01	1a	22

We now turn to iSCREAM. Recall that its S-box is the same as that of Robin, and round constants still only affect the first row of the state. We want both $K + T$ and T to live in the same coset, so we require T to lie in $(u + v + w + A)^c$, and K to lie in A^c. In our actual attack we have $u = v$ and $w = 0$ so in the end, we can set the tweak to zero (or any value in A^c), and the attack goes through with the same parameters as before.

1.6.3 Taking Advantage of the iSCREAM Tweak Schedule

In the case of iSCREAM, it is possible to leverage the tweak schedule to create a trade-off between the ratio of weak keys and the number of chosen-tweak messages required to detect a weak key. To simplify notations, we explain this technique using vector spaces; it extends to their cosets in a straightforward manner. Assume we have two vector spaces A and B with $S(A) = B$. As before, we assume $1 \in A$ and $1 \in B$ so that round constants belong to A^c and B^c. Since S is involutive, we have $S(B) = A$, so $A \to B \to A$ is a characteristic for the the S-box.

In order for this characteristic to traverse encryption, we need $K+T \in A^c$, and $T \lll_c 1 \in B^c$, which is equivalent to $T \in B^c$. For this it is enough to ask $K \in A^c + B^c = (A + B)^c$. Indeed in that case, write $K = K_A + K_B$ with $K_A \in A^c$ and $K_B \in B^c$. Then for $T = K_B$, we have $K + T \in A^c$ and $T \in B^c$, which is precisely what we want. Of course the key is unknown to the attacker, so she cannot compute T in this way. Instead, she can try every value in the supplementary space of A^c in $(A + B)^c$ (which is smaller than B^c, if only because $1 \in A \cap B$). For exactly one such value of the tweak, every plaintext in A^c will be encrypted to a ciphertext in B^c.

Now the question is to find two spaces A and B as above. Actually we look for cosets of linear spaces with the same properties, since the linear layer of iSCREAM also preserves these cosets. In summary, we look for affine spaces $u + A \neq v + B$ such that $S(u + A) = v + B$, and 1 belongs to $A \cap B$.

It turns out the largest such spaces have dimension 3. There are 11 such spaces (counting only 1 for $u + A \to v + B$ and $v + B \to u + A$), listed in Appendix 2. Furthermore, 8 of these spaces satisfy $\dim(A + B) = 5$, which is the maximal possible value since 1 belongs to $A \cap B$. Thus $K \in (A + B)^c$ yields a ratio of weak keys of $2^{-c\cdot(r-\dim(A+B))} = 2^{-48}$.

In order to detect whether a key is weak, one needs to encrypt a message for each tweak in the supplementary of A^c in $(A+B)^c$, which is of dimension $2 \cdot c$, hence 2^{32} chosen-tweak messages are required (for a random key and a given choice of the tweak, a false positive has probability only 2^{-80}, and can be discarded by one additionnal chosen-tweak message). Finally, once a weak key is detected in this way, we know $K + T \in A^c$ for one specific T, hence $K = T + A^c$, so only $2^{c\cdot\dim(A)} = 2^{48}$ possibilities remain for the value of the key.

1.6.4 Variants of the Attack

It seems natural to consider a probabilistic version of the attack, where instead of requiring that every vector in $u + A$ be mapped by the S-box to a vector in $v + A$, we only require *most* of them to comply. If only x elements in $u + A$ are not mapped to $v + A$, the probability to pass an S-box is $1 - x/2^r$. The cost for each round is then $(1 - x/2^r)^c$. In the case of Robin, there is no A of dimension 4 with $x < 3$, so there does not appear to be an obvious interesting probabilistic version of the attack.

1.7 Commuting Permutation and Invariant Subspace for Zorro

1.7.1 Description of Zorro

The block cipher Zorro was introduced at CHES 2013 [GGNPS13]. Like LS-designs, the design goal is to offer a cipher that can efficiently be made resistant to side-channel attacks through masking [PR13]. This is achieved by two main techniques: first, a carefully constructed 8-bit S-box; and second, an AES-like structure where S-boxes are only applied on the first row of the state.

The 128-bit state is represented as a 4×4 array of 8-bit cells. The round function applies the following transformations:

- `SubBytes`: A fixed 8-bit S-box is applied to the first row of the state.
- `AddConstant`: At round i, the constants i, i, i and $i <<< 3$ are added to the four cells of the first row (from left to right).
- `ShiftRow`: This step is identical to AES. Row i, counting from zero, is shifted by i cells to the left.
- `MixColumns`: This step is again identical to AES. A fixed 4×4 circulant matrix on $\mathbb{F}_{2^8}$ is applied to each column of the state. The matrix is the same as that of AES.

Four consecutive rounds are called a step. After each step, the 128-bit master key is simply added to the inner state: there is no key schedule. Encryption consists in key addition, followed by 6 steps (24 rounds), each followed by key addition.

1.7.2 Prior Cryptanalyses

Many attacks have been carried out against Zorro, mostly differential or linear in nature [GNPW13, WWGY14, RASA14, BODD+, Sol14]. The best attack in [BODD+] is a differential attack requiring 2^{41} data and time complexity 2^{45} to break the full cipher. Our attack is of a different nature: it holds in the weak key setting (with 2^{96} weak keys out of 2^{128}), requires minimal data and time, and is independent of the number of rounds. In [LMR15, Appendix F], we show that our attack can be readily extended to Zorro-like ciphers, similar to [BODD+]. Nevertheless the point of our work on Zorro is mainly to show another application of the main attack, beside LS-designs (for which there was no priori cryptanalysis).

1.7.3 Self-Similarity and Invariant Subspace

We are interested in an S-box-independent commuting linear map, as in Section 1.3.1. To simplify, we focus on a single round: commuting with every round is a sufficient condition to

commute with every step. Thus we are looking for a linear map M acting as a permutation on the S-boxes, and commuting with the linear layer.

Since there are only four S-boxes, there are only 24 choices for the permutation. In fact, because the constant added to the fourth S-box is different from the others, we impose that this S-box should remain fixed by the permutation, leaving only 6 possibilities. In this way, our linear map will automatically commute with both the S-box and constant addition layers.

For each of the 6 permutation choices on the first 3 S-boxes, the set of linear maps behaving as this particular permutation on the first 4 cells, and independently on the other cells, is itself a vector space. Furthermore the commutant of the linear layer is naturally a vector space. Thus, it suffices to intersect these two spaces to find a solution, if it exists.

It turns out there exists exactly one solution, for the permutation swapping the first and third S-boxes, and leaving the other two fixed. This solution is given in Appendix 3, together with the resulting invariant subspace. This subspace has dimension 12 over $\mathbb{F}_{2^8}$, that is, 96 over $\mathbb{F}_2$. Hence the proportion of weak keys is 2^{-32}.

In [LMR15, Appendix F], we show how to enumerate all invariant subspaces for Zorro, and deduce that the previous space is in fact the only invariant subspace (in the S-box-independent setting). The strategy used to enumerate spaces extends naturally to any SPN with a partial S-box layer of only a few S-boxes per round.

1.7.4 Key Recovery

Since the commuting map we have uncovered is involutive, the key recovery strategy from Section 1.3.4 applies to Zorro.

As a consequence, once a key is recognized as weak, 64 bits of the key can be guessed independently of the rest using one chosen plaintext (any self-related plaintext). Indeed, the part of the key in I only influences the part of the ciphertext in I. After these 64 bits have been recovered by brute force, only 32 bits remain to be guessed, due to the key being weak. Thus key recovery requires only one chosen plaintext and a time complexity of 2^{64} offline encryptions.

1.8 A Generic Algorithm to Detect Invariant Subspaces

In this section we present our algorithmic approach to detect invariant subspaces in a generic manner, followed by experimental results.

1.8.1 A Generic Algorithm

In this section we present a simple and entirely generic probabilistic algorithm able to discover invariant subspaces for a given round function. The algorithm gives instant results for vector subspaces, and is able to discover affine subspaces in time proportional to their density. Despite its simplicity, this algorithm is enough to automatically discover all invariant subspace attacks elaborated upon in the previous sections.

The algorithm will identify minimal invariant subspaces and thereby identify invariant subspace attacks automatically. However, further analysis usually allows to significantly improve upon the attacks recovered automatically by the algorithm and gain further insights in the structure of the detected weakness. Furthermore, as the expected running time is determined by the density of invariant subspaces, it might well be that not all possible attacks are detected. Thus, for the moment, this generic algorithm cannot be used to fully exclude the existence of invariant subspaces.

Identifying Minimal Subspaces

Assume we are given a permutation $F : \mathbb{F}_2^n \to \mathbb{F}_2^n$. Here F could be a (keyless) round of a block cipher or a cryptographic permutation (like Keccak-f). Our goal is to find affine subspaces $u + A \subset \mathbb{F}_2^n$ such that:

$$F(u + A) = v + A$$

for some $v \in \mathbb{F}_2^n$.

Our algorithm is based on the following trivial observation.

Lemma 7. *Assume $u + A$ is an affine subspace such that $F(u + A)$ is also an affine subspace $v + A$. Then for any subset $X \subseteq A$, the linear span of $(F(u + X) - v) \cup X$ is contained in A.*

The idea is to first guess one possible offset u' of the affine space to be found and use $v' = F(u')$. Next, we guess a one-dimensional subspace of A, denote this by A_0. The algorithm will succeed if and only if $u' + A_0$ is contained in $u + A$.

1. We compute A_{i+1} from A_i as:

$$A_{i+1} = \text{span}\{(F(u' + A_i) - v') \cup A_i\}$$

2. If the dimension of A_{i+1} equals the dimension of A_i, we found an invariant subspace and exit.

3. If not, we continue with step 1.

Thus, the idea is to start with what we denote *nucleon* of A and map it using F until it stabilizes. In the case that our initial guess was wrong and $u' + A_0$ is not contained in some non-trivial invariant subspace we will end up with the full space after at most n iterations of the above.

Note that it is not necessary to really map the complete spaces A_i using F but a randomly chosen subset of relatively small size is enough for our purpose and significantly speeds up the process.

If the largest invariant subspace of F has dimension d, the algorithm will detect this space (or any invariant subspaces of this space) after an expected number of $2^{2(n-d)}$ guesses for A_0 and u'. Thus, in this basic form, the algorithm becomes quickly impractical. However, in the case of round functions of a cipher (or a cryptographic permutation) that differ by round constants only, its running time can be greatly improved as described next.

Knowing the Nucleon

For block ciphers with identical round keys or cryptographic permutations, we actually have a very good idea about the nucleon we want to be included in the space A, namely the round constants. More precisely, we consider round functions $F_i : \mathbb{F}_2^n \to \mathbb{F}_2^n$ that differ only by the addition of constants, i.e.

$$F_i(x) = F(x) + c_i$$

for $c_i \in \mathbb{F}_2^n$, where for simplicity we assume $c_0 = 0$. We are looking for affine subspaces $u + A$ that are mapped to $v + A$ by all round functions. In particular

$$F_0(u + A) = F(u + A) = v + A$$

and

$$F_i(u + A) = F(u + A) + c_i = v + A$$

which implies

$$v + A = c_i + v + A$$

and thus $c_i \in A$. Thus, given the situation as above, any subspace that is invariant under all round functions must necessarily contain the linear span of all round constants c_i.

For the algorithm outlined above this has significant consequences. Here, the only thing we have to guess is the offset. Therefore, the expected number of iterations of the algorithm is reduced from $2^{2(n-d)}$ to 2^{n-d}.

Moreover, after running the algorithm for m iterations with randomly chosen guesses for the offset, the probability that an invariant subspace of dimension d is not detected by the approach is given by

$$p_{m,n,d} := \left(1 - 2^{n-d}\right)^m$$

which can be approximated by

$$\log p_{m,n,d} \approx -m2^{d-n}.$$

The Algorithm

Algorithm 1: CLOSURE

Input: function F, nucleon A, offset u.
Output: a non-trivial invariant subspace, if one exists.

```
v ← F(u)
StableCount ← 0
while StableCount < N do
    Pick a random x ←$ u + span{A}
    if F(x) − v ∈ span{A} then
        StableCount = StableCount + 1
    else
        Add F(x) − v to A
        StableCount ← 0
Return u + span{A}
```

For offset u and nucleon A, the above procedure outputs the smallest affine subspace containing $u + \text{span}\{A\}$, that is mapped to a coset of the same space by F (with high probability). The algorithm depends on a global parameter N that controls the risk of error. Namely, when the algorithm exits, elements of $u + \text{span}\{A\}$ are mapped to $v + \text{span}\{A\}$ with probability greater than $1 - 2^{-N}$. This probabilistic result is enough for an invariant subspace attack to go through even for moderate choices of N.

Guessing the Offset

If we are actually looking for stable *vector* spaces rather than affine spaces, as will be the case in the S-box independent setting described in Section 1.3.3, guessing the offset is not needed: we can choose zero as the offset. Then the algorithm above finds the smallest invariant subspace instantly.

Table 1.3: Experimental Results: Here n is the block size and $d_{0.001}$ is the smallest dimension of an invariant subspace that has a probability to exist upper bounded by 0.001

Primitive	n	Dimension found	$d_{0.001}$	Running Time (h)
LED	64	-	34	24
NOEKEON [DPAR00]	128	-	98	40
Fantomas	128	-	98	40
Robin	128	96	-	22
iSCREAM	128	96	-	22
Zorro	128	96	-	<1

In the general case where we are looking for any (affine) invariant subspace, we need to guess one offset u belonging to the affine space we are searching for. Then we can run the procedure above to find the generated invariant subspace, if it exists (otherwise, the algorithm will simply output the full space). If the space we are looking for has dimension d, guessing such an offset u by brute force will require 2^{n-d} tries on average. Of course we just require *one* invariant subspace; so in general 2^{n-d} can be replaced by the density of vectors belonging to (non-trivial) invariant subspaces.

Each iteration of the algorithm requires Gaussian reduction to determine whether a certain n-bit vector belongs to some subspace, amounting to n^2 operations. Hence the overall running time to find an invariant subspace of dimension d is roughly $n^2 \cdot 2^{n-d}$. Thus if n is large, the above approach will only work if $n - d$ is relatively small, or more generally the density of invariant subspaces is large. The case where n is small is also useful in order to find invariant subspaces through a single S-box: this is how we found spaces in Appendix 2 (after making the algorithm deterministic and exhaustive, which is affordable for small n).

1.8.2 Applications

We applied the algorithm to the block ciphers Zorro, Robin, Fantomas, LED and NOEKEON, as well as to the CAESAR candidate iSCREAM. We chose $N = 50$ to be very conservative. We ran the algorithm with approximately 2^{34} iterations for each primitive, stopping earlier in the case where an invariant subspace was detected. The results are summarized in Table 1.3.

For LED, NOEKEON and Fantomas, no invariant subspaces were detected given our limited iterations. In that case, Table 1.3 indicates the dimension $d_{0.001}$ of the largest invariant subspace that has a probability to exist upper bounded by 0.001. More precisely, if x denotes the codimension of the largest invariant subspace, each random guess of an offset has probability 2^{-x} of falling into this subspace. After T tries, the probability of not having found the subspace is thus $(1 - 2^{-x})^T \approx e^{-T2^{-x}}$. We want this probability to be $1/1000$ within $T = 2^{32}$ tries, which yields $x = 32 - \log(\ln(1000)) \approx 30$, so $d_{0.001} \approx n - 30$. Thus it is unlikely that invariant subspaces of dimension above 98 exist for NOEKEON. However, the existence of smaller subspaces cannot be excluded with high probability by our results.

For Zorro, Robin and the CAESAR candidate iSCREAM the largest invariant subspace has dimension 96 out of 128, i.e. density 2^{-32}. Thus the time complexity is expected to be 2^{32} Gaussian eliminations on 128×128 binary matrices. Our experiments confirm this estimation. Discovering the invariant subspace took 22 hours on a single desktop PC equipped with an Intel Xeon Core i7 with 12 virtual cores used in parallel. The invariant subspaces discovered by the algorithm are exactly those that were uncovered by the analysis of the previous sections (in

particular they all stem from a commuting linear map).

In the case of Zorro, we chose to use a single round as target function, rather than the four rounds separating key addition. It turns out many cosets of the invariant subspace in Appendix 3 are sent to another coset by a single round (namely, all cosets stemming with offsets where cells 0 and 3 are equal). Our generic approach discovers this fact and the associated subspace instantly, hence the "< 1" time in the previous table.

1.9 Discussion

In this chapter, we have presented a unified cryptanalysis of several ciphers based on invariant properties traversing the cipher under certain conditions, while providing generic tools for this type of attack. Our attacks are able to break lightweight ciphers Robin, iSCREAM and Zorro in a practical setting.

Our attacks from sections 1.5 and 1.6 are quite similar in principle. The state of an LS-design is a rectangular array. A fixed line-wise operation is performed in each direction. Each attack looks for properties of the inner state that would be *structurally* preserved in one direction (in the sense that this does not depend on the specificities of the S-box or linear layer), that would happen to also be preserved in the other (this time due to the particular choices of S or L).

In the case where the generic direction is linear, any linear space is preserved, and under some conditions any coset; if it is nonlinear, only equality spaces are preserved. In [LMR15, Appendix C and D], we prove that these are in fact the most general properties structurally preserved in each direction, so our attacks fully realize the program outlined in the previous paragraph. It remains an open question whether a similar attack could in some way combine information from both directions; that is, neither direction would preserve the invariant property in a fully generic way.

Concerning our first attack on LS-designs from sections 1.4 and 1.5 (encompassing both invariant permutations and invariant equality spaces), the structure of the linear map is a key component. It seems unlikely that the attack could succeed in cases where the linear layer is not involutive. Indeed, as shown by the matrices of SCREAM and Fantomas, even in the presence of a large number of well-behaved equality spaces, it appears that iterative characteristics do not occur by accident. By contrast, if the linear layer is involutive, any well-behaved equality space (or permutation) yields a cyclic characteristic of length at most 2; and indeed, in the case of Robin and iSCREAM, thousands of iterative characteristics exist. Of course, the matrix of Robin and iSCREAM has much more structure than a generic involutive matrix.

It is quite striking that exactly the same attacks exist on Zorro, despite its quite different structure (byte-oriented vs. bit-oriented, partial S-box layer vs. full, AES-like vs. somewhat SERPENT-like). It is worth noting however that both ciphers attempt precisely the same goal, namely to offer efficient masked implementations. As a result both reduce nonlinear operations per round to a minimum, while giving more weight to the linear layer; LS-designs achieve this by parallelizing the S-box through bit slicing; Zorro by resorting to a partial S-box layer. In both cases the contribution of the nonlinear layer is very structured with respect to the linear layer; this, together with the minimal key schedule and simple round constants leads to our attacks.

We note that all our attacks can be prevented by a careful choice of round constants. One needs only ensure that no weaker (such as probabilistic or differential) version of the attack is left behind. This is particularly true when claiming related-key security (as in iSCREAM), since in this setting our attacks do not require weak keys, and hence weaker probabilistic versions are all the more relevant.

Going back to the generic algorithm used to find the attacks, an interesting open problem is to specialize it to SPN structures, hoping to achieve better time complexity. In particular, it may be worthwhile to find an algorithm that is able to enumerate all invariant subspaces through a layer of n S-boxes, given n and the S-box. With improvements in time complexity, it may become possible to entirely disprove the existence of invariant subspaces for some SPNs.

Finally, we hope our analysis contributes some insight for the design of future ciphers with minimal key schedules and the choice of round constants in cryptographic permutations.

Chapter 2

Structural Cryptanalysis of ASASA

2.1 Introduction

In the previous chapter, we have seen some interrelated attacks on symmetric primitives. While symmetric and asymmetric primitives are fairly distinct in general, a few primitives attempt to span both worlds. In fact, the idea of creating a public-key cryptosystem by obfuscating a symmetric cipher was proposed by Diffie and Hellman in 1976, in the same seminal paper that introduced the idea of public-key encryption [DH76a]. While the RSA cryptosystem was introduced only a year later, creating a public-key scheme based on symmetric components has remained an open challenge. The interest of this problem is not merely historical: beside increasing the variety of available public-key schemes, one can hope that a solution may help bridge the performance gap between public-key and secret-key cryptosystems, or at least offer new trade-offs in that regard.

Multivariate cryptography is one way to achieve this goal. This area of research dates back to the 1980's [MI88, FD86], and has been particularly active in the late 1990's and early 2000's [Pat95, Pat96, RP97, FJ03, ...]. Many of the proposed public-key cryptosystems build an encryption function from a structured, easily invertible polynomial, which is then scrambled by affine maps (or similarly simple transformations) applied to its input and output to produce the encryption function.

This approach might be aptly described as an ASA structure, which should be read as the composition of an affine map "A", a nonlinear transformation of low algebraic degree "S" (not necessarily made up of smaller S-boxes), and another affine layer "A". The secret key is the full description of the three maps A, S, A', which makes computing both ASA' and $(ASA')^{-1}$ easy. The public key is the function ASA' as a whole, which is described in a generic manner by providing the polynomial expression of each output bit in the input bits (or group of n bits if the scheme operates on $\mathbb{F}_{2^n}$). Thus the owner of the secret key is able to encrypt and decrypt at high speed (provided that S admits an efficient expression). The downside is slow public key operations, and a large key size.

The ASASA Construction.

Historically, most attempts to build public-key encryption schemes based on the above principle have been ill-fated [FJ03, BFP11, DGS07, DFSS07, WBDY98, ...][12]. However new ideas to

[12]The HFEv- variant used in Quartz [PGC01] seems to be an exception in this regard.

build multivariate schemes were recently introduced by Biryukov, Bouillaguet and Khovratovich at Asiacrypt 2014 [BBK14]. The paradigm federating these new ideas is the so-called ASASA structure: that is, combining two quadratic mappings S by interleaving random affine layers A. With quadratic S layers, the overall scheme has degree 4, so the polynomial description provided by the public key remains of reasonable size.

This is very similar to the 2R scheme by Patarin [PG97], which fell victim to several attacks [Bih00, DFKYZD99], including a powerful decomposition attack [DFKYZD99, FP06] (later developed in a general context by Faugère *et al.* [FvzGP10, FP09a, FP09b]). The general course of this attack is to differentiate the encryption function, and observe that the resulting polynomials in the input bits live in a "small" space entirely determined by the first ASA layers. This allows the scheme to be broken down into its two ASA sub-components, which are easily analyzed once isolated. A later attempt to circumvent this and other attacks by truncating the output of the cipher proved insecure against the same technique [FP06] — roughly speaking truncating does not prevent the derivative polynomials from living in too small a space.

In order to thwart attacks including the decomposition technique, the authors of [BBK14] propose to go in the opposite direction: instead of truncating the cipher, a *perturbation* is added, consisting in new random polynomials of degree four added at fixed positions, prior to the last affine layer[13]. The idea is that these new random polynomials will be spread over the whole output of the cipher by the last affine layer. When differentiating, the "noise" introduced by the perturbation polynomials is intended to drown out the information about the first quadratic layer otherwise carried by the derivative polynomials, and foil the decomposition attack.

Based on this idea, two public-key cryptosystems are proposed. One uses random quadratic expanding S-boxes as nonlinear components, while the other relies on the χ function, most famous for its use in the SHA-3 winner Keccak. However the first scheme was broken at Crypto 2015 by a decomposition attack [GPT15]: the number of perturbation polynomials turned out to be too small to prevent this approach. This leaves open the question of the robustness of the other cryptosystem, based on χ (which we shall answer in the negative).

Black-Box ASASA.

Besides public-key cryptosystems, the authors of [BBK14] also propose a secret-key ("black-box") scheme based on the ASASA structure. While the structure is the same, the context is entirely different. This black-box scheme is in fact the exact counterpart of the SASAS structure analyzed by Biryukov and Shamir [BS01]: it is a block cipher operating on 128-bit inputs; each affine layer is a random affine map on $\mathbb{F}_2^{128}$, while the nonlinear layers are composed of 16 random 8-bit S-boxes[14]. The secret key is the description of the three affine layers, together with the tables of all S-boxes.

In some sense, the "public key" is still the encryption function as a whole; however it is only accessible in a black-box way through known or chosen-plaintext or ciphertext attacks, as any standard secret-key scheme. A major difference however is that the encryption function can be easily distinguished from a random permutation because the constituent S-boxes have algebraic degree at most 7, and hence the whole function has degree at most 49; in particular, it sums up to zero over any cube of dimension 50. The security claim is that the secret key cannot be recovered, with a security parameter evaluated at 128 bits.

[13] A similar idea was used in [Din04].

[14] Other choices for the number and size of S-boxes are obviously possible, but for the sake of concreteness we focus on the instance proposed by Biryukov *et al.*

White-Box ASASA.

The structure of the black-box scheme is also used as a basis for several white-box proposals. In that setting, a symmetric (black-box) ASASA cipher with small block (*e.g.* 16 bits) is used as a super S-box in a design with a larger block. A white-box user is given the super S-box as a table. The secret information consists in a much more compact description of the super S-box in terms of alternating linear and nonlinear layers. The security of the ASASA design is then expected to prevent a white-box user from recovering the secret information.

2.1.1 Attacks on ASASA

Structural Attack

Despite the difference in nature between the χ-based public-key scheme, the black-box and the white-box scheme, we present a *structural* attack on ASASA able to break all three constructions. We call the attack structural in the same sense as the structural cryptanalysis of SASAS in [BS01]: that is, the attack recovers internal layers of the ASASA structure from black-box access to the function, even if the layers are uniformly random. The term structural is used by opposition to traditional attacks on symmetric constructions, where all components are known to the adversary save for a small secret key. Here only the *structure* of the components is known (S-boxes, linear layers, etc).

Our attack is underpinned by a degree deficiency that manifests itself when the nonlinear layer is composed of small S-boxes; but the same deficiency is also present in the χ-based multivariate scheme, and can be exploited in the same way. In the case of the multivariate scheme, some tweaks need to be made to account for the presence of the perturbation polynomials. Nevertheless the attack applies regardless of the amount of perturbation. Thus, contrary to the attack of [GPT15], there is no hope of patching the scheme by increasing the number of perturbation polynomials.

While the same attack applies to the black-box and multivariate schemes, bottlenecks for the time complexity come from different stages of the attack. For the χ scheme, the time complexity is dominated by the need to compute the kernel of a binary matrix of dimension 2^{13}, which can be evaluated to 2^{39} basic linear operations[15]. As for the black-box scheme, the time complexity is dominated by the need to encrypt 2^{63} chosen plaintexts, and the data complexity follows.

This attack actually only peels off the last linear layer of the scheme, reducing ASASA to ASAS. In the case of the black-box scheme, the remaining layers can be recovered in negligible time using Biryukov and Shamir's techniques [BS01]. In the case of the χ scheme, removing the remaining layers poses non-trivial algorithmic challenges (such as how to efficiently recover quadratic polynomials $A, B, C \in \mathbb{F}_2[X_1, \ldots, X_n]/\langle X_i^2 - X_i \rangle$, given $A + B \cdot C$). Nevertheless, in the end the remaining layers are peeled off and the secret key is recovered in time complexity negligible relative to the cost of removing the first layer.

We note that our attack does not apply to one of the two multivariate instances of [BBK14]. This is precisely the instance that was already broken by Gilbert, Plût and Treger at Crypto 2015 [GPT15].

In addition to the structural attack, we also present two alternative, dedicated attacks. The first applies to the χ-based multivariate scheme, and the second to the white-box instances of [BBK14]. We emphasize that in both cases, the structural attack still applies, and is in fact

[15] In practice, vector instructions operating on 128-bit inputs would mean that the meaningful size of the matrix is $2^{13-7} = 2^6$, and in this context the number of basic linear operations would be much lower. We also disregard asymptotic improvements such as the Strassen or Coppersmith-Winograd algorithms and their variants. The main point is that the time complexity is quite low — well within practical reach.

more efficient. The alternative attacks merely serve to highlight other exploitable aspects of the ASASA construction.

LPN-Based Attack on the χ Scheme.

In Section 2.5.2, we present an independent attack, dedicated to the χ public-key scheme. This attack exploits the fact that each bit at the output of χ is "almost linear" in the input: indeed the nonlinear component of each bit is a single product, which is equal to zero with probability 3/4 over all inputs. Based on this property, we are able to heuristically reduce the problem of breaking the scheme to an LPN-like instance with easy-to-solve parameters. By LPN-like instance, we mean an instance of a problem very close to the Learning Parity with Noise problem (LPN), on which typical LPN-solving algorithms such as the Blum-Kalai-Wasserman algorithm (BKW) [BKW03] are expected to immediately apply. The time complexity of this approach is higher than the previous one, and can be evaluated at 2^{56} basic operations. However it showcases a different weakness of the χ scheme, providing a different insight into the security of ASASA constructions. In this regard, it is noteworthy that the security of another recent multivariate scheme, presented by Huang *et al.* at PKC 2012 [HLY12], was also reduced to an easy instance of LWE [Reg05], which is an extension of LPN, in [AFF+14][16].

Heuristic Attack on the White-Box Scheme.

In the case of the white-box ASASA instances, as an alternative to the main structural attack, we present another heuristic attack. The attack technique is unrelated to the previous ones, and relies on heuristics rather than a theoretical model. On the other hand it is very effective on the smallest white-box instances of [BBK14] (with a security level of 64 bits), which we break under a minute on a laptop computer.

Regarding the white-box instances, another attack was found independently by Dinur, Dunkelman, Kranz and Leander [DDKL15b]. Their approach focuses on small-block ASASA instances, and is thus only applicable to the white-box scheme of [BBK14]. Section 5 of [DDKL15b] is essentially the same attack as our heuristic attack. On the other hand, the authors of [DDKL15b] present other methods to attack small-block ASASA instances that are less reliant on heuristics for the same performance, showcasing another weakness of small-block ASASA.

2.1.2 Layout of the Chapter

Section 2.3 provides a brief description of the three ASASA schemes under attack. In Section 2.4, we present our main structural attack, as applied to the secret-key ("black-box") scheme. In particular, an overview of the attack is given in Section 2.4.1. The attack is then adapted to the χ public-key scheme in Section 2.5.1, while the LPN-based attack on the same scheme is presented in Section 2.5.2. Finally, attacks on the white-box scheme are presented in Section 2.6.

2.1.3 Implementation

Implementations of our attacks have been made available at:

http://asasa.gforge.inria.fr/

[16] On this topic, the authors of [BBK14] note that "the full application of LWE to multivariate cryptography is still to be explored in the future".

2.2 Notation and Definitions

Binary Vectors.

The set of n-bit vectors is denoted interchangeably by $\{0,1\}^n$ or $\mathbb{F}_2^n$. However the vectors are always regarded as elements of $\mathbb{F}_2^n$ with respect to addition $+$ and dot product $\langle\cdot|\cdot\rangle$. In particular, addition should be understood as bitwise `XOR`. The canonical basis of $\mathbb{F}_2^n$ is denoted by $e_0, \ldots, e_{n-1}$.
For any $v \in \{0,1\}^n$, v_i denotes the i-th coordinate of v. In this context, the index i is always computed modulo n, so $v_0 = v_n$ and so forth. Likewise, if F is a mapping into $\{0,1\}^n$, F_i denotes the i-th bit of the output of F.
For $a \in \{0,1\}^n$, $\langle F|a\rangle$ is a shorthand for the function $x \mapsto \langle F(x)|a\rangle$.
For any $v \in \{0,1\}^n$, $\lfloor v \rfloor_k$ denotes the truncation $(v_0, \ldots, v_{k-1})$ of v to its first k coordinates.
For any bit b, $\bar{b}$ stands for $b+1$.

Derivative of a Binary Function.

For $F : \{0,1\}^m \to \{0,1\}^n$ and $\delta \in \{0,1\}^m$, we define the derivative of F along δ as $\partial F/\partial\delta \triangleq x \mapsto F(x) + F(x+\delta)$. We write $\partial^d F/\partial v_0 \ldots \partial v_{d-1} \triangleq \partial(\ldots(\partial F/\partial v_0)\ldots)/\partial v_{d-1}$ for the order-d derivative along $v_0, \ldots, v_{d-1} \in \{0,1\}^m$. For convenience we may write F' instead of $\partial F/\partial v$ when v is clear from the context; likewise for F''.
The *degree* of F_i is its degree as an element of $\mathbb{F}_2[X_0, \ldots, X_{m-1}]/\langle X_i^2 - X_i\rangle$ in the binary input variables. The degree of F is the maximum of the degrees of the F_i's.

Cube.

A cube of dimension d in $\{0,1\}^n$ is simply an affine subspace of dimension d. The terminology comes from [DS09]. Note that summing a function F over a cube C of dimension d, *i.e.* computing $\sum_{c\in C} F(c)$, amounts to computing the value of an order-d differential of F at a certain point: it is equal to $\partial^d F/\partial v_0 \ldots \partial v_{d-1}(a)$ for a, (v_i) such that $C = a + \mathrm{span}\{v_0, \ldots, v_{d-1}\}$. In particular if F has degree d, then it sums up to zero over any cube of dimension $d+1$.

Bias.

For any probability $p \in [0,1]$, the *bias* of p is $|2p-1|$. Note that the bias is sometimes defined as $|p - 1/2|$ in the literature. Our choice of definition makes the formulation of the Piling-up Lemma more convenient:

Lemma 8 (Piling-up Lemma [Mat94])**.** *For $X_1, \ldots, X_n$ independent random binary variables with respective biases $b_1, \ldots, b_n$, the bias of $X = \sum X_i$ is $b = \prod b_i$.*

Learning Parity with Noise (LPN).

The LPN problem was introduced in [BKW03], and may be stated as follows: given $(A, As+e)$, find s, where:

- $s \in \mathbb{F}_2^n$ is a uniformly random secret vector.
- $A \in \mathbb{F}_2^{N\times n}$ is a uniformly random binary matrix.
- $e \in \mathbb{F}_2^N$ is an *error* vector, whose coordinates are chosen according to a Bernoulli distribution with parameter p.

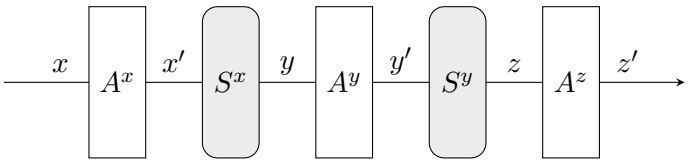

Figure 2.1: The ASASA structure.

2.3 Description of ASASA Schemes

2.3.1 Overview and Notation

ASASA is a general design scheme for public or secret-key ciphers (or cipher components). An ASASA cipher is composed of 5 interleaved layers: the letter A represents an affine layer, and the letter S represents a nonlinear layer (not necessarily made up of smaller S-boxes). Thus the cipher may be pictured as in Fig. 2.1.
We borrow the notation of [GPT15] and write the encryption function F as:

$$F = A^z \circ S^y \circ A^y \circ S^x \circ A^x$$

Moreover, $x = (x_0, \ldots, x_{n-1})$ is used to denote the input of the cipher; x' is the output of the first affine layer A^x; and so on as in Fig. 2.1. The variables x'_i, y_i, etc., will often be viewed as polynomials over the input bits $(x_0, \ldots, x_{n-1})$. Similarly, F denotes the whole encryption function, while $F^y = S^x \circ A^x$ is the partial encryption function that maps the input x to the intermediate state y, and likewise $F^{x'} = A^x$, $F^{y'} = A^y \circ S^x \circ A^x$, etc.

One secret-key ("black-box") and two public-key ASASA ciphers are presented in [BBK14]. The secret-key and public-key variants are quite different in nature, even though our main attack applies to both. We now present, in turn, the black-box construction, and the public-key variant based on χ.

Remark 1. *The name we use for each* ASASA *construction differs slightly from [BBK14]. In our work, we tend to regard the "strong white-box" scheme as a trapdoor permutation. As a consequence we refer to this variant of* ASASA *as* public-key *or* multivariate*, which are more standard terms than strong white-box. Meanwhile the "weak white-box" construction is referred to as simply* white-box *or* small-block ASASA*.*

2.3.2 Description of the Black-Box Scheme

It is worth noting that the following ASASA scheme is the exact counterpart of the SASAS structure analyzed by Biryukov and Shamir [BS01], with the affine layer taking the place of the S-box one and vice versa. Black-box ASASA is a secret-key encryption scheme, parameterized by m, the size of the S-boxes and k, the number of S-boxes. Let $n = km$ be the block size of the scheme (in bits). The overall structure of the cipher follows the ASASA construction, where the layers are as follows:

- A^x, A^y, A^z are a random invertible affine mappings $\mathbb{F}_2^n \to \mathbb{F}_2^n$. Without loss of generality, the mappings can be considered purely linear, because the affine constant can be integrated into the preceding or following S-box layer. In the remainder we assume the mappings to be linear.

- S^x, S^y are S-box layers. Each S-box layer consists in the application of k parallel random invertible m-bit S-boxes.

All linear layers and all S-boxes are chosen uniformly and independently at random among invertible elements.

In the concrete instance of [BBK14], each S-box layer contains $k = 16$ S-boxes over $m = 8$ bits each, so that the scheme operates on blocks of $n = 128$ bits. The secret key consists in three n-bit matrices and $2k$ m-bit S-boxes, so the key size is $3 \cdot n^2 + 2k \cdot m2^m$-bit long. For this instance, this amounts to 14 KB.

It should be pointed out that the scheme is not IND-CPA secure. Indeed, an 8-bit invertible S-box has algebraic degree (at most) 7, so the overall scheme has algebraic degree (at most) 49. Thus, the sum of ciphertexts on entries spanning a cube of dimension 50 is necessarily zero. As a result the security claim in [BBK14] is only that the secret key cannot be recovered, with a security parameter of 128 bits.

2.3.3 Description of the White-Box Scheme

As another application of the symmetric ASASA scheme, Biryukov *et al.* propose its use as a basis for designing white-box block ciphers. In a nutshell, their idea is to use ASASA to create small ciphers of, say, 16-bit blocks and to use them as super S-boxes in *e.g.* a substitution-permutation network (SPN). Users of the cipher in the white-box model are given access to super S-boxes in the form a table, which allows them to encrypt and decrypt at will. Yet if the small ciphers used in building the super S-boxes are secure, one cannot efficiently recover their keys even when given access to their entire codebook, meaning that white-box users cannot extract a more compact description of the super S-boxes from their tables. This achieves *weak white-box security* as defined by Biryukov *et al.* [BBK14]:

Definition 9 (Key equivalence [BBK14])**.** *Let* $\mathrm{E} : \{0,1\}^\kappa \times \{0,1\}^n \to \{0,1\}^n$ *be a (symmetric) block cipher.* $\mathbb{E}(k)$ *is called the* equivalent key set of k *if for any* $k' \in \mathbb{E}(k)$ *one can efficiently compute* E' *such that* $\forall\, p \;\; \mathrm{E}(k,p) = \mathrm{E}'(k',p)$.

Definition 10 (Weak white-box T-security [BBK14])**.** *Let* $\mathrm{E} : \{0,1\}^\kappa \times \{0,1\}^n \to \{0,1\}^n$ *be a (symmetric) block cipher.* $\mathbb{W}(\mathrm{E})(k,\cdot)$ *is said to be a* T-secure weak white-box implementation *of* $\mathrm{E}(k,\cdot)$ *if* $\forall\, p \;\; \mathbb{W}(\mathrm{E})(k,p) = \mathrm{E}(k,p)$ *and if it is computationally expensive to find* $k' \in \mathbb{E}(k)$ *of length less than* T *bits when given full access to* $\mathbb{W}(\mathrm{E})(k,\cdot)$.

Example 1. *If* S_{16} *is a secure cipher with 16-bit blocks, then the full codebook of* $S_{16}(k,\cdot)$ *as a table is a* 2^{20}*-secure weak white-box implementation of* $S_{16}(k,\cdot)$.

For their instantiations, Biryukov *et al.* propose to use several super S-boxes of different sizes, among others:

- A 16-bit ASASA_{16} where the nonlinear permutations S are made of the parallel application of two 8-bit S-boxes, with conjectured security of 64 bits.

- A 24-bit ASASA_{24} where the nonlinear permutations S are made of the parallel application of three 8-bit S-boxes, with conjectured security of 128 bits.

2.3.4 Description of the χ-based Public-Key Scheme

The χ mapping was introduced by Daemen [Dae95] and later used for several cryptographic constructions, including the SHA-3 competition winner KECCAK. The mapping $\chi : \{0,1\}^n \to \{0,1\}^n$ is defined by:

$$\chi_i(a) = a_i + \overline{a_{i+1}} a_{i+2}$$

The χ-based ASASA scheme presented in [BBK14] is a public-key encryption scheme operating on 127-bit inputs, the odd size coming from the fact that χ is only invertible on inputs of odd length. The encryption function may be written as:

$$F = A^z \circ (P + \chi \circ A^y \circ \chi \circ A^x)$$

where:

- A^x, A^y, A^z are random invertible affine mappings $\mathbb{F}_2^{127} \to \mathbb{F}_2^{127}$. In the remainder we will decompose A^x as a linear map L^x followed by the addition of a constant C^x, and likewise for A^y, A^z.
- χ is as above.
- P is the *perturbation*. It is a mapping $\{0,1\}^{127} \to \{0,1\}^{127}$. For 24 output bits at a fixed position, it is equal to a random polynomial of degree 4. On the remaining 103 bits, it is equal to zero.

Since χ has degree only 2, the overall degree of the encryption function is 4. The public key of the scheme is the encryption function itself, given in the form of degree 4 polynomials in the input bits, for each output bit. The private key is the triplet of affine maps (A^x, A^y, A^z).

Due to the perturbation, the scheme is not actually invertible. To circumvent this, some redundancy is required in the plaintext, and the 24 bits of perturbation must be guessed during decryption. The correct guess is determined first by checking whether the resulting plaintext has the required redundancy, and second by recomputing the ciphertext from the tentative plaintext and checking that it matches. This is not relevant to our attack, and we refer the reader to [BBK14] for more information.

2.4 Structural Attack on Black-Box ASASA

Our goal in this section is to recover the secret key of the black-box ASASA scheme, in a chosen-plaintext model. For this purpose, we begin by peeling off the last linear layer, A^z. Once A^z is removed, we obtain an ASAS structure, which can be broken using Biryukov and Shamir's techniques [BS01] in negligible time. Thus the critical step is the first one.

2.4.1 Attack Overview

Before progressing further, it is important to observe that the secret key of the scheme is not uniquely defined. In particular, we are free to compose the input and output of any S-box with a linear mapping of our choosing, and use the result in place of the original S-box, as long as we modify the surrounding linear layers accordingly. Thus, S-boxes are essentially defined up to linear equivalence. When we claim to recover the secret key, this should be understood as recovering an equivalent secret key; that is, any secret key that results in an encryption function identical to the black-box instance under attack.

In particular, in order to remove the last linear layer of the scheme, it is enough to determine, for each S-box, the m-dimensional subspace corresponding to its image through the last linear layer. Indeed, we are free to pick any basis of this m-dimensional subspace, and assert that each element of this basis is equal to one bit at the output of the S-box. This will be correct, up to composing the output of the S-box with some invertible linear mapping, and composing the input of the last linear layer with the inverse mapping; which has no bearing on the encryption output.

Thus, peeling off A^z amounts to finding the image space of each S-box through A^z. For this purpose, we will look for linear masks $a, b \in \{0,1\}^n$ over the output of the cipher, such that the two dot products $\langle F|a\rangle$ and $\langle F|b\rangle$ of the encryption function F along each mask, are each equal to one bit at the output of the *same* S-box in the last nonlinear layer S^y. Let us denote the set of such pairs (a, b) by S (as in "solution").

In order to compute S, the core property at play is that if masks a and b are as required, then the binary product $\langle F|a\rangle\langle F|b\rangle$ has degree only $(m-1)^2$ over the input variables of the cipher, whereas it has degree $2(m-1)^2$ in general. This means that $\langle F|a\rangle\langle F|b\rangle$ sums to zero over any cube of dimension $(m-1)^2+1$.

We now define the two linear masks a and b we are looking for as two vectors of binary unknowns. Then $f(a, b) = \langle F|a\rangle\langle F|b\rangle$ may be expressed as a quadratic polynomial over these unknowns, whose coefficients are $\langle F|e_i\rangle\langle F|e_j\rangle$ for (e_i) the canonical basis of $\mathbb{F}_2^n$. Now, the fact that $f(a, b)$ sums to zero over some cube C gives us a quadratic condition on (a, b), whose coefficients are $\sum_{c\in C}\langle F(c)|e_i\rangle\langle F(c)|e_j\rangle$.

By computing $n(n-1)/2$ cubes of dimension $(m-1)^2+1$, we thus derive $n(n-1)/2$ quadratic conditions on (a, b). The resulting system can then be solved by relinearization. This yields the linear space K spanned by S.

However we want to recover S rather than its linear combinations K. Thus in a second step, we compute S as $S = K \cap P$, where P is essentially the set of elements that stem from a single product of two masks a and b. While P is not a linear space, by guessing a few bits of the masks a, b, we can get many linear constraints on the elements of P satisfying these guesses, and intersect these linear constraints with K.

The first step may be regarded as the core of the attack; and it is also its bottleneck: essentially we need to encrypt plaintexts spanning $n(n-1)/2$ cubes of dimension $(m-1)^2+1$. We recall that in the actual black-box scheme of [BBK14], we have S-boxes over $m = 8$ bits, and the total block size is $n = 128$ bits, covered by $k = 16$ S-boxes, so the complexity is dominated by the computation of the encryption function over 2^{13} cubes of dimension 50, *i.e.* 2^{63} encryptions.

2.4.2 Description of the Attack

We use the notation of Section 2.3.1: let $F = A^z \circ S^y \circ A^y \circ S^x \circ A^x$ denote the encryption function. We are interested in linear masks $a \in \{0,1\}^n$ such that $\langle F|a\rangle$ depends only on the output of one S-box. Since $\langle F|a\rangle = \langle S^y \circ A^y \circ S^x \circ A^x|(A^z)^{\mathrm{T}}a\rangle$, this is equivalent to saying that the active bits of $(A^z)^{\mathrm{T}}a$ span a single S-box.

In fact we are searching for the set S of pairs of masks (a, b) such that $(A^z)^{\mathrm{T}}a$ and $(A^z)^{\mathrm{T}}b$ span the same single S-box. Formally, $O_t = \mathrm{span}\{e_i : mt \le i < m(t+1)\}$ be the span of the output of the t-th S-box, then:

$$S = \{(a, b) \in \{0,1\}^n \times \{0,1\}^n : \exists t, (A^z)^{\mathrm{T}}a \in O_t \text{ and } (A^z)^{\mathrm{T}}b \in O_t\}$$

The core property exploited in the attack is that if (a, b) belongs to S, then $\langle F|a\rangle\langle F|b\rangle$ has degree at most $(m-1)^2$, as shown by Lemma 9 below. On the other hand, if $(a, b) \notin S$, then

$\langle F|a\rangle\langle F|b\rangle$ will look like the product of two independent random polynomials of degree $(m-1)^2$, and reach degree $2(m-1)^2$ with overwhelming probability.

Lemma 9. *Let G be an invertible mapping $\{0,1\}^n \to \{0,1\}^n$ with $n > 2$. For any two n-bit linear masks a and b, $H = \langle G|a\rangle\langle G|b\rangle$ has degree at most $n-1$.*

Proof. It is clear that the degree cannot exceed n, since we depend on only n variables (and we live in $\mathbb{F}_2$). What we show is that it is less than $n-1$, as long as $n > 2$. If $a = 0$ or $b = 0$ or $a = b$, this is clear, so we can assume that a, b are linearly independent. Note that there is only one possible monomial of degree m, and its coefficient is equal to $\sum_{x\in\{0,1\}^n} H(x)$. So all we have to show is that this sum is zero.

Because G is invertible, $G(x)$ spans each value in $\{0,1\}^n$ once as x spans $\{0,1\}^n$. As a consequence, the pair $(\langle G|a\rangle, \langle G|b\rangle)$ takes each of its 4 possible values an equal number of times. In particular, it takes the value $(1,1)$ exactly $1/4$ of the time. Hence $\langle G|a\rangle\langle G|b\rangle$ takes the value 1 exactly 2^{n-2} times, which is even for $n > 2$. Thus $\sum_{x\in\{0,1\}^n} H(x) = 0$ and we are done. □

In the remainder, we regard two masks a and b as two sequences of n binary unknowns $(a_0, \ldots, a_{n-1})$ and $(b_0, \ldots, b_{n-1})$.

Step 1: Kernel Computation.

If a, b are as desired, $\langle F|a\rangle\langle F|b\rangle$ has degree at most $(m-1)^2$. Hence the sum of this product over a cube of dimension $(m-1)^2+1$ is zero, as this amounts to an order-$(m-1)^2+1$ differential of a degree $(m-1)^2$ function. Let then C denote a random cube of dimension $(m-1)^2+1$ – that is, a random affine space of dimension $(m-1)^2+1$, over $\{0,1\}^n$. We have:

$$\begin{aligned}\sum_{c\in C}\langle F(c)|a\rangle\langle F(c)|b\rangle &= \sum_{c\in C}\sum_{i<n} a_i F_i(c) \sum_{j<n} b_j F_j(c)\\ &= \sum_{i,j<n}\Big(\sum_{c\in C} F_i(c)F_j(c)\Big)a_i b_j\\ &= \sum_{i<j<n}\Big(\sum_{c\in C} F_i(c)F_j(c)\Big)(a_i b_j + a_j b_i)\end{aligned}$$

To deduce the last line, notice that $\sum_{c\in C} F_i F_i = 0$ since F has degree less than $\dim C$. Since the equation above really only says something about $a_i b_j + a_j b_i$ rather than $a_i b_j$ (which is unavoidable, since the roles of a and b are symmetric), we define $E = \mathbb{F}_2^{n(n-1)/2}$, see its canonical basis as $e_{i,j}$ for $i < j < n$, and define $\lambda(a,b) \in E$ by: $\lambda(a,b)_{i,j} = a_i b_j + a_j b_i$. By convention we set $\lambda_{j,i} = \lambda_{i,j}$ and $\lambda_{i,i} = 0$. The previous equations tells us that knowing only the $n(n-1)/2$ bits $\sum_{c\in C} F_i(c)F_j(c)$ yields a quadratic condition on (a,b), and more specifically a linear condition on $\lambda(a,b)$. Whence we proceed with Alg. 2.

Let M be a binary matrix of size $(n^2/2) \times (n(n-1)/2)$, whose rows are separate outputs of Alg. 2. Let K be the kernel of this matrix. Then for all $(a,b) \in S$, $\lambda(a,b)$ is necessarily in K. Thus K contains the span of the $\lambda(a,b)$'s for $(a,b) \in S$. Because M contains more than $n(n-1)/2$, with overwhelming probability K contains no other vector[17]. This is confirmed by our experiments.

[17] This point is the only reason we pick $n^2/2$ rows rather than only $n(n-1)/2$; but we may as easily choose $n(n-1)/2$ plus some small constant. In practice it we can just pick $n(n-1)/2$ rows, and add more as required until the kernel has the expected dimension $km(m-1)/2$.

Algorithm 2: GenerateCondition

Input: A random cube C of dimension $(m-1)^2+1$ over $\{0,1\}^n$

Let $sum = (0,\dots,0) \in E$
for $c \in C$ **do**
  $(x_0,\dots,x_{n-1}) \leftarrow F(c)$
  $t \leftarrow (x_i x_j \text{ for } i<j<n) \in E$
  $sum = sum + t$
return sum

Complexity analysis. Overall, the dominant cost is to compute $2^{(m-1)^2+1}$ encryptions per cube, for $n^2/2$ cubes, which amounts to a total of $n^2 2^{(m-1)^2}$ encryptions. With the parameters of [BBK14], this is 2^{63} encryptions. In practice, we could limit ourselves to dimension-$(m-1)^2+1$ subcubes of a single dimension-$(m-1)^2+2$ cube, which would cost only $2^{(m-1)^2+2}$ encryptions. However we would still need to sum (pairwise bit products of) ciphertexts for each subcube, so while this approach would certainly be an improvement in practice, we believe it is cleaner to simply state the complexity as $n^2 2^{(m-1)^2}$ encryption equivalents.

Beside that, we also need to compute the kernel of a matrix of dimension $n(n-1)/2$, which incurs a cost of roughly $n^6/8$ basic linear operations. With the parameters of [BBK14], we need to invert a binary matrix of dimension 2^{13}, costing around 2^{39} (in practice, highly optimized) operations, so this is negligible compared to the required number of encryptions.

Step 2: Mask Extraction.

Let:

$$P = \{\lambda \in E : \exists a, b \in \{0,1\}^n, \lambda = \lambda(a,b)\}$$

Clearly we have $\lambda(S) \subseteq K \cap P$. In fact, we assume $\lambda(S) = K \cap P$, which is confirmed by our experiments. We now want to compute $K \cap P$, but we do not need to enumerate the whole intersection $K \cap P$ directly: for our purpose, it suffices to recover enough elements of $\lambda(S)$ such that the corresponding masks span the output space of all S-boxes. Indeed, recall that our end goal is merely to find the image of all k S-boxes through the last linear layer. Thus, in the remainder, we explain how to find a random element in $K \cap P$. Once we have found km linearly independent masks in this manner, we will be done.

The general idea to find a random element of $K \cap P$ is as follows. We begin by guessing the value of a few pairs (a_i, b_i). This yields linear constraints on the $\lambda_{i,j}$'s. As an example, if $(a_0, b_0) = (0,0)$, then $\forall i, \lambda_{0,i} = 0$. Because the constraints are linear and so is the space K, finding the elements of K satisfying the constraints only involves basic linear algebra. Thus, all we have to do is guess enough constraints to single out an element of S with constant probability, and recover that element as the one-dimensional subspace of K satisfying the constraints.

More precisely, assume we guess $2r$ bits of a, b as:

$$a_0,\dots,a_{r-1} = \alpha_0,\dots,\alpha_{r-1}$$
$$b_0,\dots,b_{r-1} = \beta_0,\dots,\beta_{r-1}$$

We view pairs (α_i, β_i) as elements of $\mathbb{F}_2^2$. Assume there exists some linear dependency between

the (α_i, β_i)'s: that is, for some $(\mu_i) \in \{0,1\}^r$:

$$\sum_{i=0}^{r-1} \mu_i(\alpha_i, \beta_i) = (0,0)$$

Then for all $j < n$, we have:

$$\sum_{i=0}^{r-1} \mu_i \lambda_{i,j} = b_j \sum_{i=0}^{r-1} \mu_i a_i + a_j \sum_{i=0}^{r-1} \mu_i b_i = 0 \tag{2.1}$$

Now, since $\mathbb{F}_2^2$ has dimension only 2, we can be sure that there exist $r-2$ independent linear relations between the (α_i, β_i)'s, from which we deduce as above $(r-2)n$ linear relations on the $\lambda_{i,j}$'s.

Fact 1. *At least $(r-2)(n-r)$ of these relations are linearly independent.*

Proof. The problem may be formalized as follows. We are given $\alpha, \beta \in \mathbb{F}_2^r$. The set of $\mu \in \mathbb{F}_2^r$ such that $\sum \mu_i(\alpha_i, \beta_i) = (0,0)$ is (isomorphic to) the annihilator A of $\mathrm{span}\{\alpha, \beta\}$ in the dual space $(\mathbb{F}_2^r)^*$. Let $M = \mathbb{F}_2^{n \times n}$ denote the space of $n \times n$ binary matrices, with basis $e_{i,j}$. Let $D = \mathrm{span}(\{e^*_{i,i}\} \cup \{e^*_{i,j} + e^*_{j,i}\})$. The space D is isomorphic to the annihilator of $E = \mathbb{F}_2^{n(n-1)/2}$; that is, the space of the $\lambda_{i,j}$'s in Section 2.4.2. Let $B = \mathrm{span}\{\sum \mu_i e^*_{i,j} : \mu \in A, j > r\}$.

The point of the previous definitions is that the linear relations on E obtained from Eq. 2.1 for $j > r$ are exactly the vectors of the space $B \cap D$.

First, we prove that $\dim B \geq (r-2)(n-r)$. Clearly the projection of B on $C_j = \mathrm{span}\{e^*_{i,j} : i < n\}$ (which may be seen as looking at a column in the space M) has dimension $\dim A$ for $j > r$. Since $\sum C_j \subseteq B$ and the C_j's are disjoint, we have $\dim B \geq (n-r)\dim A \geq (r-2)(n-r)$ since A is the annihilator of a space of dimension 2.

Now it remains to show that $\dim(B \cap D) = \dim B$. To see this, it suffices to observe that every functional $e^*_{i,i}$ and $e^*_{i,j} + e^*_{j,i}$ involves a distinct basis element $e^*_{i,j}$ for which $i \leq j$, whereas B is entirely disjoint from the span of those elements. □

Now, the cardinality of S is $k(2^m-1)(2^m-2) \approx k2^{2m}$. Hence if we choose $r = \lfloor \log(|S|)/2 \rfloor \approx m + \frac{1}{2}\log k$, and randomly guess the values of (a_i, b_i) for $i < r$, then we can expect that with constant probability there exists exactly one element in S satisfying our guess. More precisely, each element has a probability (close to) $2^{-2\lfloor |S|/2 \rfloor} \approx 2^{-|S|}$ of fitting our guess of $2r$ bits, so this probability is close to $|S|\left(|S|^{-1}(1-|S|^{-1})^{|S|-1}\right) \approx 1/e$. Thus, if we denote by T the subspace of E of vectors satisfying the linear constraints induced by our guess, with probability roughly $1/3$, $\lambda(S) \cap T$ contains a single element.

On the other hand, K is generated by pairs of masks corresponding to distinct bits for each S-box in S^y. Hence $\dim K = km(m-1)/2 = n(m-1)/2$. As shown earlier, from our $2r$ guesses, we deduce (at least) $(r-2)(n-r)$ linear conditions on the $(\lambda_{i,j})$'s, so $\mathrm{codim}\, T \geq (r-2)(n-r)$. Since we chose $r = m + \frac{1}{2}\log k$, this means:

$$\begin{aligned} \mathrm{codim}\, T &\geq (m - 2 + \frac{1}{2}\log k) \cdot (n - m - \frac{1}{2}\log k) \\ \dim K &= (m-1) \qquad\qquad \cdot (n/2) \end{aligned}$$

Thus, having $\frac{1}{2}\log k \geq 1$, *i.e.* $k \geq 4$, and $m + \frac{1}{2}\log k \geq n/2$, which is easily the case with concrete parameters $m = 8$, $k = 16$, $n = 128$, we have $\mathrm{codim}\, T \geq \dim K$, and so $K \cap T$ is not expected to contain any extra vector beside the span of $\lambda(S) \cap T$. This is confirmed by our experiments.

In summary, if we pick $r = m + \frac{1}{2}\log k$ and randomly guess the first r pairs of bits (a_i, b_i), then with probability close to $1/e$, $K \cap T$ contains only a single vector, which belongs to $\lambda(S) \cap T$ and in particular to $\lambda(S)$. In practice it may be worthwhile to guess a little less than $m + \frac{1}{2}\log k$ pairs to ensure $K \cap T$ is nonzero, then guess more as needed to single out a solution. Once we have a single element in $\lambda(S)$, recovering the two masks (a, b) it stems from simply amounts to inverting λ.

We now show that inverting λ is straightforward. Note that λ is symmetric bilinear and $\forall a, \lambda(a, a) = 0$. As a consequence $\lambda(a, b) = \lambda(a + b, b) = \lambda(a, a + b)$. Moreover $\lambda(a, b) = 0$ if and only if a, b are linearly related; indeed the image of $\lambda(a, b)$ is the set of all determinants of 2×2 submatrices of the $2 \times n$ matrix whose rows are a and b, *i.e.* the set of all 2-minors of the matrix.

As a consequence of the previous properties, by inverting λ, we mean recovering one of $\{a, b\}, \{a + b, b\}, \{a, a + b\}$ given $\lambda(a, b)$, and provided $\lambda(a, b) \neq 0$ (otherwise any linearly dependent a, b is a preimage). For this purpose we use the following algorithm. Consider the $n \times n$ matrix M such that $M_{i,j} = \lambda(a, b)_{i,j}$. Let R^i denote the i-th row of M, *i.e.* $R^i_j = a_i b_j + a_j b_i$. Then observe that $R^i = a_i b + b_i a$. Thus we see that M has rank 2, and in order to invert λ in the previous sense, we need only pick any two linearly independent rows of M.

In the end, we recover two masks (a, b) coming from the same S-box. If we repeat this process $n = km$ times on average, the masks we recover will span the output of each S-box (indeed we recover 2 masks each time, so n tries is more than enough with high probability). Furthermore, checking whether two masks belong to the same S-box is very cheap (for two masks a, b, we only need to check whether $\lambda(a, b)$ is in K), so we recover the output space of each S-box.

Complexity analysis. In order to get a random element in S, each guess of $2r$ bits yields roughly $1/3$ chance of recovering an element by intersecting linear spaces K and T. Since K has dimension $n(m-1)/2$, the complexity is roughly $(n(m-1)/2)^3$ per try, and we need 3 tries on average for one success. Then the process must be repeated n times. Thus the complexity may be evaluated to roughly $\frac{3}{8}n^4(m-1)^3$ basic linear operations. With the parameters of [BBK14], this amounts to 2^{36} linear operations, so this step is negligible compared to Step 1 (and quite practical besides).

Before closing this section, we note that our attack does not really depend on the randomness of the S-boxes or affine layers. All that is required of the S-boxes is that the degree of $z_i z_j$ vary depending on whether i and j belong to the same S-box. This makes the attack quite general, in the same sense as the structural attack of [BS01].

2.5 Attacks on χ-based Multivariate ASASA

In this section, our goal is to recover the private key of the χ-based ASASA scheme, using only the public key. For this purpose, we peel off one layer at a time, starting with the last affine layer A^z. We actually propose two different ways to achieve this. The first attack is our main structural attack from Section 2.4, with some adjustments to account for the specificities of χ and the presence of the perturbation. It is presented in Section 2.5.1. The second attack reduces the problem to an instance of LPN, and is presented in Section 2.5.2. Once the last affine layer has been removed with either attack, we move on to attacking the remaining layers one at a time in Sections 2.5.3 and 2.5.4.

2.5.1 Structural Attack on the χ Scheme

The χ scheme can be attacked in exactly the same manner as the black-box scheme in Section 2.4. Using the notations of Sections 2.3.1 and 2.3.4, we have:

$$\begin{aligned} z_i z_{i+1} &= (y'_i + \overline{y'_{i+1}} y'_{i+2}) \cdot (y'_{i+1} + \overline{y'_{i+2}} y'_{i+3}) \\ &= y'_i y'_{i+1} + y'_i \overline{y'_{i+2}} y'_{i+3} \end{aligned}$$

Here the crucial point is that y'_{i+2} is shared by the only degree-4 term of both sides. Thus the degree of $z_i z_{i+1}$ is bounded by 6. Likewise, the degree of $z_{i+1}(z_i + z_{i+2}) = z_i z_{i+1} + z_{i+1} z_{i+2}$ is also bounded by 6, as the sum of two products of the previous form. On the other hand, any product of linear combinations $(\sum \alpha_i z_i)(\sum \beta_i z_i)$ not of the previous two forms does not share common y'_i's in its higher-degree terms, so no simplification occurs, and the product reaches degree 8 with overwhelming probability.

As a result, we can proceed as in Section 2.4. Let $n = 127$ be the size of the scheme, $p = 24$ the number of perturbation polynomials. The positions of the p perturbation polynomials are not defined in the original paper; in the sequel we assume that they are next to each other. Other choices of positions increase the tedium of the attack rather than its difficulty.Due to the rotational symmetry of χ, the positions of the perturbed bits is only defined modulo rotational symmetry; for convenience, we assume that perturbed bits are at positions z_{n-p} to z_{n-1}.

The full attack presented below has been verified experimentally for small values of n.

Step 1: Kernel Computation.

We fill the rows of an $n(n-1)/2 \times n(n-1)/2$ matrix with separate outputs of Algorithm 2, with the difference that the dimension of cubes in the algorithm is only 7 (instead of $(m-1)^2+1 = 50$ in the black-box case). Then we compute the kernel K of this matrix. Since $n(n-1)/2 \approx 2^{13}$ the complexity of this step is roughly 2^{39} basic linear operations.

Step 2: Mask Extraction.

The second step is to intersect K with the set P of elements of the form $\lambda(a, b)$ to recover actual solutions (see Section 2.4, step 2). In Section 2.4 we were content with finding random elements of $K \cap P$. Now we want to find all of them. To do so, instead of guessing a few pairs (a_i, b_i) as earlier, we exhaust all possibilities for (a_0, b_0) then (a_1, b_1) and so forth along a tree-based search. For each branch, we stop when the dimension of K intersected with the linear constraints stemming from our guesses of (a_i, b_i)'s is reduced to 1. Each branch yields a solution $\lambda(a, b)$, from which the two masks a and b can be easily recovered.

Step 3: Mask Sorting.

Let $a_i = ((L^z)^{\mathrm{T}})^{-1} e_i$ be the linear mask such that $z_i = \langle F | a_i \rangle$ (for the sake of clarity we first assume $C^z = 0$; this has no impact on the attack until step 4 in Section 2.5.3 where we will recover C^z). At this point we have recovered the set S of all (unordered) pairs of masks $\{a_i, a_{i+1}\}$ and $\{a_i, a_{i-1} + a_{i+1}\}$ for $i < n - p$, *i.e.* such that the corresponding z_i's are not perturbed. Now we want to distinguish masks $a_{i-1} + a_{i+1}$ from masks a_i. For each i such that z_{i-1}, z_i, z_{i+1} are not perturbed, this is easy enough, as a_i appears exactly three times among unordered pairs in S: namely in the pairs $\{a_i, a_{i-1}\}$, $\{a_i, a_{i+2}\}$ and $\{a_i, a_{i-1} + a_{i+1}\}$; whereas masks of the form $a_{i-1} + a_{i+1}$ appear only once, in $\{a_{i-1} + a_{i+1}, a_i\}$.

Thus we have recovered every a_i for which z_{i-1}, z_i, z_{i+1} are not perturbed. Since perturbed bits are next to each other, we have recovered all unperturbed a_i's save the two a_i's on the outer edge of the perturbation, *i.e.* a_0 and a_{n-p-1}. We can also order all recovered a_i's simply by checking whether $\{a_i, a_{i+1}\}$ is in S. In other words, we look at S as the set of edges of a graph whose vertices are the elements of pairs in S; then the chain $(a_1, \ldots, a_{n-p-2})$ is simply the longest path in this graph. In fact we recover $(a_1, \ldots, a_{n-p-2})$, minus its direction: that is, so far, we cannot distinguish it from $(a_{n-p-2}, \ldots, a_1)$. If we look at the neighbors of the end points of the path, we also recover $\{a_0, a_0 + a_2\}$ and $\{a_{n-p-1}, a_{n-p-3} + a_{n-p-1}\}$. However we are not equipped to tell apart the members of each pair with only S at our disposal.

To find a_0 in $\{a_0, a_0+a_2\}$ (and likewise a_{n-p-2} in $\{a_{n-p-1}, a_{n-p-3}+a_{n-p-1}\}$), a very efficient technique is to anticipate a little and use the distinguisher from Section 2.5.2. Namely, in short, we differentiate the encryption function F twice using two fixed random input differences $\delta_1 \neq \delta_2$, and check whether for a fraction $1/4$ of possible choices of (δ_1, δ_2), $\langle \partial^2 F / \partial\delta_1 \partial\delta_2 | x \rangle$ is equal to a constant with bias 2^{-4}: this property holds if and only if x is one of the a_i's. This only requires around 2^{16} encryptions for each choice of (δ_1, δ_2), and thus completes in negligible time. Another more self-contained approach is to move on to the next step (in Section 2.5.3), where the algorithm we use is executed separately on each recovered mask a_i, and fails for $a_0 + a_2$ but not a_1. However this would be slower in practice.

Regardless of which solution was chosen, we now assume that we know the whole ordered chain $(a_0, \ldots, a_{n-p-1})$ of masks corresponding to unperturbed bits. At this stage we are only missing the direction of the chain, *i.e.* we cannot distinguish $(a_0, \ldots, a_{n-p-1})$ from $(a_{n-p-1}, \ldots, a_0)$. This will be corrected at the next step.

As mentioned earlier, we propose two different techniques to recover the first linear layer of the χ scheme: one is our main technique, and another based on LPN. We have now just completed the algebraic technique. In the next section we present the LPN-based technique. Afterwards we will move on to the remaining steps, which are common to both techniques, and fully break the cipher with the knowledge of $(a_0, \ldots, a_{n-p-1})$, in Sections 2.5.3 and 2.5.4.

2.5.2 LPN-based Attack on the χ Scheme

We now present a different approach to remove the last linear layer of the χ scheme. This approach relies on the fact that each output bit of χ is almost linear, in the sense that the only nonlinear component is the product of two input bits. In particular this nonlinear component is zero with probability $3/4$. The idea is then to treat this nonlinear component as random noise. To achieve this we differentiate the encryption function F twice. So the first ASA layers of F'' yield a constant; then ASAS is a noisy constant due to the weak nonlinearity; and ASASA is a noisy constant accessed through A^z. This allows us to reduce the problem of recovering A^z to (a close variant of) an LPN instance with tractable parameters.

We now describe the attack in detail. First, pick two distinct random differences $\delta_1, \delta_2 \in \{0,1\}^n$. Then compute the order 2 differential of the encryption function along these two differences. That is, let $F'' = \partial F / \partial\delta_1 \partial\delta_2$. This second-order differential is constant at the output of $F^{y'} = A^y \circ \chi \circ A^x$, since χ has degree only two:

$$(F^{y'})''(x) \triangleq \partial F^{y'} / \partial\delta_1 \partial\delta_2 = C(\delta_1, \delta_2)$$

Now if we look at a single bit at the output of $F^z = \chi \circ F^{y'}$, we have:

$$\begin{aligned}(F^z)''_i(x) = (F^{y'})''_i(x) &+ \overline{F^{y'}_{i+1}} F^{y'}_{i+2}(x) + \overline{F^{y'}_{i+1}} F^{y'}_{i+2}(x + \delta_1) \\ &+ \overline{F^{y'}_{i+1}} F^{y'}_{i+2}(x + \delta_2) + \overline{F^{y'}_{i+1}} F^{y'}_{i+2}(x + \delta_1 + \delta_2) \end{aligned} \tag{2.2}$$

That is, a bit at the output of $(F^z)''$ still sums up to a constant, plus the sum of four bit products. If we look at each product as an independent random binary variable that is zero with probability 3/4, *i.e.* bias 2^{-1}, then by the Piling-up Lemma (8) the sum is equal to zero with bias 2^{-4}.

Experiments show that modeling the four products as independent is not quite accurate: a significant discrepancy is introduced by the fact that the four inputs of the products sum up to a constant. For the sake of clarity, we will disregard this for now and pretend that the four products are independent. We will come back to this issue later on.

Now a single linear layer remains between $(F^z)''$ and F''. Let $s_i \in \{0,1\}^n$ be the linear mask such that $\langle F|s_i\rangle = F_i^z$ (once again we assume $C^z = 0$, and postpone taking C^z into account until step 4 of the attack). Then $\langle F''|s_i\rangle$ is equal to a constant with bias 2^{-4}. Now let us compute N different outputs of F'' for some N to be determined later, which costs $4N$ calls to the encryption function F. Let us stack these N outputs in an $N \times n$ matrix A.

Then we know that $A \cdot s_i$ is either the all-zero or the all-one vector (depending on $(F^{y'})''_i$) plus a noise of bias 2^{-4}. Thus finding s_i is essentially an LPN problem with dimension $n = 127$ and bias 2^{-4} (*i.e.* noise $1/2 + 2^{-5}$). Of course this is not *quite* an LPN instance: A is not uniform, there are n solutions instead of one, and there is no output vector b (although we could isolate the last column of A and define it as the output vector). However in practice none of this should hinder the performance of a BKW algorithm [BKW03]. Thus we make the heuristic assumption that BKW performs here as it would on a standard LPN instance[18].

In the end, we recover the masks s_i such that $z_i = \langle F|s_i\rangle$. Before moving on to the next stage of the attack, we go back to the earlier independence assumption.

Dependency Between the Four Products.

In the reasoning above, we have modeled the four bit products in Equation 2.2 as independent binary random variables with bias 2^{-1}. That is, we assumed the four products would behave as:

$$\Pi = W_1W_2 + X_1X_2 + Y_1Y_2 + Z_1Z_2$$

where W_i, X_i, Y_i, Z_i are uniformly random *independent* binary variables. This yields an expectancy $\mathbb{E}[\Pi]$ with bias 2^{-4}. As noted above, this is not quite accurate, and we now provide a more precise model that matches with our experiments.

Since $F^{y'}$ has degree two, $(F^{y'})''$ is a constant, dependent only on δ_1 and δ_2. This implies that in the previous formula, we have $W_1+X_1+Y_1+Z_1 = (F^{y'})''_{i+1}$ and $W_2+X_2+Y_2+Z_2 = (F^{y'})''_{i+2}$. To capture this, we look at:

$$E(c_1, c_2) = \mathbb{E}[\Pi \,|\, W_1 + X_1 + Y_1 + Z_1 = c_1, W_2 + X_2 + Y_2 + Z_2 = c_2]$$

It turns out that $E(0,0)$ has a stronger bias, close to 2^{-3}; while perhaps surprisingly, $E(a,b)$ for $(a,b) \neq (0,0)$ has bias zero, and is thus not suitable for our attack. Since G'' is essentially random, this means that our technique will work for only a fraction 1/4 of output bits. However, once we have recovered these output bits, we can easily change δ_1, δ_2 to obtain a new value of G'' and start over to find new output bits.

[18] To the best of our knowledge, we have yet to see an LPN-like problem with a matrix A on which BKW underperforms significantly compared to the uniform case, unless the problem was specifically crafted for this purpose. The existence of multiple solutions is also a notable difference in our case. However in a classic application of BKW with a fast Fourier transform at the end, this only means that the Fourier transform will output several solutions. Note that the dimension of the Fourier transform will be close to $127/3 \approx 42$ [LF06], and we have only $\approx 2^{14}$ solutions, so they are distinct on their last 42 bits with very high probability.

After k iterations of the above process, a given bit at position $i \leq 127$ will have probability $(3/4)^k$ of remaining undiscovered. In order for all 103 unperturbed bits to be discovered with good probability, it is thus enough to perform $k = -\log(103)/\log(3/4) \approx 16$ iterations.

In the end we recover all linear masks a_i corresponding to unperturbed bits at the output of the second χ layer; *i.e.* $a_i = ((A^z)^{\mathrm{T}})^{-1}e_i$ for $0 \leq i < n-p$. The a_i's can then be ordered into a chain $(a_0, \ldots, a_{n-p-1})$ like in Section 2.5.1: neighbouring a_i's are characterized by the fact that $\langle F|a_i\rangle\langle F|a_{i+1}\rangle$ has degree 6. We postpone distinguishing between $(a_0, \ldots, a_{n-p-1})$ and $(a_{n-p-1}, \ldots, a_0)$ until Section 2.5.3.

Complexity analysis. According to Theorem 2 in [LF06], the number of samples needed to solve an LPN instance of dimension 127 and bias 2^{-4} is $N = 2^{44}$ (attained by setting $a = 3$ and $b = 43$). This requires $4N = 2^{46}$ encryptions. Moreover the dominant cost in the time complexity is to sort the 2^{44} samples a times, which requires roughly $3 \cdot 44 \cdot 2^{44} < 2^{52}$ basic operations. Finally, as noted above, we need to iterate the process 16 times to recover all unperturbed output bits with good probability, so our overall time complexity is increased to 2^{56} for BKW, and 2^{50} encryptions to gather samples (slightly less with a structure sharing some plaintexts between the 16 iterations).

2.5.3 From ASAS to ASA

The next layer we wish to peel off is a χ layer, which is entirely public. It may seem that applying χ^{-1} should be enough. The difficulty arises from the fact that we do not know the full output of χ, but only $n-p$ bits. Furthermore, if our goal was merely to decrypt some specific ciphertext, we could use other techniques, *e.g.* the fact that guessing one bit at the input of χ produces a cascade effect that allows recovery of all other input bits from output bits, regardless of the fact that the function has been truncated [Dae95]. However our goal is different: we want to recover the secret key, not just be able to decrypt messages. For this purpose we want to cleanly recover the input of χ in the form of degree 2 polynomials, for every unperturbed bit. We propose a technique to achieve this below.

From the previous step, we are in possession of $(a_0, \ldots, a_{n-p-1})$ as defined above. Since by definition $z_i = \langle F|a_i\rangle$, this means we know z_i for $0 \leq i < n-p$. Note that y'_i has degree only 2, and we know that $z_i = y'_i + \overline{y'_{i+1}}y'_{i+2}$. In order to reverse the χ layer, we set out to recover y'_i, y'_{i+1}, y'_{i+2} from the knowledge of only z_i, by using the fact that y'_i, y'_{i+1}, y'_{i+2} are quadratic.

This reduces to the following problem: given $P = A + B \cdot C$, where A, B, C are degree-2 polynomials, recover A, B, C. A closer look reveals that this problem is not possible exactly as stated, because P can be equivalently written in several different ways, such as: $A + B \cdot C$, $A + B + B \cdot \overline{C}$, or $A + C + (B + C) \cdot C$. On the other hand, we assume that for uniformly random A, B, C, the probability that P may be written in some unrelated way, *i.e.* $P = C + D \cdot E$ for C, D, E not in the linear span of $A, B, C, 1$, is overwhelmingly low. This situation has never occurred in our experiments. Thus our problem reduces to:

Problem 1. *Let A, B, C be quadratic polynomials in $\mathcal{Q} = \mathbb{F}_2[X_0, \ldots, X_{n-1}]/\langle X_i^2 - X_i\rangle$. Let $P = A + B \cdot C$. The problem is to recover quadratic A', B', C' such that $P = A' + B' \cdot C'$, given only P.*

Remark 2. *Problem 1 is part of a general family of polynomial decomposition problems which have very recently been shown to be solvable in polynomial time [BHT15]. However our particular instance is much easier than the general case considered in [Bha14, BHT15]. This allows us to*

propose a much simpler and more efficient dedicated algorithm. Our algorithm is unrelated to those used in the general case, which rely on higher-order Fourier analysis.

Our previous assumption says $A' \in \text{span}\{A, B, C, 1\}$; $B', C' \in \text{span}\{B, C, 1\}$. A straightforward approach to tackle the problem is to write B formally as a generic degree-2 polynomial with unknown coefficients This gives us $k = 1 + n + n(n+1)/2 \approx n^2/2$ binary unknowns. Then we observe that $B \cdot P$ has degree only 4 (since $B^2 = B$). Each term of degree 5 in $B \cdot P$ must have a zero coefficient, and thus each term gives us a linear constraint on the unknown coefficients of B. Collecting the constraints takes up negligible time, at which point we have a $k \times k$ matrix whose kernel is $\text{span}\{B, C, 1\}$. This gives us a few possibilities for B', C', which we can filter by checking that $A' = P - B' \cdot C'$ has degree 2. The complexity of this approach boils down to inverting a k-dimensional binary matrix, which costs essentially 2^{3k} basic linear operations. In our case this amounts to 2^{39} basic linear operations. While this is a straightforward approach, and its complexity is reasonable, a much more efficient algorithm is given below.

An Efficient Algorithm for Problem 1.

As previously mentioned, $A' = A, B' = B, C' = C$ cannot be the only solution; for instance $A' = A + C, B' = B + C, C' = C$ is also possible. Conceptually, our algorithm will attempt to recover B and C; but in effect it recovers any two linearly independent elements of $\text{span}\{B, C, 1\}$, which are indistinguishable from (B, C) with knowledge of only P.

In fact our algorithm only attempts to recover the homogeneous degree-2 components of B, C. The linear components can then be defined as $2n$ unknowns and recovered using simple linear algebra from the degree-3 monomials of P. This only involves inverting a matrix in dimension $2n = 254$, which has negligible cost. Moreover $A = P - B \cdot C$. Thus we focus on recovering the degree-2 monomials of B, C. In the remainder we will slightly abuse notation and write B, C to mean the homogeneous degree-2 components of B, C, *i.e.* we disregard the linear and constant components.

In an effort to reduce notational clutter, we always assume knowledge of P, and do not pass it as parameter to every algorithm. Let $n = 127$ and $[n] = \{0, \ldots, n-1\}$. For $D \in \mathcal{Q}$, we write $D_{i,j}$ for the coefficient of $X_i X_j$ in D (we identify elements of $\mathcal{Q}$ with their square-free representation in $\mathbb{F}_2[X_0, \ldots, X_{n-1}]$). By convention $D_{i,i} = 0$. Likewise we define $P_{i,j,k,l}$ as the coefficient of $X_i X_j X_k X_l$ in P. Finally, for $D \in \mathcal{Q}$, $D_{i,*}$ is the vector $(D_{i,0}, \ldots, D_{i,n-1}) \in \mathbb{F}_2^n$.

Our algorithm makes use of two simple "zero oracles" Z (Alg. 4) and Z' (Alg. 3). The oracle $Z(i, j)$ returns True if and only if $B_{i,j} = C_{i,j} = 0$. It makes use of the oracle Z', which returns True if and only if:

$$B_{i,j} = B_{j,k} = B_{i,k} = C_{i,j} = C_{j,k} = C_{i,k} = 0 \tag{2.3}$$

Both oracles attempt to find information on $B_{i,j}$ and $C_{i,j}$, and in their description above, we describe their output as depending on B and C. But the oracle answers are actually computed without access to either, as we shall see. In both cases there is a small chance of the oracle answer being wrong. However this happens with low probability, and our algorithm is made resilient to such errors at a later point.

Note that we have:

$$P_{i,j,k,l} = B_{i,j}C_{k,l} + B_{i,k}C_{j,l} + B_{i,l}C_{j,k} + B_{j,k}C_{i,l} + B_{j,l}C_{i,k} + B_{k,l}C_{i,j} \tag{2.4}$$

Algorithm 3: ZeroTripletOracle Z'

Input: distinct $i, j, k \in [n]$
Output: True if Eq. 2.3 holds, False otherwise

```
for l ≠ i, j, k ∈ [n] do
    if P_{i,j,k,l} = 1 then
        return False
return True
```

So Eq. 2.3 implies $\forall l, P_{i,j,k,l} = 0$. Conversely if Eq. 2.3 does not hold, then $P_{i,j,k,l} = 0$ holds for all $l \neq i, j, k$ with probability close to $2^{-(n-3)} = 2^{-124}$. As a result, Z' is correct except with negligible probability.

Now we use Z' to build Z, which returns True if and only if $B_{i,j} = C_{i,j} = 0$. As with Z',

Algorithm 4: ZeroPairOracle Z

Input: distinct $i, j \in [n]$
Output: True if $B_{i,j} = C_{i,j} = 0$, False otherwise

```
for k ≠ i, j ∈ [n] do
    if Z(i, j, k) then
        return True
return False
```

there is a low probability of incorrect answer for Z', but our algorithm will be made resilient to these errors later on.

Now we build a function FindGood(i) (Alg. 5), whose purpose will become clear shortly. FindGood picks j, k randomly until $Z(i, j)$ and $Z(i, k)$ hold, but not $Z(j, k)$. This is the case

Algorithm 5: FindGood

Input: $i \in [n]$
Output: $j, k \in [n]$ such that $B_{i,j} = B_{i,k} = C_{i,j} = C_{i,k} = 0$, but $(B_{j,k}, C_{j,k}) \neq (0, 0)$

```
while True do
    j ←$ [n] − {i}
    k ←$ [n] − {i, j}
    if Z(i, j) and Z(i, k) and not Z(j, k) then
        return (j, k)
```

with probability roughly 2^{-6}, and there are $n(n-1)/2$ choices for j, k so the probability of failure is negligible. Now we explain the point of FindGood.

Let $(\lambda, \mu) = (B_{j,k}, C_{j,k})$. The point of having the conditions at the output of FindGood is that due to Eq.2.4, they imply:

$$\forall l, P_{i,j,k,l} = \lambda B_{i,l} + \mu C_{i,l}$$

so we recover $(\lambda B + \mu C)_{i,l}$ for all l simply by looking at $P_{i,j,k,l}$. For simplicity we assume $(\lambda, \mu) = (1, 0)$, and so we are recovering $B_{i,l}$ (other cases correspond to the other two nonzero

elements of $\text{span}\{B, C\}$, which as pointed earlier cannot be distinguished from B). If we view B as an $n \times n$ symmetric binary matrix with entries $B_{i,j}$, this means we recover a row of B, namely $B_{i,*}$. Now we can naturally define GetSpace(i) (Alg. 6), which recovers $\text{span}\{B_{i,*}, C_{i,*}\}$:

Algorithm 6: GetSpace

Input: $i \in [n]$
Output: $\text{span}\{B_{i,*}, C_{i,*}\}$

```
Let v ∈ F_2^n
Let E = {0} ⊆ F_2^n
while dim E < 2 do
    (j, k) ← FindGood(i)
    for l ∈ [n] do
        v_l ← P_{i,j,k,l}
    E ← E + span{v}
return E
```

For all i we now know $\text{span}\{B_{i,*}, C_{i,*}\}$. All that remains to do in order to build B (or C, or $B + C$) is to choose a nonzero element of GetSpace(0) as the first row; then an element of GetSpace(1) as the second row; and so forth. At each step i, we make sure that our choice of elements is coherent up to this point by checking that the submatrix of rows 0 to i and columns 0 to i is symmetric. If not, we change our choice of element, backtracking if necessary. This is described in Alg. 7.

Algorithm 7: Solve

Input: $G, H \in \mathbb{F}_2^{n \times n}$, step $\in [n]$
Output: $\text{span}\{B, C\}$ or FAIL

```
if step = n then
    return span{G, H}
for try ∈ {1, 2} do
    for each choice of (x, y) linearly independent in GetSpace(step) do
        G_{step,*} ← x
        H_{step,*} ← y
        if ∀i < step, G_{i,step} = G_{step,i} and H_{i,step} = H_{step,i} then
            S ← Solve(G, H, step + 1)
            if S ≠ FAIL then
                return S
return FAIL
```

In the end, $\text{span}\{B, C\}$ is recovered as Solve$(0, 0, 0)$ (where the first two parameters are the zero matrix of $\mathbb{F}_2^{n \times n}$). Notice that every recursive call to Solve repeats its inner loop twice in case of failure. This is to account for the very rare case where the output of GetSpace might be wrong. Our implementation never returns FAIL and completes within a second for $n = 127$, which is the actual n value for the χ-based ASASA scheme (see Section 2.1.3 for a link to our implementation).

Application to ASAS.

Note that we only need to go through the previous algorithm for the first unperturbed bit in the chain $(z_0, \ldots, z_{n-p-1})$, namely z_0. Indeed, we then recover y'_0, y'_1, y'_2, and for the next bit we have $z_1 = y'_1 + \overline{y'_2} y'_3$, so only y'_3 remains to be determined. This can be performed in negligible time, as the system of equations stemming from this equality on the coefficients of y'_3 is very sparse[19]. By induction we can propagate this process to all other unperturbed bits.

However in the course of this process we also have to deal with the fact that even from the start, we do not recover y'_0, y'_1, y'_2 exactly, but $\mathrm{span}\{y'_0, y'_1, y'_2, 1\}$ and $\mathrm{span}\{y'_1, y'_2, 1\}$. Thus we need to guess y'_0, y'_1, y'_2 from the elements of these two vector spaces, then start the process of rebuilding the rest of the chain $(y'_0, \ldots, y'_{n-p-1})$ as in the previous paragraph. In our experiments, it turns out that as long as $p \geq 2$, there are always exactly 8 solutions for the chain of degree-2 polynomials $(y'_0, \ldots, y'_{n-p-1})$.

To understand why, we need to look at the last unperturbed bit z_{n-p-1}. For this bit, we recover $\mathrm{span}\{y'_{n-p-1}, y'_{n-p}, y'_{n-p+1}, 1\}$ and $\mathrm{span}\{y'_{n-p}, y'_{n-p+1}, 1\}$. We can recognize y'_{n-p-1} in the first space (or rather $\mathrm{span}\{y'_{n-p-1}, 1\}$) because it is also one of the factors in the expression of z_{n-p-2}. We can also identify $\mathrm{span}\{y'_{n-p}, 1\}$ for the same reason. However there is fundamentally no way to tell y'_{n-p-1} apart from $\overline{y'_{n-p-1}}$, and likewise for y'_{n-p}, $\overline{y'_{n-p}}$, because the necessary information is erased from the public key by the perturbation. For y'_{n-p} for instance, we could flip the $(n-p)$-th bit in the constant C_1 of A^y and also flip the perturbed bit z_{n-p} and this would flip y'_{n-p} without changing $(z_0, \ldots, z_{n-p-1})$. Thus all 8 solutions for $(y'_0, \ldots, y'_{n-p-1})$ are valid in the sense that they correspond to equivalent keys, and we are free to choose one of them arbitrarily.

Finally, up to this stage of the attack, we have pretended that $C^z = 0$. This actually has no impact on any algorithm so far, except the one just above. With nonzero C^z, we have $\langle F|a_i\rangle = z_i + c_i$ for $c = (A^z)^{-1}C^z$. This merely adds another degree of freedom in the construction of the previous chain: we guess c_0 and attempt to go through the process of building the chain. If our guess was incorrect the algorithm fails after two iterations. Once it goes through for two iterations we guess c_1 and attempt one more iteration, and so forth. Since the chain-building step has negligible complexity, this takes up negligible time.

Overall our algorithm is able to solve Problem 1 for the full $n = 127$ within a second on a laptop computer. Thus the time complexity of this step is negligible.

2.5.4 Peeling off the Remaining ASA Layers

From ASA to SA.

At the end of the previous step we have recovered the chain $(y'_0, \ldots, y'_{n-p-1})$ of polynomials at the output of A^y. Now we are left with the task of recovering A^x, $\lfloor A^y \rfloor_{n-p}$ from $H = \lfloor A^y \circ \chi \circ A^x \rfloor_{n-p}$. Up to now we have always taken advantage of the very simple action of χ when considering only a single output bit. However because A^y is truncated, we cannot expect that there exist linear masks on the truncated output of A^y that give us access to a single bit at the output of χ. For this reason we switch gear and set out to remove the first layer A^x instead.

First, we want to compute the linear component L^x of A^x. Let $\Delta = \{(L^x)^{-1}e_i : i < n\}$ denote the set of differences δ that activate only a single bit at the input of χ. Observe that a single bit difference at the input of χ only affects 3 output bits. As a result we have an oracle

[19] For instance each degree-4 term in z_1 may be written in only $\binom{4}{2} = 6$ ways as a product of two quadratic terms, and so the corresponding equation on the coefficients of y'_3 involves only 3 terms on average, and many such equations have only one term, yielding a direct equality.

$\mathcal{O}$ that recognizes elements of Δ: namely for $\delta \in \Delta$, the output of $H' = \partial H/\partial\delta$ has dimension only 3 as x spans $\{0,1\}^n$.

Furthermore, a closer look reveals that if we remove the constant component in the output of H', then the output of H' has dimension only 2. The reason for this is that, while each bit at the input of χ affects 3 bits at the output, only 2 bits are affected in a nonlinear manner; and since we are differentiating H, the linear component of χ only affects the constant component in the output of H'.

Let us define an input difference δ as a vector of n binary unknowns. Then we can formally compute the function H'. Assume $\delta \in \Delta$. Per the previous observation, we know that the linear component of $H'(x)$ has dimension only two as x spans $\{0,1\}^n$. That is, for any pairwise distinct $k_0, k_1, k_2 < n - p$, there exists a nonzero vector $(\lambda_0, \lambda_1, \lambda_2)$ such that $\sum \lambda_i H'_{k_i} = C(\delta)$ is constant[20].

Now observe that H' has degree only one in its input variables x_i, and the coefficient of each x_i is a linear combination of δ_i's, hence the above equality gives us n linear conditions on δ. Since δ lives in a space of dimension n, we can hope that this is enough to recover δ, at the cost of guessing the λ_i's (only 8 possibilities).

In short the algorithm so far sums up to:

1. Pick pairwise distinct $k_0, k_1, k_2 < n - p$ arbitrarily.
2. Guess $\lambda = (\lambda_0, \lambda_1, \lambda_2)$.
3. Write the polynomial equality $\sum \lambda_i H'_{k_i} = 0$. By looking at the coefficient of each x_i in this equality, we have n linear conditions on δ.

Let $K(k, \lambda)$ denote the linear subspace of vectors satisfying these conditions. Then we know that for every $\delta \in \Delta$, and every choice of k, there exists λ such that $\delta \in K(k, \lambda)$.

Note that the cardinality of Δ is $128 = 2^7$, while λ only contains 3 bits of information. Hence, in order to single out each element of Δ, we repeat the previous algorithm 3 times. This gives us 3 sets of 8 spaces $K(k, \lambda)$. For every choice of K in each set, we compute the intersection of the spaces ($8^3 = 2^9$ possibilities). This yields 2^9 intersections. By construction we know that each element of Δ is in one of the intersections. So we recover Δ by testing every element in every intersection against the oracle $\mathcal{O}$.

There are 2^9 intersections, so the only remaining question is whether some of the intersections have dimension greater than 0 or 1 (which may considerably slow down the algorithm). Our experiments show that this is in fact the case, but the resulting spaces still have very low dimension. This is due to "false positives" caused by differences δ that activate 2 or 3 differences at the input of δ; but these are quickly weeded out by testing against the oracles $\mathcal{O}$.

We have now recovered the linear component of A^x. Thus we have access to $\lfloor A^y \circ \chi \circ (\oplus C^x) \rfloor_{n-p}$, where $\oplus C^x$ denotes the addition of the constant C^x. In order to recover C^x, we can use the fact that $\chi(v) + \chi(v + e_i) = e_i$ (where e_i is the canonical basis of $\mathbb{F}_2^n$) if and only if $v_{i-1} = v_{i+1} = 0$. So for each i, we can flip the bits at position $i - 1$ and $i + 1$ at the input of $\lfloor A^y \circ \chi \circ (\oplus C^x) \rfloor_{n-p}$ until the previous equality holds. This allows us to recover C^x very quickly.

Overall the complexity of this step can be approximated by 2^9 intersections of 3 spaces of dimension 128, which costs around $2^9 \cdot 2 \cdot (2^7)^3 = 2^{31}$, so this step is negligible compared to step 1. In fact we have implemented this step on the full version of the scheme, and it takes only about a minute to complete on a laptop computer.

[20] We always mean "constant" and "non-constant" with respect to the *input* x of H', and not the difference δ.

From SA to A.

We know $\lfloor A^y \circ \chi \rfloor_{n-p}$, and we want to recover $\lfloor A^y \rfloor_{n-p}$. Observe that $\chi(0) = 0$, so $\lfloor C^y \rfloor_{n-p} = F(0)$. Moreover $\chi(e_i) = e_i$ so $\lfloor L^y(e_i) \rfloor_{n-p} = \lfloor A^y \circ \chi \rfloor_{n-p}(e_i)$ and we are done.

2.6 Attacks on White-Box ASASA

In this section we show that the actual security of small-block ASASA ciphers is much lower than was estimated by Biryukov *et al.*. First, we note that our main structural attack still applies. Then, we present a different heuristic attack. As mentioned in the introduction, the structural attack is not only more general, but also more efficient. The interest of the heuristic attack is merely to show a different approach to analyzing ASASA.

2.6.1 Application of the Structural Attack

The observation that our structural attack still applies to small-block ASASA instances is due to Itai Dinur, Orr Dunkelman, Thorsten Kranz and Gregor Leander [DDKL15a]. We are grateful to them for bringing this fact to our attention, and allowing us to mention it here. At first sight it may seem that our structural attack does not apply. Take for example the 16-bit instance, composed of two 8-bit S-boxes. The overall degree is bounded by $7 \cdot 7 = 49$, which would require a cube of dimension 50. This is not possible in $\mathbb{F}_2^{16}$.

The crux of the matter is that the algebraic degree is actually much lower. This result is due to Boura and Canteaut [BC13], and used by Dinur *et al.* in [DDKL15b] to show that the degree of the ASASA construction is at most $n - k$, where n is the block size and k the number of S-boxes. A cube of dimension $n - k + 1$ is then enough: the structural attack still applies, with complexity close to $n^2 2^n$. In fact this implies the structural attack applies to *any* ASASA construction, as far as degree deficiency is concerned. The only limitation is its complexity as the block size grows larger.

The observation that our structural attack extends beyond its original target as a direct consequence of generic bounds on the algebraic degree of SPN constructions is quite fruitful. It was also observed independently by Biryukov and Khovratovich [BK15] to extend the attack to ASASASA and beyond. Once again we are thankful to Dinur *et al.* for pointing this out.

2.6.2 Overview of the Heuristic Attack

We now describe a different procedure to recover the secret components of small-block schemes, thus breaking their weak white-box security (Definition 10). Our algorithm relies rather heavily on heuristics, and evaluating its efficiency requires actual implementation. We focused on the smallest white-box instance, 16-bit ASASA_{16}, whose claimed security level is 64 bits. Our algorithm was able to recover its secret components under one minute on a laptop computer.

The small block size of white-box ASASA instances makes it possible to compute the distribution of output differences for a single input difference in very reasonable time. In fact, one can compute and store the entire difference distribution table (DDT) of a 16-bit cipher using just a standard PC. For slightly bigger instances such as a 24-bit cipher, computing and storing the entire DDT is still barely possible, even though it would require 3 TB of space and 2^{48} invocations of the cipher; computing the distribution of only a few differences on the other hand remains manageable.

Remark 3. *Our attack makes use of the full codebook of the ciphers, which in general may be seen as a very strong requirement. This is however only natural in the case of attacking white-box implementations, as the user is actually* required *to be given the full codebook of the super S-boxes as part of the implementation.*

Similarly to the attack of the black box scheme, it is already enough to recover only one of the external affine (or linear) layers in order to break the security of ASASA. Indeed, this allows to reduce the cipher to either of ASAS or SASA, which can then be attacked in practical time [BS01]. Thus we focus on removing the first linear layer. In accordance with the opening remarks of Section 2.4.1, this amounts to finding the image space of each S-box through $(A^x)^{-1}$.

The general idea of the attack is to create an oracle able to recognize whether an input difference δ activates one or two S-boxes in the first S-box layer S^x. More accurately, we create a ranking function $\mathcal{F}$ such that $\mathcal{F}(\delta)$ is expected to be significantly higher if δ activates only one S-box rather than two.

We present two choices for $\mathcal{F}$ which are both heuristic but nonetheless quite efficient as shown by experiments. Both begin by computing the entire output difference distribution $D(\delta)$ for the input difference δ, *i.e.* the row corresponding to δ in the DDT. Then the value of $\mathcal{F}(\delta)$ is computed from $D(\delta)$.

Walsh Transform.

The idea behind this version of the attack is quite intuitive. If δ activates only one S-box, then after the first SA layers, two inner states computed from any two plaintexts with input difference δ are equal on the output of the inactive S-box. Hence after the first ASA layers, they are equal along $2^8 - 1$ non-zero linear masks. Since these masks only traverse a single S-box layer before the output of the cipher, linear cryptanalysis [Mat94] tells us that we can expect some linear masks to be biased at the output of the cipher. On the other hand if both S-boxes are active in the first round, no such phenomenon occurs, and linear biases on the output differences are expected to be weaker.

In order to measure this difference, we propose to compute, for every output mask a, the value $f(a) = (\sum_{x\in\{0,1\}^{16}} \langle \partial F \partial\delta(x)|a\rangle) - 2^{15}$ (where the sum is computed in $\mathbb{Z}$). That is, $2^{-15}f(a)$ is the bias of the output differences $D(\delta)$ along mask a. The function f can be computed efficiently, since it is precisely the Walsh transform of the characteristic function of $D(\delta)$, and we can use a fast Fourier transform algorithm. Then as a ranking function $\mathcal{F}$ we simply choose $\max(f)$, *i.e.* the highest bias among all output masks[21].

Number of Collisions.

It turns out that performing the Walsh transform is not truly necessary. Indeed, the number of collisions in $D(\delta)$ is higher when δ activates only 1 S-box; where by number of collisions we mean 2^{15} minus the number of distinct values in $D(\delta)$. This may be understood as a consequence of the fact that whenever δ activates a single S-box, only 2^7 output differences are possible after the first ASA layers; and depending on the properties of the active (random) S-box, the distribution between these differences may be quite uneven. Whereas if both S-boxes are active,

[21] Alternatively a less clean but more efficient ranking function in practice is to compute the number of large values of f, where a value is considered large if it is higher than 4σ, for σ the standard deviation in the case where δ activates 2 S-boxes (which needs only be computed once for some fixed random ASASA instance — in fact $\delta \approx 250$ for ASASA_{16}).

2^{15} differences are possible and the distribution is expected to be less skewed. Thus we pick as ranking function $\mathcal{F}$ the number of collisions in $D(\delta)$ in the previous sense.

Once we have chosen a ranking function $\mathcal{F}$, we simply compute the ranking of every possible input difference, sort the differences, and choose the highest 16 linearly independent differences according to our ranking. Our hope is that these differences only activate a single S-box. In a second step, we will group together differences that activate the same S-box.

2.6.3 Description of the Heuristic Attack

We now describe the attack in detail. We focus on the collision ranking function, which is slightly more efficient in practice.

First Step.

We wish to recover the individual components of the ASASA_{16} cipher $A^z \circ S^y \circ A^y \circ S^x \circ A^x$. The first step of our attack consists in finding 16 linearly independent input differences to the cipher such that only one of the two S-boxes of S^x is active. In other words, we want to find a family of differences δ_i such that for all i, $A^x(\delta_i)$ is zero in its 8 most significant or 8 least significant bits. As A^x is invertible, there are $2 \times (2^8 - 1)$ non-trivial such differences, but we need an efficient way to test if a given difference is one of them. This can be done by considering the distribution of its corresponding output differences, and counting the number of collisions, as outlined in the previous section.

That is, we attempt to recover 16 suitable differences δ_i by computing the entire DDT of ASASA_{16}, sorting the input differences by their decreasing number of collisions, and selecting the first 16 linearly independent entries. We describe this formally as Algorithm 8.

Algorithm 8: Finding a basis of differences activating only one S-box of S^x at a time

Input: An instance of ASASA_{16}
Output: A set $\mathbb{D}$ of 16 linearly independent differences activating only one S-box of S^x

1 **for** δ := *1 to* $2^{16} - 1$ **do**
2 $\mathcal{F}[\delta] := 2^{15}$ - $\#\,\mathrm{range}(\partial\mathsf{ASASA}_{16}/\partial\delta)$
3 $L :=$ sorted differences δ's in decreasing order for $\mathcal{F}$
4 $\mathbb{D} := \emptyset$
5 $i := 0$
6 **while** $\#\mathbb{D} <$ *16* **do**
7 **if** $L[i]$ *is linearly independent from* $\mathbb{D}$ **then**
8 $\mathbb{D} := \mathbb{D} \cup L[i]$
9 $i := i + 1$
10 Return $\mathbb{D}$

Complexity analysis (Alg. 8). Computing one iteration of Line 2 requires 2^{16} calls to ASASA_{16}, each one corresponding to one memory access, and at most 2^{16} words of temporary storage. The loop of Line 1 therefore requires 2^{32} calls in total and 2^{17} words of memory, including R. The sorting of Line 3 can be done in about 2^{20} accesses to R and does not require additional memory. The loop of Line 6 has negligible cost. The total time complexity is thus of

the order of 2^{32} memory accesses and the memory complexity of the order of 2^{17} words (about 2^{18} bytes in this case), which are respectively quadratic and linear in the size of the domain of ASASA_{16}

Second Step.

From the previous step, we know 16 input differences δ_i that each activate only one S-box of S^x, and we can now use this knowledge to recover the first layer A^x. Similarly to the attack of the black-box scheme of Section 2.4, it is not possible to uniquely determine A^x; but it is enough to recover one of the equivalent mappings and later choose the affine equivalent representation of the S-boxes accordingly. The consequence of this is that we only need to identify two groups of 8 linearly independent δ_i's, respectively activating the first and the second S-box. Once this is done, we may assume any value for the images of a group as long as they are linearly independent and indeed activate only one S-box.

In order to achieve our goal, we can use the fact that sums of differences of a single group still activate only one S-box, while if the differences come from two groups they obviously activate two. We can use the same method as in the first step to determine whether a sum activates one or two S-boxes, and thence we may hope to find the correct grouping by ensuring that every linear combination of a group (or equivalently every combination of two differences of a group) only activates a single S-box.

We can conveniently describe the resulting problem with a weighted graph and solve it with a simple greedy algorithm. We define the vertices of the graph as being the differences δ_i, and draw an edge between every pair of two (thus making the graph complete); the weight of the edge (δ_i, δ_j) is defined as the ranking $\mathcal{F}(\delta_i + \delta_j)$ of $\delta_i + \delta_j$. The two partitions of 8 differences are then initialized arbitrarily, with their weight defined as the sum of the weight of edges between vertices belonging to the same partition. Finally, the following process is iterated until a fixed point is reached: for any pair of vertices in different partitions, the pair is swapped if and only if this would result in increasing the weight of the partition. We describe this formally as Algorithm 9.

Complexity analysis (Alg. 9). Assuming that the computations of Line 4 have been cached when running Algorithm 8, every individual step can be computed with negligible time and memory. We then only need to estimate how many times the loop of Line 7 is executed before exiting on Line 16. First it is easy to see that the algorithm does indeed eventually terminate, as $\mathcal{W}(\mathbb{S}_1) + \mathcal{W}(\mathbb{S}_2)$ increases every time there is as swap of differences. Then, because there are at most $\binom{16}{8} = 12870$ partitions to consider, we know that the process stops within this number of iterations. In practice, it is unlikely for two S-boxes to be swapped more than once, and the observed time complexity is actually very small.

2.6.4 Experimental Results and Discussion

We implemented the previous algorithm in C++. The implementation was able to recover the first linear layer of 16-bit ASASA in a minute on average. This strongly invalidates the security level of 64 bits estimated by [BBK14].

Moreover we implemented a faster but more heuristic algorithm in the Sage formal computation language [Tea]. This variant implements an oracle $\mathcal{O}$ that predicts whether a difference δ activates one or two S-boxes, rather than a ranking function. The oracle simply calls the ranking function, and decides that δ activates a single S-box if $\mathcal{F}(\delta)$ is above a certain threshold. The threshold is determined by comparing values of $\mathcal{F}$ for δ's that activate one S-box, versus values

Algorithm 9: Computing partitions of $\mathbb{D}$ activating the same S-box

Input: A set $\mathbb{D}$ of 16 linearly independent differences activating only one S-box of S^x
Output: A partition $(\mathbb{S}_1, \mathbb{S}_2)$ of $\mathbb{D}$ such that the S-box activated by differences in $\mathbb{S}_1$ (resp. $\mathbb{S}_2$) is the same for every difference

```
Define W(S) as Σ_{i,j∈S, i≠j} W[i][j]
for i := 1 to 16 do
    for j := 1 to 16 do
        W[i][j] := F(D[i] + D[j])
S1 := {D[i], i := 0...7}
S2 := {D[i], i := 8...15}
while ∞ do
    for δ_i ∈ S1 do
        for δ_j ∈ S2 do
            S1' := ((S1 − {δ_i}) ∪ {δ_j})
            S2' := ((S2 − {δ_j}) ∪ {δ_i})
            if W(S1') + W(S2') > W(S1) + W(S2) then
                S1 := S1'
                S2 := S2'
    if Neither S1 nor S2 has been modified then
        Return (S1, S2)
```

obtained when δ activates both S-boxes (this computation only occurs once, by picking a known but random instance and performing the measure).

The algorithm then proceeds by picking random δ's and using the oracle to check whether δ activates a single S-box. Once 16 such linearly independent δ's are found the first step is complete. The second step is identical to the original algorithm. Each δ has probability 2^{-8} of activating a single S-box. Hence the expected number of δ's that will need to be tested is $16 \cdot 2^8 = 2^{24}$, which is lower than the 2^{32} DDT row computations incurred by the original algorithm. However this algorithm requires the existence of a clean threshold between the two cases for δ's.

Sage is a high-level interpreted language, which makes it significantly slower than an equivalent implementation in C++. Nonetheless due to its lower complexity, the previous algorithm was also able to succeed in a minute on average, with either choice of the ranking function $\mathcal{F}$.

All of the implementations are publicly available. A link is provided in Section 2.1.3.

Remark 4. *An interesting observation is that both ranking functions distinguish δ's that activate one or two S-boxes much less efficiently if A^y is maximum distance separable (MDS). However the attack still goes through, meaning that relying on such matrices is not a suitable countermeasure.*

2.6.5 Adapting the Algorithm to Larger White-Box Instances

The algorithm from the previous section can be adapted to larger white-box instances in a straightforwards manner. However the required computational power is higher. Beside the 16-bit instance, we have also successfully run the attack on the 20-bit instance.

Chapter 3

Efficient and Provable White-Box Primitives

3.1 Introduction

White-Box Cryptography

We have seen some constructions of white-box cryptography in the previous chapter. Our structural attacks was able to break these schemes. In this chapter, we propose our own constructions, which offer provable security guarantees. We begin with an overview of white-box cryptography, aiming to introduce the incompressibility model in which our constructions hold.

The notion of white-box cryptography was originally introduced by Chow et al. [CEJO02a, CEJO02b]. The basic goal of white-box cryptography is to provide implementations of cryptographic primitives that offer cryptographic guarantees even in the presence of an adversary having direct access to the implementation. The exact content of these security guarantees varies, and different models have been proposed.

Ideally, white-box cryptography can be thought of as trying to achieve security guarantees similar to a Trusted Execution Environment [ARM09] or trusted enclaves [CD16], purely through implementation means–in so far as this is feasible. Of course this line of research finds applications in many situations where code containing secret information is deployed in untrusted environments, such as software protection (DRM) [Wys09, Gil16].

Concretely, the initial goal in [CEJO02a, CEJO02b] was to offer implementations of the DES and AES block ciphers, such that an adversary having full access to the implementation would not be able to extract the secret keys. Unfortunately both the initial constructions and later variants aiming at the same goal (such as [XL09]) were broken [BGEC04, GMQ07, WMGP07, DMRP12, ...]: to this day no secure white-box implementation of DES or AES is known.

Beside cryptanalytic weaknesses, defining white-box security as the impossibility to extract the secret key has some drawbacks. Namely, it leaves the door open to code lifting attacks, where an attacker simply extracts the encryption function as a whole and achieves the same functionality as if she had extracted the secret key: conceptually, the encryption function can be thought of as an equivalent secret key[22].

This has led research on white-box cryptography into two related directions. One is to find new, sound and hopefully achievable definitions of white-box cryptography. The other is to propose new constructions fulfilling these definitions.

[22]This can be partially mitigated by the use of external encodings [CEJO02a]

In the definitional line of work, various security goals have been proposed for white-box constructions. On the more theoretical end of the spectrum, the most demanding property one could hope to attain for a white-box construction would be that of virtual black-box obfuscation [BGI+01]. That is, an adversary having access to the implementation of a cipher would learn no more than they could from interacting with the cipher in a black-box way (i.e. having access to an oracle computing the output of the cipher). Tremendous progress has been made in recent years in the domain of general program obfuscation, starting with [GGH+13b]. However the current state of the art is still far off practical use, both in terms of concrete security (see e.g. [Hal15b]) and performance (see e.g. an obfuscation of AES in [Zim15]).

A less ambitious goal, proposed in [DLPR13, BBK14] is that an adversary having access to the implementation of an encryption scheme may be able to encrypt (at least via code lifting), but should remain unable to decrypt. This notion is called *strong white-box* in [BBK14] and *one-wayness* in [DLPR13]. Such a goal is clearly very similar to that of a trapdoor permutation. And indeed, known constructions rely on public-key primitives. As a consequence they are no faster than public key encryption. An interesting way to partially circumvent this issue, proposed in [BBK14], is to use multivariate cryptography, where knowledge of the secret information allows encryption and decryption at a speed comparable to standard symmetric ciphers (although public key operations are quite slow). However multivariate cryptography lacks security reductions to well-established hard problems (although they are similar in flavor to MQ), and numerous instantiations have been broken, including those of [BBK14]: see [GPT15, DDKL15b, MDFK15].

Finally, on the more modest but efficiently achievable end of the spectrum, one can ask that an adversary having access to the white-box implementation cannot produce a functionally equivalent program of significantly smaller size. This notion has been called *incompressibility* in [DLPR13], *weak white-box* in [BBK14] and *space-hardness* in [BI15][23]. This definition implies in particular that it is difficult for an adversary to extract a short master key, which captures the goal of the original white-box constructions by Chow *et al.* In addition, the intent behind this approach is that large, incompressible code can more easily be made resistant to code lifting when combined with engineering obfuscation techniques [BBK14, BI15, Gil16]; and make code distribution more cumbersome for a potential hacker.

As mentioned earlier, there is no known implementation of AES or DES that successfully hides the encryption key. A fortiori there is no known way to achieve incompressibility for AES, DES or indeed any pre-existing cipher. However recent constructions have proposed new, ad-hoc, and quite efficient ciphers specifically designed to meet the incompressibility criterion [BBK14, BI15]. These constructions aim for incompressibility by relying on a large pseudo-random table hard-coded into the implementation of the cipher. Repeated calls to the table are made during the course of encryption. The idea is that, without knowledge of all or most of the table, most plaintexts cannot be encrypted. This enforces incompressibility.

In [BBK14], the table is used as an S-box in a custom block cipher design. This requires building the table as a permutation, which is achieved using an ASASA construction, alternating secret affine and non-linear layers. Unfortunately this construction was broken [DDKL15b, MDFK15]. This type of attack is completely avoided in the new SPACE construction [BI15], where the table is built by truncating calls to AES. This makes it impossible for an adversary to recover the secret key used to generate the table, based solely on the security of AES. However this also implies that the table is no longer a permutation and cannot be used as an S-box. Accordingly, in SPACE, the table is used as a round function in a generalized Feistel network. While an ad-

[23] Here, we lump together very similar definitions, although they are technically distinct. More details are provided in Section 3.2.1.

versary seeking to extract the key is defeated by the use of AES, there is no provable resistance against an adversary trying to compress the cipher.

WhiteBlock and WhiteKey: Provable Schemes

Both of the previously mentioned constructions in [BBK14, BI15] use ad-hoc designs. They are quite efficient, but cannot hope to achieve provable security. Our goal is to offer provable constructions, while retaining similar efficiency.

First, we introduce new formal definitions of incompressibility, namely weak and strong incompressibility. *Weak* incompressibility is very close to incompressibility definitions in previous work [BBK14, BI15], and can be regarded as a formalization of the space-hardness definition of [BI15]. *Strong incompressibility* on the other hand is a very demanding notion; in particular it is strictly stronger than the incompressibility definition of [DLPR13].

Our main contribution is to introduce two provably secure white-box constructions, named WhiteKey and WhiteBlock. We prove both constructions in the weak model. The bounds we obtain are close to a generic attack, and yield quite efficient parameters. Moreover we also prove WhiteKey in the strong model.

Previous work has concentrated on building white-box block ciphers. This was of course unavoidable when attempting to provide white-box implementations of AES or DES. However, it was already observed in the seminal work of Chow *et al.* that the use of white-box components could be limited to key encapsulation mechanisms [CEJO02a]. That is, the white-box component is used to encrypt and decrypt a symmetric key, which is then used to encrypt or decrypt the rest of the message. This is of course the same technique as hybrid encryption, and beneficial for the same reason: white-box component are typically slower than standard symmetric ciphers (albeit to a lesser extent than public-key schemes).

In this context, the white-box component must not necessarily be a block cipher, and our WhiteKey construction is in fact a key generator. That is, it takes a random string as input and outputs a key, which can then be used with any standard block cipher. Its main feature is that it is provably strongly incompressible. Roughly speaking, this implies it is infeasible for an adversary, given full access to an white-box implementation of WhiteKey, to produce a significantly smaller implementation that is functionally equivalent on most inputs. In fact, an efficient adversary knowing this smaller implementation cannot even use it to distinguish, with noticeable probability, outputs of the original WhiteKey instance from random.

However, WhiteKey is not invertible, and in particular it is not a block cipher, unlike prior work. Nevertheless we also propose a white-box block cipher named WhiteBlock. WhiteBlock can be used in place of any 128-bit block cipher, and is not restricted to key generation. However this comes at a cost: WhiteBlock has a more complex design, and is slightly less efficient than WhiteKey. Furthermore, it is proven only in the weak incompressibility model (essentially the same model as that of SPACE [BI15]), using a heuristic assumption. Thus WhiteKey is a cleaner and more efficient solution, if the key generation functionality suffices (which is likely in most situations where a custom white-box design can be used).

Regarding the proof of WhiteKey in the strong incompressibility model, the key insight is that what we are trying to build is essentially an entropy extractor. Indeed, roughly speaking, the table can be regarded as a large entropy pool. If an adversary tries to produce an implementation significantly smaller than the table, then the table still has high (min-)entropy conditioned on the knowledge of the compressed implementation. Thus if the key generator functions as a good entropy extractor, then the output of the key generator looks uniform to an (efficient) adversary knowing the compressed implementation.

Furthermore, for efficiency reason, we want our extractor to be local, i.e. we want our white-box key generator to make as few calls to the table as possible. Hence a local extractor does precisely what we require, and as a result our proof relies directly on previous work on local extractors [Vad04]. Meanwhile our proofs in the weak incompressibility model use dedicated combinatorial arguments.

Finally, we provide concrete instantiations of WhiteKey and WhiteBlock, named PuppyCipher and CoureurDesBois respectively. Our implementations show that these instances are quite efficient, yielding performance comparable to previous ad-hoc designs such as SPACE. Like in previous work, our instances also offer various choices in terms of the desired size of the white-box implementation.

Relation of White-Box Cryptography with Other Models

It is noteworthy that the standard formalization of white-box cryptography is very close to other models. For example, the bounded-storage model considers the problem of communicating securely given a long public random string which the adversary is unable to store. Indeed, up to renaming, it is essentially the same as the incompressibility of the key, and one of our design is inspired by a solution proposed to this problem [Vad04]. Another model, even stronger than incompressibility, is intrusion-resilience [Dzi06]. The goal is to communicate securely, even when a virus may output any data to the adversary during the computations of both parties, as long as the total data leaked is somewhat smaller than the key size. The disadvantage of this model is that it requires rounds of communication (e.g. 9 rounds in [CDD+07]), while white-box solutions need only add some computations.

An independent work by Bellare, Kane and Rogaway was accepted to Crypto 2016 [BKR16], whose underlying goal and techniques are similar to our strong incompressibility model, and the WhiteKey construction in particular. Although the setting of [BKR16] is different and no mention is made of white-box cryptography, the design objective is similar. The setting considered in [BKR16] is that of the bounded-retrieval model [ADW09], and the aim is to foil key exfiltration attempts by using a large encryption key. The point is that encryption should remain secure in the presence of an adversary having access to a bounded exfiltration of the big key. The exfiltrated data is modeled as the output of an adversarially-defined function of the key with bounded output.

The compressed implementation plays the same role in our definition of strong incompressibility: interestingly, our strong model almost matches big-key security in that sense (contrary to prior work on incompressible white-box cryptography, which is closer to our weak model). Relatively minor differences include the fact that we require a bound on the min-entropy of the table/big key relative to the output of the adversarially-defined function, rather than specifically the number of bits; and we can dispense with a random oracle at the output because we do not assume that the adversary is able to see generated keys directly, after the compression phase. A notable difference is how authenticity is treated: we require that the adversary is unable to encrypt most plaintexts, given the compressed implementation; whereas the authors of [BKR16] only enforce authenticity when there is no leakage. A word-based generalization of the main result in [BKR16], as mentioned in discussion of that paper, would be very interesting from our perspective, likely allowing better bounds for WhiteKey in the strong incompressibility model. Proofs of weak incompressibility, the WhiteBlock construction, as well as the concrete design of the WhiteKey instance using a variant of the extractor from [CMNT11], are unrelated.

As mentioned earlier in the introduction, the design of local extractors is also directly related to our proof in the strong incompressibility model, most notably [Vad04].

3.2 Models

3.2.1 Context

As noted in the introduction, the term white-box cryptography encompasses a variety of models, aiming to achieve related, but distinct security goals. Here we are interested in the *incompressibility* model. The basic goal is to prevent an attacker who has access to the full implementation of a cipher to produce a more compact implementation.

Incompressibility has been defined under different names and with slight variations in prior work. It is formally defined as (λ, δ)-*Incompressibility* in [DLPR13]. A very similar notion is called *weak white-box* in [BBK14], and *space-hardness* in [BI15]. In [BBK14], the *weak white-box* model asks that an efficient adversary, given full access to the cipher implementation, is unable to produce a new implementation of the same cipher of size less than some security parameter T. In [BI15], this notion is refined by allowing the adversary-produced implementation to be correct up to a negligible proportion 2^{-Z} of the input space. Thus a scheme is considered (T, Z)-*space-hard* iff an efficient adversary is unable to produce an implementation of the cipher of size less than T, that is correct on all but a proportion 2^{-Z} of inputs. This is essentially equivalent to the (λ, δ)-*incompressibility* definition of [DLPR13], where λ and δ play the respective roles of T and 2^{-Z}.

In this work, we introduce and use two main notions of incompressibility, which we call *weak* and *strong* incompressibility. Weak incompressibility may be regarded as a formalization of space-hardness from [BI15]. As the names suggest, strong incompressibility implies weak incompressibility (cf. Section 3.2.6). The point of strong incompressibility is that it provides stronger guarantees, and is a natural fit for the WhiteKey construction.

3.2.2 Preliminary Groundwork

To our knowledge, all prior work that has attempted to achieve white-box incompressibility using symmetric means[24] has followed a similar framework. The general idea is as follows. The white-box implementation of the cipher is actually a symmetric cipher that uses a large table as a component. The table is hard-coded into the implementation. To an adversary looking at the implementation, the table looks uniformly random. An adversary attempting to compress the implementation would be forced to retain only part of the table in the compressed implementation. Because repeated pseudo-random calls to the table are made in the course of each encryption and decryption, any implementation that ignores a significant part of the table would be unable to encrypt or decrypt accurately most messages. This enforces incompressibility.

To a legitimate user in possession of the shared secret however, the table is not uniformly random. It is in fact generated using a short secret key. Of course this short master key should be hard to recover from the table, otherwise the scheme could be dramatically compressed.

Thus a white-box encryption scheme is made up of two components: an *encryption scheme*, which takes as input a short master secret key and uses it to encrypt data, and a white-box *implementation*, which is functionally equivalent, but does not use the short master secret key directly. Instead, it uses a large table (which can be thought of as an equivalent key) that has been derived from the master key. This situation is generally formalized by defining a white-box scheme as an encryption scheme together with a *white-box compiler*, which produces the white-box implementation of the scheme.

[24] This excludes the incompressible construction from [DLPR13], which is based on a modified RSA.

Definition 11 (encryption scheme)**.** *An encryption scheme is a mapping* $E : \mathcal{K} \times \mathcal{R} \times \mathcal{P} \to \mathcal{C}$*, taking as input a key* $K \in \mathcal{K}$*, possibly some randomness* $r \in \mathcal{R}$*, and a plaintext* $P \in \mathcal{P}$*. It outputs a ciphertext* $C \in \mathcal{C}$*. Furthermore it is required that the decryption scheme be invertible, in the sense that there exists a decryption function* $D : \mathcal{K} \times \mathcal{C} \to \mathcal{P}$ *such that* $\forall K, R, P, D(K, E(K, R, P)) = P$*.*

Definition 12 (white-box encryption scheme)**.** *A* white-box encryption scheme *is defined by a pair of two encryption schemes:*

$$E_1 : \mathcal{K} \times \mathcal{R} \times \mathcal{P} \to \mathcal{C}$$
$$E_2 : \mathcal{T} \times \mathcal{R} \times \mathcal{P} \to \mathcal{C}$$

together with a white-box compiler $C : \mathcal{K} \to \mathcal{T}$*, such that for all* $K \in \mathcal{K}$*,* $E_1(K, \cdot, \cdot)$ *is functionally equivalent to* $E_2(C(K), \cdot, \cdot)$*.*

In the definition above, E_1 can be thought of as a standard encryption scheme relying on a short (say, 128-bit) master key K, while E_2 is its white-box implementation, relying on a large table T derived from K. To distinguish between E_1 and E_2, we will sometimes call the first scheme the *cipher*, and the second the (white-box) *implementation*.

3.2.3 Splitting the Adversaries

A white-box scheme is faced with two distinct adversaries:

- The *black-box* adversary attempts to attack the cipher itself with respect to some standard black-box security notion.
- The *white-box* adversary attempts to break incompressibility by producing a smaller implementation of the encryption scheme.

The black-box adversary can be evaluated with respect to standard security notions such as IND-CCA. The specificity of white-box schemes is of course the second adversary, on which we now focus. The white-box adversary itself can be decomposed into two distinct adversaries:

- The *compiler* adversary attempts to recover the master key K of E_1 given the implementation E_2. This is the adversary that succeeds in the cryptanalyses of e.g. [BGEC04, GMQ07, DDKL15b, MDFK15]. More generally this adversary attempts to distinguish $C(K)$ for $K \leftarrow^{\$} \mathcal{K}$ from a uniform element of $\mathcal{T}$.
- Finally, the *implementation* adversary does not attempt to distinguish T, and instead regards T as uniformly random. She focuses purely on the white-box implementation E_2. She attempts to produce a functionally equivalent (up to some error rate specified by the security parameters), but smaller implementation of E_2.

Nicely enough, the three black-box, compiler and implementation adversaries target respectively the E_1, C, and E_2 components of the white-box scheme (hence their name). Of course the two white-box adversaries (targeting the compiler and implementation) break incompressiblity, so they can be captured by the same security definition (as in [DLPR13]). However it is helpful to think of the two as separate adversaries, especially because they can be thwarted by separate mechanisms. Moreover it is clear that resistance to both adversaries implies incompressibility (the dichotomy being whether the table can be efficiently distinguished from random).

The authors of [BI15] introduce a new general method to make sure that the compiler adversary fails, i.e. $C(T)$ is indistinguishable from uniform. Namely, they propose to generate the table T by truncating the output of successive calls to AES (or some other fixed block cipher). In this scenario the master key K of E_1 is the AES key. Assuming AES cannot be distinguished from a uniformly random permutation, and the truncated output is (say) at most half of the original cipher, then the table T is indistinguishable from a random function.

3.2.4 Weak Incompressibility

As noted in the previous section, using the technique from [BI15], defeating the compiler adversary is quite easy, and relies directly and provably on the security of a standard cipher. As a result, our security definition (and indeed, our constructions) focus on the *implementation* adversary.

The weak incompressibility notion we define below is very close to the space-hardness notion of [BI15], indeed it is essentially a formalization of it. Like in [BBK14, BI15], the definition is specific to the case where the table T is actually a table (rather than an arbitrary binary string). As such the table is a function (or permutation) $T : \mathcal{I} \to \mathcal{O}$, and can be queried on inputs $i \in \mathcal{I}$.

We write weak incompressibility as ENC-TCOM: *ENC* reflects the fact that the adversary's ultimate goal is to *encrypt* a plaintext. *TCOM* stands for *table-compressed*, as the adversary is given access to a compressed form of the table. This is of course weaker than being given access to a compressed implementation defined in an arbitrary adversarially-defined way, as will be the case in the next section.

In the following definition, the encryption scheme should be thought of as the white-box implementation E_2 from the previous sections. In particular the "key" can be thought of as a large table.

Definition 13 (Weak incompressibility, ENC-TCOM)**.** *Let $E : \mathcal{T} \times \mathcal{R} \times \mathcal{P}$ denote an encryption scheme. Let s, λ denote security parameters. Let us further assume that the key $T \in \mathcal{T}$ is a function $T : \mathcal{I} \to \mathcal{O}$ for some input and output sets $\mathcal{I}$ and $\mathcal{O}$. The encryption scheme is said to be τ-secure for (s, λ, δ)-*weak incompressibility *iff, with probability at least $1 - 2^{-\lambda}$ over the random choice of $T \in \mathcal{T}$ (performed in the intial step of the game), the probability of success of the adversary running in time τ in the following game is upper-bounded by δ.*

1. *The challenger $\mathcal{B}$ picks $T \in \mathcal{T}$ uniformly at random.*
2. *For $0 \leq i < s$, the adversary chooses $q_i \in \mathcal{I}$, and receives $T(q_i)$ from the challenger. Note that the queries are adaptive.*
   At this point the adversary is tasked with trying to encrypt a random message:
3. *The challenger chooses $P \in \mathcal{P}$ uniformly at random, and sends P to the adversary.*
4. *The adversary chooses $C \in \mathcal{C}$. The adversary wins iff C decrypts to P (for key T).*

In other words, a scheme is (s, λ, δ)-weakly incompressible iff any adversary allowed to adaptively query up to s entries of the table T can only correctly encrypt up to a proportion δ of plaintexts (except with negligible probability $2^{-\lambda}$ over the choice of T). Note that (s, λ, δ)-weak incompressibility matches exactly with $(s, -\log(\delta))$-space-hardness in [BI15]. The only difference is that our definition is more formal, as is necessary since we wish to provide a security proof. In particular we specify that the adversary's queries are adaptive.

It should also be noted that the adversary's goal could be swapped for e.g. indistinguishability in the definition above. The reason we choose a weaker goal here is that it matches with prior white-box defintions, namely space-hardness [BI15] and weak white-box [BBK14]. Moreover it makes sense in white-box contexts such as DRM, where a hacker is attempting to create a rogue encryption or decryption algorithm: the point is that such an algorithm should fail on most inputs, unless the adversary has succeeded in extracting the whole table (or close to it), and the algorithm includes it.

It is noteworthy that in our definitions, "incompressibility" is captured as a power given to the adversary. The adversary's goal, be it encryption or indistinguishability, can be set independently of the specific form of compressed implementation she is allowed to ask for. This makes the definition conveniently modular, in the spirit of standard security notions such as IND-CCA.

3.2.5 Strong Incompressibility

We now introduce a stronger notion of incompressibility. This definition is stronger in two significant ways.

1. First, there is no more restriction on how the adversary can choose to compress the implementation. In the case of weak incompressibility, the adversary was only allowed to "compress" by learning a portion of the table. With strong incompressbility, she is allowed to compress the implementation in an arbirary way, as long as the table T retains enough randomness from the point of view of the adversary (*e.g.* she does not learn the whole secret).

2. Second, the adversary's goal is to distinguish the output of the encryption function from random, rather than being able to encrypt. This requirement may be deemed too demanding for some applications, but can be thought of as the best form of incompressibility one can ask for.

We denote strong incompressibility by IND-COM because the ultimate goal of the adversary is to break an indistinguishability game (IND), given a compressed (or compact) implementation of their choice (COM). We actually give more power to the adversary than this would seem to imply, as the adversary is also given the power to query plaintexts of her choice after receiving the compressed implementation.

Note that in the following definitions, f is not computationally bounded, so generating the tables via a pseudorandom function is not possible.

Definition 14 (Strong incompressibility, IND-COM)**.** *Let $E : \mathcal{T} \times \mathcal{R} \times \mathcal{P}$ denote an encryption scheme. Let μ denote a security parameter. Let us further assume that the key $T \in \mathcal{T}$ is chosen according to some distribution D (typically uniform). The scheme E is said to be (τ, ϵ)-secure for* μ-strong incompressibility *iff the advantage of an adversary $\mathcal{A}$ running in time τ and playing the following game is upper-bounded by ϵ.*

1. *The adversary chooses a set $\mathcal{S}$ and a function $f : \mathcal{T} \to \mathcal{S}$, subject only to the condition that for all $s \in \mathcal{S}$, the min-entropy of the variable T conditioned on $f(T) = s$ is at least μ. The function f should be thought of as a compression algorithm chosen by the adversary.*

2. *Meanwhile the challenger $\mathcal{B}$ picks $T \in \mathcal{T}$ according to the distribution D (thus fixing an instance of the encryption scheme).*

3. *The adversary receives $f(T)$. At this point the adversary is tasked with breaking a standard IND-CPA game, namely:*

4. *The adversary may repeatedly choose any plaintext $P \in \mathcal{P}$, and learns $E(T, R, P)$.*

5. *The adversary chooses two plaintext messages $P_0, P_1 \in \mathcal{P}$, and sends (P_0, P_1) to $\mathcal{B}$.*

6. *The challenger chooses a uniform bit $b \in \{0, 1\}$, randomness $R \in \mathcal{R}$, and sends $E(T, R, P_b)$ to the adversary.*

7. *The adversary computes $b' \in \{0, 1\}$ and wins iff $b' = b$.*

It may be tempting, in the previous definition, to allow the adversary to first query E, and choose f based on the answers. However it is not necessary to add such interactions to the definition: indeed, such interactions can be folded into the function f, which can be regarded as an arbitrary algorithm or protocol between the adversary and the challenger having access to T. The only limitation is that the min-entropy of T should remain above μ from the point of view of the adversary. It is clear that a limitation of this sort is necessary, otherwise the adversary could simply learn T.

Furthermore, while a definition based on min-entropy may seem rather impractical, it encompasses as a special case the simpler space-hard notion of [BI15]. In that case the table T is a uniform function, and f outputs a fixed proportion $1/4$ of the table. The min-entropy μ is then simply the number of unknown output bits of the table (namely $3/4$ of its output).

The WhiteKey construction we define later on is actually a key generator. That is, it takes as input a uniformly random string and outputs a key. The strong incompressibility definition expects an encryption scheme. In order for the WhiteKey key generator to fulfill strong incompressibility, it needs to be converted into an encryption scheme. This is achieved generically by using the generated key (the output of WhiteKey) with a conventional symmetric encryption scheme, as in a standard hybrid cryptosystem. For instance, the plaintext can be XORed with the output of a pseudorandom generator whose input is the generated key. Strictly speaking, when we say that WhiteKey satisfies strong incompressilibility, we mean that this is the case when WhiteKey is used as a key generator in combination with any conventional symmetric encryption process.

Note that this does not enforce authenticity. For instance, if the generated key is used as an input to a stream cipher, forgeries are trivial. More generally it is not possible to prevent existential forgeries, as the adversarially compressed implementation could include any fixed arbitrary valid ciphertext. However universal forgeries can be prevented. This is naturally expressed by the following model. The model actually captures the required goal in previous definitions of incompressibility, in fact the model as a whole is essentially equivalent to incompressibility in the sense of [DLPR13].

Definition 15 (Encryption incompressibility, ENC-COM)**.** *Let $E : \mathcal{T} \times \mathcal{R} \times \mathcal{P}$ denote an encryption scheme. Let μ denote a security parameter. Let us further assume that the key $T \in \mathcal{T}$ is chosen according to some distribution D (typically uniform). The scheme E is said to be (τ, ϵ)-secure for μ-*strong incompressibility *iff the advantage of an adversary $\mathcal{A}$ running in time τ and playing the following game is upper-bounded by ϵ.*

1. *The adversary chooses a distribution $\mathcal{D}$ with min-entropy at least μ.*

2. *The adversary chooses a set $\mathcal{S}$ and a function $f : \mathcal{T} \to \mathcal{S}$, subject only to the condition that for all $s \in \mathcal{S}$, the min-entropy of the variable T conditioned on $f(T) = s$ is at least μ. The function f should be thought of as a compression algorithm chosen by the adversary.*

3. *Meanwhile the challenger $\mathcal{B}$ picks $T \in \mathcal{T}$ according to the distribution D (thus fixing an instance of the encryption scheme).*

4. *The adversary receives $f(T)$.*
   At this point the adversary is tasked with forging a message, namely:

5. *The adversary samples a plaintext $M \in \mathcal{P}$ from the distribution $\mathcal{D}$.*

6. *The adversary may repeatedly choose any plaintext $P \in \mathcal{P}$, and learns $E(T, R, P)$.*

7. *The adversary wins iff she can computes a $C \in \mathcal{C}$ such that $D(T, C) = M$.*

This model can also be fulfilled by the WhiteKey scheme, if we derive the required randomness from $H(P) + r$ where H is a random oracle, P is the plaintext, and r is uniform value of μ bits put in the encryption. The decryption starts by recovering the key, and then checks if the randomness used came from $H(P', r)$ where P' is the decrypted plaintext. This naturally makes any encryption scheme derived from a key generator resistant to universal forgeries.

Remark that it is necessary in the model to have the forged message generated independently of $f(\mathcal{T})$, otherwise one can simply put an encryption of the message in $f(T)$.

Finally, observe that ENC-COM is stronger than ENC-TCOM, as ENC-TCOM it is the special case of ENC-COM where the adversary's chosen function f does nothing more than querying T on some adaptatively chosen inputs, and returning the outputs.

3.2.6 Strong Incompressibility Implies Incompressibility

In this section, we show that the ENC-COM security definition from Section 3.2.5 implies incompressibility as defined in [DLPR13]. More precisely, recall from Section 3.2 that we split the white-box adversary into two separate entities, called the compiler and implementation adversaries. In a nutshell, we show that if both adversaries fail, then the adversary against incompressibility in the sense of [DLPR13] also fails. Thus, we actually prove two things: 1) it was indeed legitimate to split the white-box incompressibility adversary into compiler and implementation adversaries, and 2) resistance against both adversaries (the second of which is captured by strong incompressibility) implies incompressibility in the sense of [DLPR13]. Since resistance against the compiler adversary is easy using the technique from [BI15] (cf. Section 3.2), the statement (2) can be summed up as "strong incompressibility implies incompressibility".

The main point of this result is that incompressibility in the sense of [DLPR13] captures very directly a natural notion of a program being incompressible. Meanwhile, our definition of strong incompressibility, albeit more expressive and general, may not appear at first as saying anything about the encryption scheme being incompressible. The following proof clarifies this point.

In order to establish the result, we first need to formally defined the compiler adversary. This can be done quite naturally as follows.

Definition 16 (Compiler security)**.** *A white-box compiler $C : \mathcal{K} \to \mathcal{T}$ (as defined in Definition 12) is said to be (τ, ϵ)-secure iff the advantage of an adversary running in time τ and playing the following game is upper-bounded by ϵ. The game is a simple real-or-random game where the adversary interacts with a challenger.*

1. *The challenger chooses a uniform bit $b \in \{0,1\}$. If $b = 0$, the challenger picks T uniformly in $\mathcal{T}$. If $b = 1$, the challenger picks K uniformly in K, and sets $T = C(K)$. In both cases the challenger sends T to the adversary.*

2. *Upon receiving T, the adversary computes a bit b' and wins iff $b' = b$.*

We now recall the incompressibility definition of [DLPR13], adapated to our setting. The version given below is slightly simpler than the original, because we specialize it to a symmetric setting, where it is assumed that an adversary given access to a white-box implementation can encrypt and decrypt at will. Thus we ignore the difference between CPA and CCA versions of the definition. We also disregard recompilation attacks, although this section could be extended in a straightforward way to include that case (by modifying the definition of compiler security to allow recompilations). We also slightly adapt the definition to account for the fact that we allow randomness in the encryption scheme. Finally we also add the constraint that the running time of the program P output by the adversary is upper-bounded by the running time τ of the adversary. This is to eliminate trivial and generic attacks in the case where P is computationally unbounded (*e.g.* part of T is hard-coded in P and P uses it to brute force the master secret K).

Definition 17 ((λ, δ)-incompressibility)**.** *Let (E_1, E_2, C) be a white-box scheme, with cipher $E_1 : \mathcal{K} \times \mathcal{R} \times \mathcal{P} \to \mathcal{C}$, white-box implementation $E_2 : \mathcal{T} \times \mathcal{R} \times \mathcal{P} \to \mathcal{C}$ and compiler $C : \mathcal{K} \to \mathcal{T}$. The scheme is said to be (τ, ϵ)-secure for* (λ, δ)-incompressibility *iff the success probability of an adversary running in time τ and playing the following game is upper-bounded by ϵ.*

1. *The challenger chooses K uniformly in $\mathcal{K}$, and sends $C(K)$ to the adversary.*

2. *The adversary creates a program $p : \mathcal{P} \to \mathcal{C}$. The adversary wins iff the size of p is less than λ, and for all but a fraction δ of plaintexts P, $p(P)$ decrypts to P.*

Thus informally, (λ, δ)-incompressibility says that an adversary, given a white-box implementation of an instance of the scheme, is unable to produce with noticeable probability a program of size less than λ that is functionally equivalent on a fraction δ of inputs, *i.e.* that correctly encrypts a fraction δ of plaintexts.

For simplicity, we assume below that tables $T \in \mathcal{T}$ are uniformly random functions of a fixed size, and denote by $|T|$ the total number of output bits of a table (equivalently, the total number of bits necessary to encode T).

Theorem 1. *Let (E_1, E_2, C) denote a white-box encryption scheme. Assume that E_2 is $(\tau, \epsilon - \epsilon')$-secure for $(|T| - \lambda, -\log(\epsilon), \delta)$-ENC-COM, and C is (τ', ϵ')-secure for compiler security. Then the scheme is (τ, ϵ)-secure for (λ, δ) incompressibility.*

Proof. Assume we have an adversary $\mathcal{A}$ breaking (τ, ϵ)-security for (λ, δ) incompressibility. Also assume that C is (τ', ϵ')-secure. Then we are going to build an adversary breaking $(\tau, \epsilon - \epsilon')$-security for $(|T| - \lambda, -\log(\epsilon), \delta)$-ENC-COM.

To see this, first consider the adversary $\mathcal{A}$, playing the incompressibility game G_{INC}. We now replace G_{INC} with G'_{INC}, where the only difference is that in the step 1 of G_{INC}, instead of choosing K and sending $T = C(K)$ to the adversary, the challenger directly chooses T uniformly at random in $\mathcal{T}$ and sends it to the adversary. Distinguishing the views of the adversary in G_{INC} and G'_{INC} is exactly expressed by the compiler security game. As a result, provided $\tau' \geq \tau$, the same adversary $\mathcal{A}$ breaks $(\tau, \epsilon - \epsilon')$-security for G'_{INC}.

We now build an adversary $\mathcal{A}'$ against ENC-COM. The function f chosen by $\mathcal{A}'$ is actually the adversary $\mathcal{A}$ itself, viewed as a function $f = \mathcal{A}$ of T outputting a program p. Since the size

of p is upper-bounded by λ and T is uniform, the min-entropy of T conditioned on $p = f(T)$ is at least $|T| - \lambda$. Moreover, given p, $\mathcal{A}'$ is able to correctly encrypt a fraction more than δ of plaintexts, except with probability at most ϵ, simply by running the program p on any challenge plaintext. This is by definition of G'_{INC}. It follows that $\mathcal{A}'$ breaks $(\tau, \epsilon - \epsilon')$-security for $(|T| - \lambda, -\log(\epsilon), \delta)$-ENC-COM security definition. □

3.3 Constructions

In this section, we present two constructions that are provably secure in the weak white-box model of Section 3.2: the WhiteBlock block cipher, and the WhiteKey key generator. WhiteKey is also provable in the strong model. We also propose PuppyCipher and CoureurDesBois as concrete instantiations of each construction, using the AES as underlying primitive.

3.3.1 The WhiteBlock Block Cipher

The general idea of WhiteBlock is to build a Feistel network whose round function uses calls to a large table T. An adversary who does not extract and store a large part of this table should be unable to encrypt most plaintexts. For that purpose, it is important that the inputs of table calls be pseudo-random, or at least not overly structured. Otherwise the adversary could attempt to store a structured subset of the table that exploits this lack of randomness. In WhiteBlock, the pseudo-randomness of table calls is enforced by interleaving calls to a block cipher between each Feistel round. The point of our proof is to show that this introduces enough randomness that the adversary is essentially unable to select a subset of the table to store that would significantly outperform just storing a random subset of the same size (in terms of how many plaintexts this subset of the table allows the adversary to encrypt).

Concretely, WhiteBlock defines a family of block ciphers with blocks of size $b = 128$ bits, and a key of size $\kappa = 128$ bits[25]. The family is parameterized with a *size* parameter which corresponds to the targeted size of a white-box implementation. In principle, this size can be anything from a few dozen bytes up to $\approx 2^{64}$ bytes, but we will mostly restrict this description to the smallest case considered in this article, which has an implementation of size 2^{21} bytes.

Formally, we define one round of WhiteBlock (with tables of input size 16 bits) as follows. Let $\mathcal{A}_k$ denote a call to the block cipher $\mathcal{A}$ with key k, and $\mathcal{T}_i : \{0,1\}^{16} \to \{0,1\}^{64}$ denote the i-th table. The Feistel round function is defined by:

$$\begin{aligned}&\mathbb{F} : \{0,1\}^{64} \to \{0,1\}^{64},\\ &x_{63} \dots x_0 \mapsto \mathcal{T}_3(x_{63} \dots x_{48}) \oplus \mathcal{T}_2(x_{47} \dots x_{32}) \oplus \mathcal{T}_1(x_{31} \dots x_{16}) \oplus \mathcal{T}_0(x_{15} \dots x_0)\end{aligned}$$

and one round of WhiteBlock with key k is defined as:

$$\begin{aligned}\mathcal{R}_k : \{0,1\}^{128} &\to \{0,1\}^{128}\\ x_{127} \dots x_0 &\mapsto \mathcal{A}_k\left(((x_{127} \dots x_{64}) \oplus \mathbb{F}(x_{63} \dots x_0)) || x_{63} \dots x_0\right)\end{aligned}.$$

A full instance of WhiteBlock is then simply the composition of a certain number of independently-keyed round functions, with the addition of one initial top call to $\mathcal{A}$: $\text{WhiteBlock}_{k_0,\dots k_r} : \{0,1\}^{128} \to \{0,1\}^{128}, x \mapsto \mathcal{A}_{k_r} \circ \mathcal{R}_{k_{r-1}} \circ \dots \circ \mathcal{R}_{k_0}(x)$. We give an illustration of this construction (omitting the outer sandwiching calls to $\mathcal{A}$) in Figure 3.1.

[25] This generalizes well to other sizes.

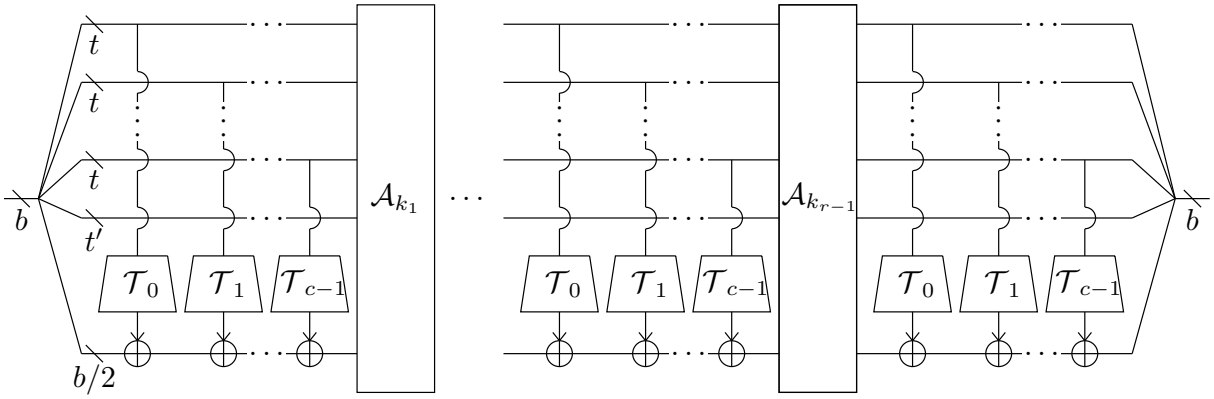

Figure 3.1: The WhiteBlock construction, with tables on t bits, without the outer calls to $\mathcal{A}$. We have $t' = (b/2) \mod s$, $c = \lfloor (b/2)/t \rfloor$.

Constructing the Tables.

For WhiteBlock instances with small tables, the most efficient way to implement the cipher is simply to use the white-box implementation, *i.e.* use a table-based implementation of $\mathbb{F}$ (this will be clear from the results of Section 3.5). In that case, it is easy to generate the tables "perfectly" by drawing each entry uniformly at random, either by using a suitable source of randomness (in that case, no one may be able to compress the tables) or by using the output of a cryptographically-strong PRG seeded with a secret key. In the latter case, the owner of the secret knows how to compactly represent the tables, but this knowledge seems to be hard to exploit in a concrete implementation.

For larger instances, it is not true anymore that the fastest implementation is table-based, and it may be useful in some contexts to be able to compute the output of a table more efficiently than by querying it. Surely, if one knows how to compactly represent a table, it is desirable that he would be able to do so, at least for large tables. In that respect, drawing the tables at random would not be satisfactory anymore.

Consequently, the tables used in WhiteBlock are generated as follows. Let again $\mathcal{T}_i : \{0,1\}^{16} \to \{0,1\}^{64}$ be such a table (in the 16-bit case), then an instance of it is defined with two parameters $k \in \{0,1\}^{128}$, $c \in \{0,1\}^{128-16}$ as $\mathcal{T}(x) := \lfloor \mathcal{A}_k(c||x) \rfloor_{64}$, with $\lfloor \cdot \rfloor_{64}$ denoting the truncation to the 64 lowest bits.

An instance of WhiteBlock can thus always be described and implemented compactly when knowing k and c. Of course this knowledge is not directly accessible in a white-box implementation, where a user would only be given the tables as a whole.

Concrete Parameters for Various Instances of WhiteBlock.

We need to address two more points before finishing this description of WhiteBlock in the ideal case: 1) given the size of the tables, how many rounds r are necessary to obtain a secure white-box construction; 2) how to generate the multiple round keys $k_0, \ldots k_r$. The answer to 1) is provided by the analysis of the construction done in Section 3.4.2. By instantiating the formula of Thm. 3 with concrete parameters, we obtain the results given in Tbl. 3.1. As for 2), we further delay the discussion to Section 3.5 dealing with concrete speed-ups.

Instance	WB size	# Tables/round	WB security	#rounds
WhiteBlock 16	2^{21} B	4	112 bits @ 1/4	18
WhiteBlock 20	$2^{24.6}$ B	3	108 bits @ 1/4	23
WhiteBlock 24	2^{28} B	2	104 bits @ 1/4	34
WhiteBlock 28	2^{32} B	2	100 bits @ 1/4	34
WhiteBlock 32	2^{36} B	2	96 bits @ 1/4	34

Table 3.1: Number of rounds for WhiteBlock instances with tables of selected input sizes from $t = 16$ to 32 bits, at a white-box security level of $128 - t$ bits for a compression factor of 4. Black-box security is 128 bits in all cases.

PuppyCipher: WhiteBlock in Practice.

So far WhiteBlock has been described from an abstract point of view, where all components are derived from a block cipher $\mathcal{A}$. In practice, we need to specify a concrete cipher; we thus define the PuppyCipher family as an instatiation of WhiteBlock using AES128 [DR02] for the underlying block cipher. Furthermore, though relying to a secure block cipher is an important argument in the proof of the construction, one can wish for a less expensive round function in practice. Hence we also define the lighter, more aggressive alternative "Hound" which trades provable security for speed. The only differences between PuppyCipher and Hound are:

1. The calls to the full AES128 are traded for calls to AES128 reduced to five rounds (this excludes the calls in the table generation, which still use the full AES)
2. The round keys $k_r \dots k_0$ used as input to $\mathcal{A}$ are simply derived from a unique key K as $k_i := K \oplus i$. Note that using a tweakable cipher such as KIASU would also be possible [JNP14].

In Section 3.5, we discuss the efficiency of PuppyCipher and Hound implemented with the AES instructions, for tables of 16, 20, and 24-bit inputs.

3.3.2 The WhiteKey Key Generator

In WhiteBlock, we generated pseudo-random calls to a large table by interleaving a block cipher between table calls. If we are not restricted by the state size of a block cipher, generating pseudo-random inputs for the table is much easier: we can simply use a pseudo-random generator. From a single input, we are then able to generate a large number of pseudo-random values to be used as inputs for table calls. It then remains to combine the outputs of these table calls into a single output value of appropriate size. For this purpose, we use an entropy extractor. More details on our choice of extractor are provided in the design rationale below.

We now describe the WhiteKey function family, which can in some way be seen as an unrolled and parallel version of WhiteBlock, with some adjustments. As with WhiteBlock, we describe the main components of WhiteKey for use with a 128-bit block cipher and tables of 16-bit inputs, but this generalizes easily to other sizes.

Thus WhiteKey uses a table $T : \{0,1\}^{16} \to \{0,1\}^{128}$. Let n denote the number of table calls (which will be determined later on by security proofs), $t \triangleq \lceil n/8 \rceil$ and $d \triangleq \lceil \sqrt{n} \rceil$. At a high level, the construction of WhiteKey can be described by the following process: 1) from a

random seed, generate t 128-bit values using a block cipher $\mathcal{A}$ with key k in counter mode; 2) divide each such value into eight 16-bit words; 3) use these words as n inputs to the table T (possibly ignoring from one to seven of the last generated values), resulting in n 128-bit values $Q_{i,j}, 0 \leq i,j \leq d = \lceil\sqrt{n}\rceil$ (if n is not a square, the remaining values $Q_{i,j}$ are set to zero); 4) from a random seed, generate d 128-bit values a_i and d 128-bit values b_i using $\mathcal{A}$ with key k' in counter mode; 5) the output of WhiteKey is $\sum_{i,j} Q_{i,j} \cdot a_i \cdot b_j$, the operations being computed in $\mathbb{F}_{2^{128}}$.

Let us now define this more formally. We write $\mathcal{A}_k^t(s)$ for the t first 128-bit output blocks of $\mathcal{A}$ in counter mode with key k and initial value s. We write $\mathcal{C}_n$ for the parallel application of $n \leq 8 \times t$ tables $\mathcal{T} : \{0,1\}^{16} \to \{0,1\}^{128}$ (written here in the case $n = 8 \times t$ for the sake of simplicity):

$$\begin{aligned} \mathcal{C}_n : \{0,1\}^{t\times 128} &\to \{0,1\}^{n\times 128} \\ x_{t128-1}x_{t128-2}\dots x_0 &\mapsto \mathcal{T}(x_{t128-1}\dots x_{t128-16})||\dots||\,\mathcal{T}(x_{15}\dots x_0) \end{aligned}$$

We write $\mathcal{S}_n$ for the "matrixification" mapping; taking $d := \lceil\sqrt{n}\rceil$ (here with $n = 57$, for a not too complex general case):

$$\begin{aligned} \mathcal{S}_n : \{0,1\}^{n\times 128} &\to \mathcal{M}_d(\mathbb{F}_{2^{128}}) \\ x_{n128-1}x_{n128-2}\dots x_0 &\mapsto \begin{pmatrix} x_{127}\dots x_0 & x_{255}\dots x_0 & \cdots & x_{1023}\dots x_{896} \\ \vdots & \vdots & \ddots & \vdots \\ x_{n128-1}\dots x_{n128-128} & 0 & \cdots & 0 \end{pmatrix}. \end{aligned}$$

Finally, we write $\mathcal{E}$ the "product" mapping:

$$\begin{aligned} \mathcal{E} : \mathbb{F}_{2^{128}}^d \times \mathbb{F}_{2^{128}}^d \times \mathcal{M}_d(\mathbb{F}_{2^{128}}) &\to \mathbb{F}_{2^{128}} \\ a, b, Q &\mapsto \textstyle\sum_{i,j} Q_{i,j} \cdot a_i \cdot b_j \end{aligned}$$

We can then describe an instance of WhiteKey parametered by (k_1, s_1, k_2, s_2) over t and n values as $\text{WhiteKey}^{t,n}_{k_1,s_1,k_2,s_2} := \mathcal{E} \circ \mathcal{A}^d_{k_2}(s_2) \circ \mathcal{A}^d_{k_2}(s_2 + d) \circ \mathcal{S}_n \circ \mathcal{C}_n \circ \mathcal{A}^t(k_1, s_1)$ (using a Curried version of $\mathcal{E}$ for simplicity of notations).

Constructing the Tables.

The table used in an instance of WhiteKey is built in the same way as for WhiteBlock. The only difference is that the output of $\mathcal{A}$ is not truncated and the full 128 bits are used.

Design Rationale of WhiteKey

The first part of the scheme is to select part of the key, which is a mandatory feature. The fastest way to do so is to parallelize memory accesses, and they are therefore generated independently.

The second part is to derive a short key from the table values, which are of high min-entropy. The standard way to build a key derivation function is to use a hash function [Kra10]. However it is slow, since even a fast hash function like BLAKE2b takes 3 cycles per byte on modern processors [ANWOW13]. Instead, we decided to use an extractor, which has also the advandage to be unconditionally secure for a uniform seed. The extractor literature focused primarily on reducing the number of seed bits and maximizing the number of extracted bits, because of their importance in theoretical computer science; see [Sha11] for a survey. In our case, we want to extract only a few bits and speed is the principal concern. The approach recommended by [BDK+11] is to generate pseudo-random elements in a large field using a standard pseudorandom generator

(say, AES-CTR) and to compute a dot product with the input. The main problem of this extractor is that it uses a seed which is very large, and it takes about as much time to generate it (with AES-NI) as to use it. Hence, we decided to use the extractor introduced in [CMNT11], which has a seed length around the square root of the length of the input. Since we can evaluate $\sum_{i,j} Q_{i,j} a_i b_j$ with about one multiplication and one addition in the field per input value, the computation of the extractor takes essentially the same time. Indeed, the complexity of the extractor is similar to GHASH.

Another possibility for the extractor is to increase the degree, for instance use $\sum_{i,j,k} Q_{i,j,k} a_i b_j c_k$. While this approach, proposed by [CNT12], is indeed sound and allows to reduce the seed further, the best bound we know on the statistical distance of the output is about $q^{-1/2}$ when working over $\mathbb{F}_q$. The main problem is that the tensor decomposition of $Q_{i,j,k}$ does not have the needed properties, so that Coron et al. use a generic bound on the number of zeroes, which must account for elliptic curves and therefore a deviation of $q^{-1/2}$ is required. The specific case of $\sum_{k=0}^{1} \sum_{i,j} Q_{i,j,k} a_i b_j c_k$ can probably be tackled using linear matrix pencil theory, at a cost of a much more difficult proof.

Concrete Parameters for Various Instances of WhiteKey.

Once the size of an instance of WhiteKey has been chosen (*i.e.* the output size of the table $\mathcal{T}$), the only parameter that needs to be determined is the number of calls to the tables n, and thus the number of output blocks t of $\mathcal{A}$. This is obtained by instantiating the formula of Thm. 3 for a given white-box security (note that unlike WhiteBlock, the maximum security that is possible to reach for a table of x-bit inputs is $128 - x$). We give the parameters for instances of various sizes in Tbl. 3.2. The tables used in these instances have the same input size as the ones of the WhiteBlock instances of Tbl. 3.1, but they are twice as large because of their larger output size, which impacts the size of a white-box implementation similarly. On the other hand, a single table is used in WhiteKey, whereas up to four (for input sizes of 16 bits and more) are necessary in WhiteBlock.

Instance	WB size	# Table/block	WB security	#Table calls (#blocks)
WhiteKey 16	2^{20} B	8	112 bits @ 1/4	57 (8)
WhiteKey 20	2^{24} B	6	108 bits @ 1/4	55 (10)
WhiteKey 24	2^{28} B	5	104 bits @ 1/4	53 (11)
WhiteKey 28	2^{32} B	4	100 bits @ 1/4	51 (13)
WhiteKey 32	2^{36} B	4	96 bits @ 1/4	49 (13)

Table 3.2: Number of table calls for WhiteKey instances with tables of selected input sizes from 16 to 32 bits, at a white-box security level of 96 to 112 bits for a compression factor of 4. Black-box security is 128 bits in all cases.

CoureurDesBois: WhiteKey in Practice.

Similarly to WhiteBlock and PuppyCipher, we define the CoureurDesBois family as a concrete instantiation of WhiteKey. It simply consists in using AES128 for $\mathcal{A}$ and a specific representation for $\mathbb{F}_{2^{128}}$, *e.g.* $\mathbb{F}_2[x]/x^{128} + x^7 + x^2 + x + 1$ (the "GCM" field).

Unlike PuppyCipher, the components of CoureurDesBois are not cascaded multiple times; hence we cannot hope for a similar tradeoff of provable security against speed. However, the main advantage of CoureurDesBois compared to PuppyCipher is that it lends itself extremely well to parallelization. This allows to optimally hide the latency of the executions of AES and of the queries to the table in memory.

We further discuss the matter in Section 3.5, with implementations of CoureurDesBois with AES instructions for tables of 16 to 24-bit inputs.

3.4 Security Proofs

For both the WhiteBlock and WhiteKey constructions, we provide proofs in the weak incompressibility model. These proofs provide concrete bounds, on which we base our implementations. This allows direct comparison to previous work [BBK14, BI15]. Moreover in the case of WhiteKey, we provide a proof in the strong incompressibility model. This proof shows the soundess of the general construction in a very demanding model. However we do not use it to fix the parameters of our constructions.

Recall that weak incompressibility (Definition 13) depends on three parameters (s, λ, δ): essentially if the number of outputs of the table known to the adversary is s, then (s, λ, δ)-incompressibility says that with probability at least $1 - 2^{-\lambda}$, the adversary is unable to encrypt more than a ratio δ of plaintexts, no matter which s table outputs she chooses to learn. If inputs to the table are t-bit long, then $\alpha = s2^{-t}$ is the fraction of the table known to the adversary. We can fix $\alpha = 1/4$ as in [BI15], hence $s = \alpha 2^t$. In that case weak incompressibility essentially matches $(s, -\log(\delta))$-space hardness from [BI15], and $-\log(\delta)$ can be thought of as the number of bits of white-box security.

However we do not claim security for $\delta = 2^{-128}$, which would express 128 bits of white-box security. Instead, we claim security for $\delta = 2^{-128+t}$. Thus for larger table of size $\approx 2^{28}$, white-box security drops to around 2^{100}. We believe this is quite acceptable.

The reason we claim only $128 - t$ bits of white-box security rather than 128 is a result of our security proofs, as we shall see. This should be compared with the fact that an adversary allowed to store s table inputs could use the same space to store s outputs of the whole scheme (within a small constant factor λ/t due to the size difference between outputs of the table and outputs of the scheme). Such an adversary would naturally be able to encrypt a proportion $s2^{-\lambda}$ of inputs. Since $s = 2^t/4$, with a small constant factor $1/4$, this yields the $128 - t$ bits of white-box security achieved by our proofs.

Our security claims are summarized in tables 3.3.1 and 3.3.2. We provide proofs of both WhiteKey and WhiteBlock in the weak incompressibility model. In the case of WhiteKey, a proof is also available in the strong incompressibility model. We begin with the proof of WhiteKey for weak incompressibility, which is fairly straightforward, yields better bounds (as one would expect), and also serves as a warm-up for the combinatorially more involved proof of WhiteBlock.

3.4.1 Weak Incompressibility of WhiteKey

Overview.

The initial layer of WhiteKey is comprised of a PRF generating the inputs of the table calls. Using standard arguments, this pseudo-random function can be replaced by a random function. The effect this has on the weak incompressibility adversary is upper-bounded by the distinguishing

advantage of a real-or-random adversary against the PRF. Thus we are essentially free to treat the initial PRF as a random function.

In the weak incompressibility game, the adversary learns the output of the table on some adaptatively chosen inputs. By nature of white-box security, any keying material present in the PRF is known to the adversary (formally, in our definition of white-box encryption scheme this keying material would have to be appended to the table T of the white-box implementation, and could be recovered with a single or few queries). Hence the adversary can choose which table inputs she queries based on full knowledge of the initial PRF.

On the other hand, for a given PRF input, as soon as the adversary does not know a single output of the table, due to the linearity of the final layer of the construction, the output has full 128-bit entropy from the point of view of the adversary.

Thus the core of the proof, is to show that, with high probability over the random choice of the PRF, for the best possible choice of s table inputs the adversary chooses to query[26], most PRF outputs still include at least one table input that is unknown to the adversary. We explicitely compute this upper bound in the next section.

More precisely, Theorem 2 will show:

$$\log\left(\Pr\left[\mu(s) \geq k\right]\right) \leq 2^t - k\log\left(\frac{k}{\rho}\right) - (n-k)\log\left(\frac{n-k}{n-\rho}\right) \tag{3.1}$$

where:

$n = 2^\lambda$	is the size of the input space of WhiteKey;
t	is the number of bits at the input of a table;
s	is the number of table entries stored by the adversary;
ρ	$= 2^\lambda (s/2^t)^m$, with m the number of table calls in the construction;
k	is the maximal number of inputs the adversary may be able to encrypt;

and $\mu(s)$ is the maximal number of WhiteKey inputs that can be encrypted with storage size s; it is a random variable over the uniform choice of the initial PRF ($\mathcal{A}$ in counter mode, in the previous description).

We want this bound to be below $-\lambda$. We are now interested in what this implies, in terms of number of table calls m necessary to achieve a given security level. As noted earlier, the bound imposes $k \approx 2^t$. For simplicity we let $k = 2^t$, which means we achieve $\lambda - t$ bits of white-box security (*i.e.* $\delta = 2^{t-\lambda}$ in the sense of Definition 13). We can also fix $s/2^t = 1/4$ for the purpose of being comparable to [BI15].

The term $(n-k)\ln\left((n-k)/(n-\rho)\right)$ is equivalent to $\rho - k$ as k/n tends to zero[27]. Since we are looking for an upper bound we can approximate it by k. This yields a probability:

$$\begin{aligned} 2^t\left(1 - k2^{-t}\left(\log\left(\frac{k}{\rho}\right) - 1\right)\right) &= 2^t\left(1 - k2^{-t}\left(\log(k) - \lambda + 2m - 1\right)\right) \\ &= -2^t\left(\log(k) - \lambda + 2m\right) \end{aligned}$$

In the end, we get that m only needs to be slightly larger than $\frac{\lambda - \log(k)}{2}$. Indeed, as long as this is the case, the 2^t factor will ensure that the bound is (much) lower than -128.

This actually matches a generic attack. If the adversary just stores $s = 2^t/4$ random outputs of the table, then on average she is able to encrypt a ratio 2^{-2m} of inputs. This imposes $2^{-2m} < k2^{-\lambda}$, so $m > (\lambda - \log(k))/2$. When testing our parameter choices against Eq. 3.1,

[26] In this respect, the adversary we consider is computationally unbounded.

[27] In fact, simple functional analysis shows that we can bound the right-hand term by $4(\rho - k)$ provided $\alpha^m < 1/2$ and $k < 4n$, which will always be the case.

we find that it is enough to add a single table call beyond what the generic attack requires: in essence, Theorem 2 implies that no strategy is significantly better than random choices.

Proof of Weak Incompressibility for WhiteKey

The following assumptions sum up our proof model.

1. The table T is modeled as being chosen uniformly at random among functions from t bits into λ bits. It is fully known to the adversary.

2. The PRG F used in the initial part of the construction is also modelled as being chosen uniformly at random among functions from λ bits into $m \cdot t$ bits. It is also fully known to the adversary.

3. The adversary attempts to compress the construction by storing the full outputs of some (adversarially chosen) subset S of the entries of the table.

Note that our proof is within the information theoretical setting: in other words, the adversary is allowed unlimited computational power. We formalize the previous assumptions as follows. We use the same notation as in the previous section, and in addition:

Notation.

$$\begin{aligned} F &\quad \text{is a function from } \lambda \text{ bits into } tm \text{ bits (the initial PRG).} \\ \alpha \triangleq s2^{-t} &\quad \text{is the proportion of the table stored by the adversary.} \end{aligned}$$

For $x \in \{0,1\}^{m \cdot t}$ and $i < m$, we split x into m t-bit substrings and define $x^{(i,t)}$ as the i-th t-bit substring of x, i.e. $x^{(i,t)} = x_{ti} \dots x_{t(i+1)-1}$. In the remainder t is fixed and always denotes the number of bits at the input of T, so we will simply write $x^{(i)}$.

For $F : \{0,1\}^\lambda \to \{0,1\}^{tm}$ and $S \subseteq \{0,1\}^t$, we say that $x \in \{0,1\}^\lambda$ is *bad* iff the adversary has stored all table inputs appearing at the output of $F(x)$. More formally, we define the set of bad x's as:

$$B_S^F \triangleq \{x \in \{0,1\}^\lambda : \forall i < m, F(x)^{(i)} \in S\}$$

If F is chosen uniformly at random, B_S^F becomes a random variable, which we will simply write as B_S.

For a given F, the adversary in our model wants to find S so as to maximize $|B_S^F|$. Thus we are interested in bounding, whp over the random choice of F, the random variable:

$$\mu(s) = \max_{|S|=s} |B_S|$$

Note that if we fix $|S| = s$, although the $|B_S|$'s are not independent, they follow the same distribution. Moreover, because the outputs of F are uniformly random and independent, $\mathbb{E}(|B_S|) = 2^\lambda (s/2^t)^m = \rho$.

Theorem 2. *Let F be chosen uniformly at random among functions $\{0,1\}^\lambda \to \{0,1\}^{tm}$. Then for all $k > \rho$, we have:*

$$\log\left(\Pr\left[\mu(s) \geq k\right]\right) \leq 2^t - k \log\left(\frac{k}{\rho}\right) - (n-k)\log\left(\frac{n-k}{n-\rho}\right) \tag{3.2}$$

Proof. In the remainder we consider probabilities over the random choice of F. Since F is a random function, each output is uniformly random and independent. It follows that each output of F has probability α^m of having its m t-bit substrings within S. As a result $|B_S|$ follows a simple binomial distribution:

$$\Pr\left[|B_S| = k\right] = \binom{n}{k}\alpha^{km}(1-\alpha^m)^{n-k}$$

On the other hand we have:

$$\begin{aligned}\Pr\left[\mu(s) \geq k\right] &= \Pr\left[\bigcup_{|S|=s}\left[|B_S| \geq k\right]\right] \\ &\leq \sum_{|S|=s}\Pr\left[|B_S| \geq k\right] \\ &= \binom{2^t}{s}\Pr\left[|B_S| \geq k\right]\end{aligned}$$

Indeed, for any choice of S of fixed size s, the probability of the event $|B_S| \geq k$ is the same (recall that the probability is taken over the uniformly random choice of F).

The terms $|B_S| \geq k$ above are the tails of a binomial distribution, namely that of $|B_S|$. As such they can be bounded using e.g. Hoeffding's or Chernoff's inequalities. In our case α^m will typically be very small, and we use the following form of Chernoff's bound:

$$\Pr\left[|B_S| \geq k\right] \leq \exp\left(-n \cdot D\left(\frac{k}{n}||\alpha^m\right)\right)$$

where $D(a||b)$ denotes the relative entropy of Bernouilli trials with parameters a and b:

$$D(a||b) \triangleq a \cdot \ln\frac{a}{b} + (1-a)\ln\frac{1-a}{1-b}$$

This yields:

$$\begin{aligned}\Pr\left[|B_S| \geq k\right] &\leq \exp\left(-k\ln\left(\frac{k}{n\alpha^m}\right) - (n-k)\ln\left(\frac{1-\frac{k}{n}}{1-\alpha^m}\right)\right) \\ &= \exp\left(-k\ln\left(\frac{k}{\rho}\right) - (n-k)\ln\left(\frac{n-k}{n-\rho}\right)\right)\end{aligned}$$

Finally, bounding $\binom{2^t}{s}$ by 2^{2^t}, we get the result.

It is worth noting that the term on the left-hand side is essentially the tail bound for a Poisson distribution (recall that the binomial distribution converges towards the Poisson distribution for bounded ρ as $n \to \infty$). □

3.4.2 Weak Incompressibility of WhiteBlock.

Overview.

The general approach of the proof is the same as above. However the combinatorial arguments are much trickier, essentially because table calls are no longer independent (they depend on table outputs in the previous round.). Nevertheless an explicit bound is proven in the next section.

However, what we prove is only that whp, for most inputs to WhiteBlock, during the computation of the output, at least two table calls at different rounds are unknown to the adversary. Since table outputs cover half a block, this implies at two separate rounds during the course of the computation, 64 bits are unknown and uniform from the point of view of the adversary. At this point we heuristically assume that for an efficient adversary, this implies the output cannot be computed with probability significantly higher then 2^{-128}. In practice the bottleneck in the bound provided by the proof comes from other phenomena, namely we prove $128 - t$ bits of security for t-bit tables. Nevertheless this means our proof is heuristic.

More precisely, Theorem 3 shows:

$$\log\left(\Pr\left[\mu(s) \geq k\right]\right) \leq 2^t + k\left(\lambda + m\Big(1 - \frac{1}{k} - \frac{1}{r}\Big)\log\left(\frac{s}{2^t}\right)\right)$$

where:

- λ is the input size of WhiteBlock;
- t is the number of bits at the input of a table;
- r is the number of rounds;
- m is the total number of table calls in the construction ($m \triangleq \lfloor(\lambda/2)/t\rfloor \cdot r$);
- s is the number of table entries stored by the adversary;
- k is the maximal number of inputs the adversary may be able to encrypt;

and $\mu(s)$ is the maximal number of WhiteBlock inputs that can be encrypted with storage size s; it is a random variable over the uniform choice of the round permutations $\mathcal{A}_{k_i}$.

We are now interested in what this bound implies, in terms of number of rounds r to achieve a given security level. Observe that the bound requires $k \approx 2^t$. For simplicity we let $k = 2^t$, which means we achieve $\lambda - t$ bits of white-box security (*i.e.* $\delta = 2^{t-\lambda}$ in the sense of Definition 13). We can also fix $s/2^t = 1/4$ for the purpose of being comparable to [BI15]. Observe that $1/k$ is negligible compared to $1/r$. Let $c = \lfloor(\lambda/2)/t\rfloor$ be the number of table calls per round. Then our bound asks:

$$\lambda - 2m\left(1 - \frac{1}{r}\right) = \lambda - 2c(r-1) < 0$$

Indeed, as long as this value is negative, the preceding $k = 2^t$ factor will ensure that the bound is (much) lower than -128. We get:

$$r > \frac{\lambda}{2c} + 1$$

We can compare this bound with the previous generic attack, where the adversary stores table outputs at random. As we have seen, this attack implies $m > (\lambda - \log(k))/2$, so $r > (\lambda - \log(k))/(2c)$. Instead our proof requires $r > \frac{\lambda}{2c} + 1$. Thus the extra number of rounds required by our security proof, compared to the lower bound coming from the generic attack, is less than $\log(k)/(2c) + 1$: it is only a few extra rounds (and not, for instance, a multiplicative factor).

Proof of Weak Incompressibility for WhiteBlock.

The outline of the proof is the same as WhiteKey, but the reasoning is more intricate because table inputs are no longer independent. The goal of the proof is summed up as follows.

1. The table T is modeled as being chosen uniformly at random among functions from t bits into $\lambda/2$ bits. It is fully known to the adversary.

2. The PRP's $P_1, \ldots, P_r$ are modelled as being chosen independently and uniformly at random among permutations on λ bits. They are also fully known to the adversary.

3. The adversary attempts to compress the construction by storing the full outputs of some (adversarially chosen) subset S of the entries of the table.

Note that our proof is within the information theoretical setting: in other words, the adversary is allowed unlimited computational power. We formalize the previous assumptions as follows. We reuse the previous notation, and in addition:

Notation.

$c \triangleq \lfloor (\lambda/2)/t \rfloor$	is the number of table calls per round.
$P_1, \ldots, P_{r-1}$	are permutations on λ bits (the PRP's).
$n \triangleq 2^\lambda$	is the number of entries of the P_i's.
$\alpha \triangleq s2^{-t}$	is the proportion of the table stored by the adversary.

For $x \in \{0,1\}^\lambda$ and $i < c$, write $x^{(i,t)}$ for the i-th t-bit substrings of x, i.e. $x^{(i,t)} = x_{ti} \ldots x_{t(i+1)-1}$. As before t is fixed and always denotes the number of bits at the input of the table T, so we will omit it and write $x^{(i)}$.

Let Q_i denote the i-th round function of the construction. That is, Q_i is the composition of one round of c table calls followed by P_i. Since P_i is uniformly random, so is Q_i. Let $R_i \triangleq Q_i \circ \cdots \circ Q_1$ denote the first i rounds of the construction. Note that the R_i's still form a sequence of uniformly random and independent permutations. Let $\vec{R} = (R_1, \ldots, R_{r-1})$ denote the sequence of the R_i's.

If $x \in \{0,1\}^\lambda$ is the input of the cipher, then $R_i(x)$ is the state of the cipher after i rounds, with the convention that R_0 is the identity. As a consequence the $R_i(x)^{(j)}$'s for $0 \leq i < r, 0 \leq j < c$ are the inputs of the table calls during the execution of the cipher on input x.

Note that R_0 plays a special role as it is the identity, whereas the other R_i's are uniformly random. Before proceeding it is convenient to "symmetrize" the problem by composing every permutation R_i with the same uniformly random permutation. This does not change the value of any random variable we are interested in since it just amounts to permuting the input of the construction. But this allows all R_i's to be uniformly random, avoiding unnecessary special treatment for R_0.

As earlier we denote by S the set of inputs of the table T that is stored by the adversary. In our construction the output of the table has size $\lambda/2$ bits. Hence in order to force the adversary to guess λ bits to compute the output of the cipher on input x, we want that $R_i(x)^j$ falls outside of S for at least two distinct rounds i and i' (if this happens on the same round, note that the adversary still only has to guess $\lambda/2$ bits).

Thus, given $\vec{R}$ and S, we shall say that an input x is *bad* if for all but at most one round, the adversary knows all $R_i(x)^{(j)}$'s. Formally:

$$x \text{ is bad iff } |\{0 \leq i < r : \forall j < c, R_i^{(j)} \in S\}| \geq r - 1$$

Let $B_S^{\vec{R}}$ denote the set of bad x's. If $\vec{R}$ is chosen uniformly at random, $B_S^{\vec{R}}$ becomes a random variable, which we will simply write as B_S.

For a given $\vec{R}$, the adversary in our model wants to find S so as to maximize $|B_S^{\vec{R}}|$. Thus we are interested in bounding, whp over the random choice of $\vec{R}$, the random variable:

$$\mu(s) = \max_{|S|=s} |B_S|$$

Theorem 3. *Let $\vec{R}$ be chosen uniformly at random among sequences of r permutations on $\{0,1\}^\lambda$. Assume $r < n/e$ and $s \leq (n-k)/2$. Then with the previous notation we have:*

$$\log\left(\Pr\left[\mu(s) \geq k\right]\right) \leq 2^t + k\left(\lambda + m\Big(1 - \frac{1}{k} - \frac{1}{r}\Big)\log\left(\frac{s}{2^t}\right)\right)$$

Proof. As in the previous case, we begin by bounding $\mu(s)$ as follows:

$$\begin{aligned}\Pr\left[\mu(s) \geq k\right] &= \Pr\left[\bigcup_{|S|=s}\left[|B_S| \geq k\right]\right] \\ &\leq \sum_{|S|=s}\Pr\left[|B_S| \geq k\right] \\ &= \binom{2^t}{s}\Pr\left[|B_S| \geq k\right]\end{aligned}$$

for any fixed S satisfying $|S| = s$. The difficulty lies in the fact that $|B_S|$ does not quite follow a binomial distribution anymore. Our goal from now on is to upper bound the tail distribution of $|B_S|$ whp, i.e. upper bound $\Pr\left[|B_S| \geq k\right]$.

Let $X \subseteq \{0,1\}^\lambda$ denote the set of states for which the adversary knows all table calls, that is:

$$X \triangleq \{x \in \{0,1\}^\lambda : \forall i < c, x^{(i)} \in S\}$$

Thus x is bad iff for all but at most one $i < r$, $R_i(x) \in X$. It is worth noting in passing that the only aspect of the R_i's that actually matters is whether $R_i(x)$ lies in X, rather that its exact value. Let $s' \triangleq s^c 2^{\lambda - tc}$ so that $|X| = s'$.

Fix $k \in \mathbb{N}$ and $S \subseteq \{0,1\}^t$ with $|S| = s$. Since $\vec{R}$ is uniform, computing the probability of $|B_S| = k$ is the same as counting the number of $\vec{R}$'s for which $|B_S^{\vec{R}}| = k$.

Let us consider the event where a fixed subset $A \subseteq \{0,1\}^\lambda$ of size $|A| = k$ contains only bad x's, i.e. define the event:

$$E_{A,S} = [A \subseteq B_S]$$

The point is that we have:

$$\begin{aligned}\left[|B_S| \geq k\right] &\subseteq \bigcup_{|A|=k}\left[A \subseteq B_S\right] \\ \Pr\left[|B_S| \geq k\right] &\leq \sum_{|A|=k}\Pr\left[E_{A,S}\right]\end{aligned}$$

so we are now interested in upper bounding $\Pr\left[E_{A,S}\right]$.

The event $E_{A,S}$ tells us that for all $a \in A$, all $R_i(a)$'s are in X except at most one. We now partition this event by considering which $R_i(a)$'s are in X. Formally, we consider the set of functions $\mathcal{I} : A \to \{0,\ldots,r\} \cup \{\perp\}$, and for $I \in \mathcal{I}$ we define the event:

$$E'_{I,A,S} \triangleq [\forall a \in A, \forall 0 \leq i < r, (R_i(a) \notin X) \Leftrightarrow (I(a) = i)]$$

Thus, we have:

$$\begin{aligned}E_{A,S} &= \bigcup_{I \in \mathcal{I}} E'_{I,A,S} \\ \Pr\left[E_{A,S}\right] &\leq \sum_{I \in \mathcal{I}}\Pr\left[E'_{I,A,S}\right]\end{aligned}$$

Lemma 10. *Using the previous notation, for all choices of I, A, S, if $2s' \leq n-k$ then:*

$$\Pr\left[E'_{I,A,S}\right] \leq \left(\binom{n-k}{s'-(k-\lceil k/r \rceil)}\binom{n}{s'}^{-1}\right)^r$$

Proof. Fix I, A and S. For $0 \leq i < r$, let $b(i) \triangleq |I^{-1}(i)|$. The event $E'_{I,A,S}$ occurs iff for all $0 \leq i < r$:

$$\{a \in A : R_i(a) \notin X\} = I^{-1}(i)$$

Thus $E'_{I,A,S}$ imposes independent constraints on each of the R_i's, which are sampled independently. It follows that:

$$\Pr\left[E'_{I,A,S}\right] = \prod_{0 \leq i < r} \Pr\left[\{a \in A : R_i(a) \notin X\} = I^{-1}(i)\right]$$

Fact 2. *Fix X, A, $i < r$ and $I \in \mathcal{I}$ as above. The following holds.*

$$\Pr\left[\{a \in A : R_i(a) \notin X\} = I^{-1}(i)\right] = \binom{n-k}{s'-k+b(i)}\binom{n}{s'}^{-1}$$

Proof. The event $E = [\{a \in A : R_i(a) \notin X\} = I^{-1}(i)]$ may be rewritten as $[R^{-1}(X) \cap A = I^{-1}(i)]$. As such the event is entirely determined by the value of $R^{-1}(X)$. Because R_i is uniformly random, all $\binom{n}{s'}$ choices of $R^{-1}(X)$ are equiprobable. Thus computing the probability of E amounts to a simple counting problem, namely how many of the $\binom{n}{s'}$ choices of $R^{-1}(X)$ satisfy $R^{-1}(X) \cap A = I^{-1}(i)$. In fact there are exactly $\binom{n-k}{s'-(k-b(i))}$ such choices. Indeed, there are $\binom{n-k}{s'-(k-b(i))}$ choices for $R^{-1}(X) \cap (\{0,1\}^\lambda \setminus A)$. □

We can now deduce:

$$\Pr\left[E'_{I,A,S}\right] = \prod_{0 \leq i < r} \left(\binom{n-k}{s'-k+b(i)}\binom{n}{s'}^{-1}\right) \tag{3.3}$$

Since $\bigcup I^{-1}(i) \subseteq A$, the $b(i)$'s satisfy:

$$\sum_{0 \leq i < r} b(i) \leq k \tag{3.4}$$

with equality iff $I^{-1}(\perp)$ is empty. We wish to upper bound $\Pr[E_{I,A,S}]$. For this purpose we set out to upper bound the product (3.3) subject to inequality (3.4).

To this end we begin by eliminating cases where (3.4) is not an equality, i.e. $I^{-1}(\perp) \neq \varnothing$. This is taken care of by the following fact.

Fact 3. *Assume $2s \leq n-k$. Choose $I \in \mathcal{I}$ such that $I^{-1}(\perp) \neq \varnothing$. Then there exists $I' \in \mathcal{I}$ such that $E_{I',A,S}$ is more probable than $E_{I,A,S}$. Hence I does not maximize the probability of $E_{I,A,S}$.*

Proof. Let I be such that $I^{-1}(\perp) \neq \varnothing$. Note that necessarily $b(0) < k$. Pick $i \in I^{-1}(\perp)$. Take I' such that $I'(i) = 0$ and $I'(j) = I(j)$ for $j \neq i$. Then by (3.3) we have:

$$\begin{aligned}\frac{\Pr\left[E'_{I',A,S}\right]}{\Pr\left[E'_{I,A,S}\right]} &= \binom{n-k}{s'-k+b(0)+1}\binom{n-k}{s'-k+b(0)}^{-1} \\ &= \frac{n-s'-b(0)}{s'-k+b(0)+1} \\ &\geq \frac{n-s'-k}{s} \qquad \text{using } b(0) < k \\ &\geq 1 \qquad \text{using the assumption } 2s' \leq n-k\end{aligned}$$

□

Thus in order to maximize $\Pr[E_{I,A,S}]$, we are free to only consider the case where (3.4) is an equality. The following fact completes the lemma.

Fact 4. *Pick integers r and $k < s < n$. Consider vectors $\vec{b} = (b(i))_{0\leq i\leq r}$ in $\mathbb{N}$ such that $\sum b(i) = k$. Then the quantity:*

$$f(\vec{b}) \triangleq \prod_{0\leq i<r}\left(\binom{n-k}{s'-k+b(i)}\binom{n}{s'}^{-1}\right)$$

is maximal when $\lfloor k/r\rfloor \leq b(i) \leq \lceil k/r\rceil$ for all i.

Proof. In essence Fact 4 can be viewed as a convexity argument on the function $x \mapsto \log(\binom{n-k}{s'-k+x})$. However because we are dealing with integers it will be less cumbersome to prove it directly.

Assume that the statement $\forall i, \lfloor k/r\rfloor \leq b(i) \leq \lceil k/r\rceil$ is false. We are going to show that $f(\vec{b})$ is not maximal. Because $\forall i, \lfloor k/r\rfloor \leq b(i) \leq \lceil k/r\rceil$ is false, there exist $i \neq j$ such that $b(i)+2 \leq b(j)$. Define $\vec{b'}$ by $b'(i) = b(i)+1$, $b'(j) = b(j)-1$ and $b'(l) = b(l)$ for $l \neq i, j$. We claim that $f(\vec{b'}) > f(\vec{b})$. To see this, letting $n' = n-k$, $s'' = s'-k$, compute:

$$\begin{aligned}\frac{f(\vec{b'})}{f(\vec{b})} &= \binom{n'}{s''+b(i)+1}\binom{n'}{s''+b(i)}^{-1}\binom{n'}{s''+b(j)-1}\binom{n'}{s''+b(j)}^{-1} \\ &= \frac{n'-b(i)}{s''+b(i)+1}\cdot\frac{s''+b(j)}{n'-b(j)-1} \\ &= \frac{n'-b(i)}{n'-b(j)-1}\cdot\frac{s''+b(j)}{s''+b(i)+1} \\ &> 1 \qquad \text{using } b(i)+1 < b(j)\end{aligned}$$

Thus if $f(\vec{b})$ is maximal, then $\forall i, \lfloor k/r\rfloor \leq b(i) \leq \lceil k/r\rceil$. Conversely, if $\forall i, \lfloor k/r\rfloor \leq b(i) \leq \lceil k/r\rceil$, because $\sum b(i) = k$, the values of the $b(i)$'s are fixed up to permutation. In particular $f(\vec{b})$ is entirely determined, and necessarily maximal. □

Combining Facts 3 and 4 with Eq. (3.3) yields the lemma. □

Corollary 1. *With the previous notation, as long as $2s' \leq n-k$, it holds that:*

$$\log\left(\Pr\left[E'_{I,A,S}\right]\right) \leq r\left(k + (k-\lceil k/r\rceil)\log\left(\frac{s'}{n}\right)\right)$$

Proof. Letting $k' \triangleq k - \lceil k/r \rceil$, we have:

$$\begin{aligned}\binom{n-k}{s'-(k-\lceil k/r\rceil)}\binom{n}{s'}^{-1} &= \frac{s'!}{(s'-k')!}\prod_{i=0}^{k'-1}\frac{1}{n-i}\prod_{i=k'}^{s'}\frac{n-i-\lceil k/r\rceil}{n-i}\\ &\leq \prod_{i=0}^{k'-1}\frac{s'-i}{n-i}\\ &\leq \left(\frac{s'}{n}\right)^{k'}\end{aligned}$$

Plugging this inequality into Lemma 10 yields:

$$\Pr\left[E'_{I,A,S}\right] \leq \left(\frac{s'}{n}\right)^{k'r}$$ □

Tracing our way back through the previous computations and letting $k' = k - \lceil k/r \rceil$ as before, we have:

$$\log\left(\Pr\left[E'_{I,A,S}\right]\right) \leq r\left(k - \lceil k/r \rceil\right)\log\left(\frac{s'}{n}\right)$$

Using $|\mathcal{I}| = r^k$ we deduce:

$$\log\left(\Pr\left[E_{A,S}\right]\right) \leq k\log(r) + rk'\log\left(\frac{s'}{n}\right)$$

$$\log\left(\Pr\left[|B_S| \geq k\right]\right) \leq \log\binom{n}{k} + k\log(r) + rk'\log\left(\frac{s'}{n}\right)$$

Finally, upper bounding $\binom{2^t}{s'}$ by 2^{2^t} we get:

$$\log\left(\Pr\left[\mu(s) \geq k\right]\right) \leq 2^t + \log\binom{n}{k} + k\log(r) + rk'\log\left(\frac{s'}{n}\right)$$

We could immediately upper bound $\log\binom{n}{k}$ by $k\log(n) = k\lambda$ but we would like to get rid of the term $k\log(r)$ above, which is clearly negligible. To get the desired result, we show that $\log\binom{n}{k} + k\log(r) \leq k\lambda$:

$$\begin{aligned}\log\binom{n}{k} &\leq k\log(n) - \log(k!)\\ &\leq k\lambda - k\log\left(\frac{k}{e}\right) \qquad \text{using } k! > \left(\frac{k}{e}\right)^n\\ \log\binom{n}{k} + k\log(r) &\leq k\lambda + k\left(\log(r) - \log\left(\frac{n}{e}\right)\right)\\ &\leq k\lambda\end{aligned}$$

In the last line, we make use of the assumption $r < n/e$. Note that in practice n is exponentially larger than r. In the end, we get:

$$\log\left(\Pr\left[\mu(s) \geq k\right]\right) \leq 2^t + k\lambda + rk'\log\left(\frac{s'}{n}\right)$$

Lower bounding $k' = k - \lceil k/r \rceil$ by $k - k/r - 1$ yields the result. □

3.4.3 Strong Incompressibility of WhiteKey

We first prove that $\sum_{i,j} Q_{i,j} a_i b_j \in \mathbb{F}_q$ is a strong extractor. This extractor comes mostly from [CMNT11, Section 4.2] but we tighten the proof.

Definition 18. *A family $\mathcal{H}$ of hash functions $h : X \mapsto Y$ is ϵ-pairwise independent if*

$$\sum_{x \neq x'} \left(\Pr_{h \leftarrow \mathcal{H}}[h(x) = h(x')] - \frac{1}{Y} \right) \leq \frac{\epsilon |X|^2}{Y}.$$

The next lemma is a variant of the leftover hash lemma, proven in [Sti02, Theorem 8.1].

Lemma 11. *Let $h \in \mathcal{H}$ be uniformly sampled, and $x \in X$ be an independent random variable with min-entropy at least k. Then, the statistical distance between $(h(x), h)$ and the uniform distribution is at most*

$$\sqrt{|Y| 2^{-k} + \epsilon}.$$

We now prove that our function is indeed pairwise independent.

Lemma 12. *Let $\mathcal{H} = \mathbb{F}_q^{2n}$, $X = M_n(\mathbb{F}_q)$ and $Y = \mathbb{F}_q$. Then, the function $h_{a,b}(Q) = \sum_{i,j} Q_{i,j} a_i b_j = a^t Q b$ is $11q^{-n}$-pairwise independent.*

Proof. We first count the number of a, b such that $\sum_{i,j} Q_{i,j} a_i b_j = a^t Q b = 0$. Let Q be a matrix of rank r. Then, there exist r vectors u, v such that $Q = \sum_{k=0}^{r-1} u_i v_i^t$ and the u_i as well as the v_i are linearly independent. Thus,

$$a^t Q b = \sum_{k=0}^{r-1} a^t u_i v_i^t b$$

and therefore, by a change of basis, this form has the same number of zeros as

$$\sum_{k=0}^{r-1} a_i b_i$$

which is $q^{2n-1} + q^{2n-r} - q^{2n-r-1}$.

Now, there are $\prod_{k=1}^{r-1} \frac{(q^n - q^k)^2}{q^r - q^k}$ matrices of rank r. We deduce :

$$\begin{aligned}
\sum_{x \neq x'} \left(\Pr_{h \leftarrow \mathcal{H}}[h(x) = h(x')] - \frac{1}{Y} \right) &= \sum_{r=1}^{n} \left((q^{-r} - q^{-r-1}) q^{-n^2} \prod_{k=0}^{r-1} \frac{(q^n - q^k)^2}{q^r - q^k} \right) \\
&\leq \sum_{r=1}^{n} q^{-r} q^{-n^2} q^{2nr - r^2} \prod_{k=1}^{\infty} \frac{1}{1 - 1/q^k} \\
&\leq \frac{2 - 1/q}{1 - 1/q} q^{-n} \prod_{k=1}^{\infty} \frac{1}{1 - 1/q^k} \\
&\leq 11 q^{-n}
\end{aligned}$$

□

Hence, if the input of our extractor has at least 2μ bits of entropy, the generated key will be essentially uniform. The proof for the security of sampling the seed from a pseudorandom generator (from which we cannot build a public-key primitive) is in [BDK+11]. We now prove that the input has indeed a lot of entropy.

Lemma 13. *Let $f : [n] \mapsto [0;1]$ be of average μ. Then, the average of the image k uniform elements is at least $\mu - \delta$, except with probability*

$$\exp(-\frac{k^2\delta^2/2}{k/4 + \delta\mu/3}).$$

Proof. This is the result of Bernstein's inequality (see [BLB04, Theorem 3]), since the variance of all terms is at most 1/4 and they are all positive. □

We now use a lemma of Vadhan [Vad04, Lemma 9] :

Lemma 14. *Let S be a random variable over $[n]^t$ with distinct coordinates and $\mu, \delta, \epsilon > 0$, such that for any function $f : [n] \mapsto [0;1]$ of average $(\delta - 2\tau)/\log(1/\tau)$, we have that the probability that the average of the image of the t positions given by S is smaller than $(\delta - 3\tau)/\log(1/\tau)$ is at most ϵ.*

Then, for every X of min-entropy δn over $\{0,1\}^n$, the variable (S, X_S) where X_S is the subset of bits given by S is $\epsilon + 2^{-\Omega(\tau n)}$ close to (A, B) where B conditioned on $A = a$ has a min-entropy $(\delta - \tau)t$.

Finally, it is clear that if the sampling is done with a pseudorandom generator instead of a uniform function leads to a low min-entropy key, we have a distinguisher on the pseudorandom generator.

3.5 Implementation

In this section, we evaluate the efficiency of PuppyCipher {16,20,24}, Hound {16,20,24} and CoureurDesBois {16,20,24}, when implemented with the AES and PCLMULQDQ instructions (the latter being only used for the finite field arithmetic of CoureurDesBois) on a recent *Haswell* CPU. For each algorithm, we tested table-based white-box implementations and "secret" implementations where one has the knowledge of the key used to generate the tables. In the case of PuppyCipher, we also implemented the Hound variants.

The number of rounds we choose was directly deduced from proofs in the weak model (cf. Sections 3.3 and 3.4). Since this model essentially matches that of previous work [BBK14, BI15], this allows for a direct comparison.

The processor on our test machine was an Intel Xeon E5-1603v3, which has a maximal clock frequency of 2.8 GHz and a 10 MB cache (which is thus larger than the implementation sizes of the '16 instances). The machine has 32 GB of memory, in four sticks of 8 GB all clocked at 2133 MHz. All measurements were done on an idle system, without Turbo Boost activated[28]. As a reference, we first measured the performance of AES128 implemented with the AES instructions, given in Tbl. 3.3. We give the average (Avg.) number of clock cycles and the standard deviation (Std. Dev.) for one execution, both in the transient and steady regime (in practice, when performing series of independent runs, the transient regime only corresponds to the first run of the series). The average and standard deviation are computed from 25 series of 11 runs. The figures obtained from this test are coherent with the theoretical performance of the AES instruction set (even if slightly lower): on a Haswell architecture, the `aesenc` and `aesenclast` instructions both have a latency of 7 cycles, and the cost of a single full AES128 is dominated by the 10×7 calls to perform the 10 rounds of encryption.

[28] As a matter of fact, this CPU does not have Turbo Boost support.

	Transient Avg.	Transient Std. Dev.	Steady Avg.	Steady Std. Dev.
AES128	79	3.6	68	2.4

Table 3.3: Performance of a single call to AES128 with AES instructions on a Xeon E5-1603v3. All numbers are in clock cycles.

3.5.1 PuppyCipher

Writing a simple implementation of PuppyCipher is quite straightforward. The main potential for instruction-level parallelism (ILP) are the calls to the tables (or the analogous on-the-fly function calls); the rest of the cipher is chiefly sequential, especially the many intermediate calls to the (potentially reduced) AES. This parallelism is however somewhat limited, especially starting from PuppyCipher 24 where only two parallel calls to the tables can be made.

In all implementations, we precompute the sub-keys for the calls to AES (including the potential table function calls). Not doing so would only add a negligible overhead.

The performance measurements were done in a setting similar to the reference test on AES128 from above. We give the results for PuppyCipher {16,20,24} in Tbl. 3.4 and for Hound {16,20,24} in Tbl. 3.5. In both tables, we also express the performance in the steady regime as the number of equivalent AES128 calls (Eq. A) with AES instructions on the same platform (taken to be 68 cycles, as per Tbl. 3.3), as it is a block cipher with similar expected security, and as the number of equivalent ephemeral Diffie-Hellman key exchanges with the Four$\mathbb{Q}$ elliptic curve (Eq. F), one of the fastest current implementation of ECDHE [CL15] (measured at 92000 cycles on the Haswell architecture), as there is some overlap in what white-box and public-key cryptography try to achieve.

	Tr. Avg.	Tr. Std. Dev.	St. Avg.	St. Std. Dev.	Eq. A	Eq. F
PC 16 (white-box)	2960	130	2800	70	41	0.030
PC 16 (secret)	4140	60	3940	10	58	0.043
PC 20 (white-box)	13660	1000	11500	1190	169	0.125
PC 20 (secret)	4810	60	4540	100	67	0.049
PC 24 (white-box)	27570	1410	23390	1340	344	0.25
PC 24 (secret)	6760	120	6600	60	97	0.072

Table 3.4: Performance of a single call to PuppyCipher {16,20,24} ("PC") on a Xeon E5-1603v3. All numbers are in clock cycles, rounded to the nearest ten. The "white-box" instances are table-based, and the "secret" instances uses on-the-fly computations of the tables on their queried values. All calls to AES use the AES instructions.

Discussion.

As it was mentioned in Section 3.3, for a small white-box implementation such as the one of PuppyCipher 16, table-based implementations may be the most efficient way of implementing

	Tr. Avg.	Tr. Std. Dev.	St. Avg.	St. Std. Dev.	Eq. A	Eq. F
HD 16 (white-box)	2300	180	2190	130	32	0.024
HD 16 (secret)	3520	80	3280	2	48	0.036
HD 20 (white-box)	11870	980	9940	1030	146	0.11
HD 20 (secret)	4000	230	3700	65	54	0.040
HD 24 (white-box)	26540	1450	21740	1230	320	0.24
HD 24 (secret)	5490	60	5360	60	79	0.058

Table 3.5: Performance of a single call to HOUND {16,20,24} ("HD") on a Xeon E5-1603v3. All numbers are in clock cycles, rounded to the nearest ten. The "white-box" instances are table-based, and the "secret" instances uses on-the-fly computations of the tables on their queried values. All calls to AES use the AES instructions.

the cipher, especially as the entire tables can usually fit in the cache. However, from a certain size on, the random RAM accesses inherent to such implementations cost more than recomputing the necessary outputs of the tables (when the secret is known).

It is quite easy to estimate how much time is spent in RAM accesses compared to the time spent in calls to the (potentially reduced) AES. Indeed, knowing the number of rounds and the cost of one AES execution, one can subtract this contribution to the total. For instance, based on the cycle counts in the steady and transient regimes, for PUPPYCIPHER 24, at least $2380 = 35 \times 68$ and at most $2765 = 35 \times 79$ cycles are expected to be spent in AES instructions; the real figure in this case is about 2690 cycles, for an average cost per AES call of 77 cycles. All in all, this means that in steady regime, close to 90% of the time is spent in RAM accesses. This is understandingly slightly more for the HOUND 24 variant, where RAM accesses represent about 93% of the execution time.

It is also interesting to look at how many RAM accesses can effectively be done in parallel. As two to four table calls are independent every round, we may hope to partially hide the latency of some of these. For PUPPYCIPHER 24, removing one of the two table accesses decreases the cycle count to 19400 on average. This means that the second table call only adds less than 4000 cycles. Put another way, using a single table per round, one table access takes 490 cycles on average, but this goes down to 300 cycles when two tables are accessed per round. In the end, the 68 table access of PUPPYCIPHER 24 only cost an equivalent 42 purely sequential accesses. A similar analysis can be performed for PUPPYCIPHER 20 and PUPPYCIPHER 16, where the 69 and 72 parallel accesses cost 31 and 23 equivalent accesses respectively.

Comparison with SPACE. We can compare the performance of PUPPYCIPHER with the one of SPACE-(16,128) and SPACE-(24,128), which offer similar white-box implementation sizes as PUPPYCIPHER 16 and PUPPYCIPHER 24 respectively. As the authors of SPACE do not provide cycle counts for their ciphers but only the number of necessary cache or RAM accesses, a few assumptions are needed for a brief comparison. Both SPACE instances need 128 table accesses, which is much more than the 72 of PUPPYCIPHER 16 and 68 of PUPPYCIPHER 24. However, there is an extra cost in PUPPYCIPHER due to the sandwiching AES calls, which need to be taken into account. On the other hand, the table accesses in SPACE are necessarily sequential, which is not the case for PUPPYCIPHER, and we have just seen that parallel accesses can bring a

considerable gain. It is thus easiest to use our average sequential access times as a unit. In that respect, PuppyCipher 24 and Hound 24 cost on average $48 = 23390/490$ and $44 = 21790/490$ table accesses, which is significantly less than the 128 of SPACE-(24,128). Similarly, we measured one sequential table access for PuppyCipher 16 to take 59 cycles on average, and we thus have a cost of $47 = 2800/59$ and $37 = 2190/59$ for table accesses for PuppyCipher 16 and Hound 16.

The performance gap reduces slightly when one considers the case of "secret" implementations. As the tables of SPACE use the AES as a building block, the cost of a secret SPACE (24-128) implementation should correspond to approximately 128 sequential calls to AES; the corresponding PuppyCipher and Hound implementations cost an equivalent 97 and 79 AES respectively.

3.5.2 CoureurDesBois

The main advantage of CoureurDesBois compared to PuppyCipher is the higher degree of parallelism that it offers. Unlike PuppyCipher, the calls to AES can be made in parallel, and there is no limit either in the potential parallelism of table accesses. Because the output of the tables are of a bigger size, there is also fewer accesses to be made. Consequently, we expect CoureurDesBois to be quite more efficient than PuppyCipher.

A consequence of the higher parallelism of CoureurDesBois is that there are more potential implementation tradeoffs than for PuppyCipher. In our implementations, we chose to parallelize the AES calls up to four calls at a time, and the table accesses (or equivalent secret computations) at the level of one block (*i.e.* from eight parallel accesses for CoureurDesBois 16 to five for CoureurDesBois 24). The final step of CoureurDesBois also offers some parallelism; we have similarly regrouped the calls to AES used for random generation by four, and the finite field multiplications are regrouped by rows of eight.

The results for CoureurDesBois {16,20,24} are given in Tbl. 3.6.

	Tr. Avg.	Tr. Std. Dev.	St. Avg.	St. Std. Dev.	Eq. A	Eq. F
CDB 16 (white-box)	3190	460	2020	20	29.7	0.022
CDB 16 (secret)	3100	380	2150	30	31.6	0.023
CDB 20 (white-box)	7880	880	4700	600	69.1	0.051
CDB 20 (secret)	4060	460	2900	20	42.6	0.032
CDB 24 (white-box)	17360	980	11900	610	175	0.13
CDB 24 (secret)	4470	560	3050	30	44.9	0.033

Table 3.6: Performance of a single call to CoureurDesBois {16,20,24} ("CDB") on a Xeon E5-1603v3. All numbers are in clock cycles, rounded to the nearest ten. The "white-box" instances are table-based, and the "secret" instances uses on-the-fly computations of the tables on their queried values. All calls to AES use the AES instructions.

Discussion.

We can notice a few things from these results. First, CoureurDesBois is indeed more efficient than PuppyCipher; for instance, CoureurDesBois 24 is about twice as fast as Hound 24.

Second, the performance gap between secret and white-box implementations is somewhat smaller for the smaller instances of CoureurDesBois; on the other hand, the gap between transient and steady regime performance is slightly bigger than for PuppyCipher.

As pointed out above, more tradeoffs are possible in implementing CoureurDesBois than for PuppyCipher. As a result, it would be interesting to evaluate alternatives in practice.

Chapter 4

Cryptanalysis of the CLT15 Multilinear Map

4.1 Introduction

Cryptographic multilinear maps are a powerful and versatile tool to build cryptographic schemes, ranging from one-round multipartite Diffie-Hellman to witness encryption and general program obfuscation. The notion of cryptographic multilinear map was first introduced by Boneh and Silverberg in 2003, as a natural generalization of bilinear maps such as pairings on elliptic curves [BS03]. However it was not until 2013 that the first concrete instantiation over ideal lattices was realized by Garg, Gentry and Halevi [GGH13a], quickly inspiring another construction over the integers by Coron, Lepoint and Tibouchi [CLT13]. Alongside these first instantiations, a breakthrough result by Garg, Gentry, Halevi, Raykova, Sahai and Waters achieved (indistinguishability) obfuscation for all circuits from multilinear maps [GGH+13b]. From that point multilinear maps have garnered considerable interest in the cryptographic community, and a host of other applications have followed.

However this wealth of applications rests on the relatively fragile basis of only three constructions of multilinear maps to date: namely the original construction over ideal lattices [GGH13a], the construction over the integers [CLT13], and another recent construction over lattices [GGH15]. Moreover none of these constructions relies on standard hardness assumptions. In fact the first two constructions have since been broken for applications requiring low-level encodings of zero, including the "direct" application to one-round multipartite Diffie-Hellman [HJ15, CHL+15]. Thus building candidate multilinear maps and assessing their security may be regarded as a work in progress, and research in this area has been very active in recent years.

Following the attack by Cheon *et al.* on the [CLT13] multilinear map over the integers, several attempts to repair the scheme were published on ePrint, which hinged on hiding encodings of zero in some way; however these attempts were quickly proven insecure [CGH+15]. At Crypto 2015, Coron, Lepoint and Tibouchi set out to repair their scheme by following a different route [CLT15]: they essentially retained the structure of encodings from [CLT13], but added a new type of noise designed to thwart Cheon *et al.*'s approach. Their construction was thus able to retain the attractive features of the original, namely conceptual simplicity, relative efficiency, and wide range of presumed hard problems on which applications could be built.

In the remainder of this chapter, we propose a polynomial attack on the new multilinear map over the integers presented by Coron, Lepoint and Tibouchi at Crypto 2015 [CLT15]. The attack operates by computing the secret parameter x_0, and from there all other secret parameters

can be recovered via (a close variant of) Cheon *et al.*'s attack [CHL+15].

In the optimized version of the scheme where an exact multiple of x_0 is provided in the public parameters, the attack recovers x_0 instantly. In the more general non-optimized version of the scheme, the complexity of our polynomial attack is very close to the security parameters for the concrete instances implemented in [CLT15], *e.g.* 2^{82} for the 80-bit instance.

Moreover the attack applies to virtually all possible applications of the CLT15 multilinear map. Indeed, while it does require low-level encodings of zero, these encodings are provided by the ladders given in the public parameters. In this respect CLT15 is weaker than CLT13. A more detailed look at the impact of our attack is provided in Section 4.1.2. Our attacks have been verified on the reference implementation of CLT15.

We also describe a secondary, probabilistic attack on CLT15, with the same effect as our main attack. The probabilistic attack relies on finding and exploiting divisors of the secret parameter v_0. While it is less simple than the main attack, it offers a different approach to attacking the scheme.

4.1.1 Overview of the Main Attack

We begin by briefly recalling the CLT15 multilinear map (more precisely, graded encoding scheme). The message space is $\mathbb{Z}_{g_1} \times \cdots \times \mathbb{Z}_{g_n}$ for some small primes $g_1, \ldots, g_n$, and $(m_1, \ldots, m_n)$ is encoded at some level $k \leq \kappa$ as:

$$\mathsf{CRT}_{(p_i)}\Big(\frac{r_i g_i + m_i}{z^k}\Big) + a x_0$$

where:

(p_i)	is a sequence of n large primes.
x_0	$= \prod p_i$.
$\mathsf{CRT}_{(p_i)}(x_i)$	is the unique integer in $(-x_0/2, x_0/2]$ congruent to x_i modulo p_i.
z	is a fixed secret integer modulo x_0.
r_i	is a small noise.
a	is another noise.

Encodings at the same level can be added together, and the resulting encoding encodes the sum of the messages. Similarly encodings at levels i and j can be multiplied to yield an encoding at level $i + j$ of the coordinate-wise product of the encoded messages. This behavior holds as long as the values $r_i g_i + m_i$ do not go over p_i, *i.e.* reduction modulo p_i does not interfere. In order to prevent the size of encodings from increasing as a result of additions and multiplications, a *ladder* of encodings of zero of increasing size is published at each level. Encodings can then be reduced by subtracting elements of the ladder at the same level.

The power of the multilinear map comes from the zero-testing procedure, which allows users to test whether an encoding at the maximal level κ encodes zero. This is achieved by publishing a so-called zero-testing parameter denoted $\boldsymbol{p}_{zt} \in \mathbb{Z}$, together with a large prime $N \gg x_0$. An encoding at the maximal level κ may be written as:

$$e = \sum (r_i + m_i g_i^{-1} \bmod p_i) u_i + a x_0$$
$$\text{where } u_i \triangleq \big(g_i z^{-\kappa} (p_i^*)^{-1} \bmod p_i\big)\, p_i^* \quad \text{with } p_i^* = \prod_{j \neq i} p_j.$$

That is, some constants independent of the encoding have been folded with the CRT coefficients into u_i. Now $\boldsymbol{p}_{zt}$ is chosen such that $v_i \triangleq u_i \boldsymbol{p}_{zt} \bmod N$ and $v_0 \triangleq x_0 \boldsymbol{p}_{zt} \bmod N$ satisfy $|v_i| \ll N$ and $|v_0| \ll N$. In this way, for any encoding e of zero at level κ, since $m_i = 0$, we have:

$$|e\boldsymbol{p}_{zt} \bmod N| = \Big|\sum r_i v_i + a v_0\Big| \ll N$$

provided the noises r_i and a are small enough. Thus, users can test whether e is an encoding of zero at level κ by checking whether $|e\boldsymbol{p}_{zt} \bmod N| \ll N$.

Integer Extraction.

Our attack proceeds in two steps. As a first step, we define the integer extraction procedure $\phi : \mathbb{Z} \to \mathbb{Z}$. In short, ϕ computes $\sum_i r_i v_i + a v_0$ over the integers for any level-κ encoding e (of size up to the largest ladder element). Note that this value is viewed over the integers and not modulo N. If e is "small", then $\phi(e) = e\boldsymbol{p}_{zt} \bmod N$, *i.e.* ϕ matches the computation from the zero-testing procedure.

If e is "large" on the other hand, then e would need to be reduced by the ladder before zero-testing can be applied. However the crucial observation is that ϕ is $\mathbb{Z}$-linear as long as the values $r_i g_i + m_i$ associated with each encoding do not go over p_i. Thus e can be ladder-reduced into e', then $\phi(e') = e'\boldsymbol{p}_{zt} \bmod N$ is known, and $\phi(e)$ can be recovered from $\phi(e')$ by compensating the ladder reduction using $\mathbb{Z}$-linearity. In a nutshell, ϕ allows us to ignore ladder reductions in equations appearing in the rest of the attack.

Recovering x_0.

In the optimized variant of the scheme implemented in [CLT15], a small multiple qx_0 of x_0 is given in the public parameters. In that case qx_0 may be regarded as an encoding of zero at level κ, and $\phi(qx_0) = qv_0$. Since this holds over the integers, we can compute $q = \gcd(qx_0, qv_0)$ and then $x_0 = qx_0/q$.

In the general case where no exact multiple of x_0 is given in the public parameters, pick $n+1$ encodings a_i at some level t, and $n+1$ encodings of zero b_i at level $\kappa - t$. Note that ladder elements provide encodings of zero even if the scheme itself does not. Then compute:

$$\omega_{i,j} \triangleq \phi(a_i b_j).$$

If we write $a_i \bmod v_0 = \mathsf{CRT}_{(p_j)}(a_{i,j}/z^t)$ and $b_i \bmod v_0 = \mathsf{CRT}_{(p_j)}(r_{i,j} g_j / z^{\kappa - t})$, then we get:

$$\omega_{i,j} \bmod v_0 = \sum_k a_{i,k} r_{j,k} v_k \bmod v_0.$$

Similar to Cheon *et al.*'s attack on the CLT13 multilinear map, this equality can be viewed as a matrix product. Indeed, let Ω denote the $(n+1) \times (n+1)$ integer matrix with entries $\omega_{i,j}$, let A denote the $(n+1) \times n$ integer matrix with entries $a_{i,j}$, let R denote the $(n+1) \times n$ integer matrix with entries $r_{i,j}$, and finally let V denote the $n \times n$ diagonal matrix with diagonal entries v_i. If we embed everything into $\mathbb{Z}/v_0\mathbb{Z}$, then we have:

$$\Omega = A \cdot V \cdot R^{\mathrm{T}} \qquad \text{in } \mathbb{Z}/v_0\mathbb{Z}.$$

Since A and R are $(n+1) \times n$ matrices, this implies that Ω is not full-rank when embedded into $\mathbb{Z}/v_0\mathbb{Z}$. As a consequence v_0 divides $\det(\Omega)$. We can repeat this process with different choices of the families (a_i), (b_i) to build another matrix Ω' with the same property. Finally we recover v_0 as $v_0 = \gcd(\det(\Omega), \det(\Omega'))$, and $x_0 = v_0/\boldsymbol{p}_{zt} \bmod N$.

Recovering other secret parameters.

Once x_0 is known, Cheon *et al.*'s attack can be applied by taking all values modulo v_0, and every remaining secret parameter is recovered, fully breaking the scheme.

4.1.2 Impact of the Attack

Two variants of the CLT15 multilinear map should be considered. Either a small multiple of x_0 is provided in the public parameters. In that case x_0 can be recovered instantly, and the scheme becomes equivalent to CLT13 in terms of security (cf. Section 4.6.1). In particular it falls victim to Cheon *et al.*'s attack when low-level encodings of zero are present, but it may still be secure for applications that do not require such encodings, such as obfuscation. However the scheme is strictly less efficient than CLT13 by construction, so there is no point in using CLT15 for those applications.

Otherwise, if no small multiple of x_0 is given out in the public parameters, then ladders of encodings of zero must be provided at levels below the maximal level. Thus we have access to numerous encodings of zero below the maximal level, even if the particular application of multilinear maps under consideration does not require them. As a result our determinant-based attack is applicable (cf. Section 4.6.5), and we still recover x_0 in polynomial time, albeit less efficiently than the previous case. Moreover once x_0 is recovered, encodings of zero provided by the ladder enable Cheon *et al.*'s attack, and every secret parameter is recovered.

In summary, the optimized version of CLT15 providing a small multiple of x_0 is no more secure than CLT13, and less efficient. On the other hand in the general non-optimized case, the scheme is broken for virtually all possible applications due to encodings of zero provided by the ladder. Thus overall the CLT15 scheme can be considered fully broken.

4.1.3 Independent Attack by Cheon *et al.*

Another polynomial attack on CLT15 was discovered independently by Cheon, Lee and Ryu (CLR) [CLR15]. The impact of both attacks is the same, and their practical complexity is similar. Our attacks were merged for publication at Eurocrypt [CFL+16].

The CLR attack relies on integer extraction as well, which is defined in the same manner. The second half of the attack is where it differs. The CLR attack looks into the exact expression of the value a in the term av_0 appearing in integer extractions. This makes it possible to uncover a matrix product similar to Cheon *et al.*'s original attack on CLT13, albeit a more complex one. By contrast our attack treats the value a in av_0 as a noise, which we get rid off by recovering v_0 and taking equations modulo v_0.

4.1.4 Layout of the Chapter

We begin by defining multilinear maps and graded encoding schemes in Section 4.3. The CLT15 construction itself is described in Section 4.4. In Section 4.5 we recall Cheon *et al.*'s attack on CLT13 since it serves as a follow-up to our attack once x_0 is recovered, and shares similar ideas. Readers already familiar with the CLT15 multilinear map can skip straight to Section 4.6 where we describe our main attack. The main attack has been verified on a reference implementation (some implementation issues are discussed in the appendix of [MF15]). As an alternative to the main attack, a probabilistic attack is given in Section 4.7.

4.2 Notation

For n an integer, size(n) is the size of n in bits.

Modular arithmetic.

The group $\mathbb{Z}/n\mathbb{Z}$ of integers modulo n is denoted by $\mathbb{Z}_n$. The notation "mod p" should be understood as having the lowest priority. For instance, the expression $a \cdot b \bmod p$ is equivalent to $(a \cdot b) \bmod p$.

We always view $a \bmod p$ as an integer in $\mathbb{Z}$. The representative closest to zero is always chosen, positive in case of tie. In other words $-p/2 < a \bmod p \leq p/2$.

Chinese Remainder Theorem.

Given n prime numbers (p_i), we define p_i^* as in [Hal15a]:

$$p_i^* = \prod_{j \neq i} p_j.$$

For $(x_1, \ldots, x_n) \in \mathbb{Z}^n$, let $\mathsf{CRT}_{(p_i)}(x_i)$ denote the unique integer in $\mathbb{Z} \cap (-\frac{1}{2}\prod p_i, \frac{1}{2}\prod p_i]$ such that $\mathsf{CRT}_{(p_i)}(x_i) \bmod p_i = x_i \bmod p_i$, as per the Chinese Remainder Theorem.

It is useful to observe that for any $(x_1, \ldots, x_n) \in \mathbb{Z}^n$:

$$\mathsf{CRT}_{(p_i)}(x_i p_i^*) = \sum_i x_i p_i^* \bmod \prod_i p_i. \tag{4.1}$$

4.3 Short Introduction to Multilinear Maps

In this section we give a brief introduction to multilinear maps. In particular we only consider symmetric multilinear maps. We refer the interested reader to [GGH13a, Hal15b] for a more thorough presentation.

4.3.1 Multilinear Maps and Graded Encoding Schemes

Cryptographic multilinear maps were introduced by Boneh and Silverberg [BS03], as a natural generalization of bilinear maps stemming from pairings on elliptic curves, which had found striking new applications in cryptography [Jou00, BF01, ...]. A (symmetric) multilinear map is defined as follows.

Definition 19 (Multilinear Map [BS03])**.** *Given two groups $\mathbb{G}, \mathbb{G}_T$ of the same prime order, a map $e : \mathbb{G}^\kappa \to \mathbb{G}_T$ is a* κ-multilinear map *iff it satisfies the following two properties:*

1. *for all $a_1, \ldots, a_\kappa \in \mathbb{Z}$ and $x_1, \ldots, x_\kappa \in \mathbb{G}$,*

$$e(x_1^{a_1}, \ldots, x_\kappa^{a_\kappa}) = e(x_1, \ldots, x_\kappa)^{a_1 \cdots a_\kappa}$$

2. *if g is a generator of $\mathbb{G}$, then $e(g, \ldots, g)$ is a generator of $\mathbb{G}_T$.*

A natural special case are *leveled* multilinear maps:

Definition 20 (Leveled Multilinear Map [HSW13])**.** *Given $\kappa + 1$ groups $\mathbb{G}_1, \ldots, \mathbb{G}_\kappa, \mathbb{G}_T$ of the same prime order, and for each $i \leq \kappa$, a generator $g_i \in \mathbb{G}_i$, a* κ-leveled multilinear map *is a set of bilinear maps $\{e_{i,j} : \mathbb{G}_i \times \mathbb{G}_j \to G_{i+j} | i, j, i + j \leq \kappa\}$ such that for all i, j with $i + j \leq \kappa$, and all $a, b \in \mathbb{Z}$:*

$$e_{i,j}(g_i^a, g_j^b) = g_{i+j}^{ab}.$$

Similar to public-key encryption [DH76b] and identity-based cryptosystems [Sha85], multilinear maps were originally introduced as a compelling target for cryptographic research, without a concrete instantiation [BS03]. The first multilinear map was built ten years later in the breakthrough construction of Garg, Gentry and Halevi [GGH13a]. More accurately, what the authors proposed was a *graded encoding scheme*, and to this day all known cryptographic multilinear maps constructions are actually variants of graded encoding schemes [Hal15b]. For this reason, and because both constructions have similar expressive power, the term "multilinear map" is used in the literature in place of "graded encoding scheme", and we follow suit.

Graded encoding schemes are a relaxed definition of leveled multilinear map, where elements x_i^a for $x_i \in \mathbb{G}_i, a \in \mathbb{Z}$ are no longer required to lie in a group. Instead, they are regarded as "encodings" of a ring element a at level i, with no assumption about the underlying structure. Formally, encodings are thus defined as general binary strings in $\{0,1\}^*$. In the following definition, $S_i^{(\alpha)}$ should be regarded as the set of encodings of a ring element α at level i.

Definition 21 (Graded Encoding System [GGH13a])**.** *A* κ-graded encoding system *consists of a ring R and a system of sets $\mathcal{S} = \{S_i^{(\alpha)} \subset \{0,1\}^* | \alpha \in R, 0 \leq i \leq \kappa\}$, with the following properties:*

1. *For each fixed i, the sets $S_i^{(\alpha)}$ are pairwise disjoint as α spans R.*

2. *There is an associative binary operation '$+$' and a self-inverse unary operation '$-$' on $\{0,1\}^*$ such that for every $\alpha_1, \alpha_2 \in R$, every $i \leq \kappa$, and every $u_1 \in S_i^{(\alpha_1)}, u_2 \in S_i^{(\alpha_2)}$, it holds that:*

$$u_1 + u_2 \in S_i^{(\alpha_1+\alpha_2)} \quad \text{and} \quad -u_1 \in S_i^{(-\alpha_1)}$$

   where $\alpha_1 + \alpha_2$ and $-\alpha_1$ are addition and negation in R.

3. *There is an associative binary operation '$\times$' on $\{0,1\}^*$ such that for every $\alpha_1, \alpha_2 \in R$, every $i_1, i_2 \in \mathbb{N}$ such that $i_1 + i_2 \leq \kappa$, and every $u_1 \in S_{i_1}^{(\alpha_1)}, u_2 \in S_{i_2}^{(\alpha_2)}$, it holds that $u_1 \times u_2 \in S_{i_1+i_2}^{(\alpha_1 \cdot \alpha_2)}$. Here $\alpha_1 \cdot \alpha_2$ is the multiplication in R, and $i_1 + i_2$ is the integer addition.*

Observe that a leveled multilinear map is a graded encoding system where $R = \mathbb{Z}$ and, with the notation from the definitions, $S_i^{(\alpha)}$ contains the single element g_i^α. Also note that the behavior of addition and multiplication of encodings with respect to the levels i is the same as that of a graded ring, hence the *graded* qualifier.

All known constructions of graded encoding schemes do not fully realize the previous definition, insofar as they are "noisy"[29]. That is, all encodings have a certain amount of noise; each operation, and especially multiplication, increases this noise; and the correctness of the scheme breaks down if the noise goes above a certain threshold. The situation in this regard is similar to somewhat homomorphic encryption schemes.

[29] In fact the question of achieving the functionality of multilinear maps without noise may be regarded as an important open problem [Zim15].

4.3.2 Multilinear Map Procedures

The exact interface offered by a multilinear map, and called upon when it is used as a primitive in a cryptographic scheme, varies depending on the scheme. However the core elements are the same. Below we reproduce the procedures for manipulating encodings defined in [CLT15], which are a slight variation of [GGH13a].

In a nutshell, the scheme relies on a trusted third party that generates the instance (and is typically no longer needed afterwards). Users of the instance (that is, everyone but the generating trusted third party) cannot encode nor decode arbitrary encodings: they can only combine existing encodings using addition, negation and multiplication, and subject to the limitation that the level of an encoding cannot exceed κ. The power of the multilinear map comes from the zero-testing (resp. extraction) procedure, which allows users to test whether an encoding at level κ encodes zero (resp. roughly get a λ-bit "hash" of the value encoded by a level-κ encoding).

Here users are also given access to random level-0 encodings, and have the ability to re-randomize encodings, as well as promote any encoding to a higher-level encoding of the same element. These last functionalities are tailored towards the application of multilinear maps to one-round multi-party Diffie-Hellman. In general different applications of multilinear map require different subsets of the procedures below, and sometimes variants of them.

$\mathsf{instGen}(1^\lambda, 1^\kappa)$: the randomized instance procedure takes as input the security parameter λ, the multilinearity level κ, and outputs the public parameters $(\mathsf{pp}, \boldsymbol{p}_{zt})$, where pp is a description of a κ-graded encoding system as above, and $\boldsymbol{p}_{zt}$ is a zero-test parameter (see below).

$\mathsf{samp}(\mathsf{pp})$: the randomized sampling procedure takes as input the public parameters pp and outputs a level-0 encoding $u \in S_0^{(\alpha)}$ for a nearly uniform $\alpha \in R$.

$\mathsf{enc}(\mathsf{pp}, i, u)$: the possibly randomized encoding procedure takes as input the public parameters pp, a level $i \leq \kappa$, and a level-0 encoding $u \in S_0^{\alpha}$ for some $\alpha \in R$, and outputs a level-i encoding $u' \in S_i^{(\alpha)}$.

$\mathsf{reRand}(\mathsf{pp}, i, u)$: the randomized rerandomization procedure takes as input the public parameters pp, a level $i \leq \kappa$, and a level-i encoding $u \in S_i^{\alpha}$ for some $\alpha \in R$, and outputs another level-i encoding $u' \in S_i^{(\alpha)}$ of the same α, such that for any $u_1, u_2 \in S_i^{(\alpha)}$, the output distributions of $\mathsf{reRand}(\mathsf{pp}, i, u_1)$ and $\mathsf{reRand}(\mathsf{pp}, i, u_2)$ are nearly the same.

$\mathsf{neg}(\mathsf{pp}, u)$: the negation procedure is deterministic and that takes as input the public parameters pp, and a level-i encoding $u \in S_i^{(\alpha)}$ for some $\alpha \in R$, and outputs a level-i encoding $u' \in S_i^{(-\alpha)}$.

$\mathsf{add}(\mathsf{pp}, u_1, u_2)$: the addition procedure is deterministic and takes as input the public parameters pp, two level-i encodings $u_1 \in S_i^{(\alpha_1)}, u_2 \in S_i^{(\alpha_2)}$ for some $\alpha_1, \alpha_2 \in R$, and outputs a level-i encoding $u' \in S_i^{(\alpha_1+\alpha_2)}$.

$\mathsf{mult}(\mathsf{pp}, u_1, u_2)$: the multiplication procedure is deterministic and takes as input the public parameters pp, two encodings $u_1 \in S_i^{(\alpha_1)}, u_2 \in S_j^{(\alpha_2)}$ of some $\alpha_1, \alpha_2 \in R$ at levels i and j such that $i + j \leq \kappa$, and outputs a level-$(i + j)$ encoding $u' \in S_{i+j}^{(\alpha_1 \cdot \alpha_2)}$.

$\mathsf{isZero}(\mathsf{pp}, u)$: the zero-testing procedure is deterministic and takes as input the public parameters pp, and an encoding $u \in S_\kappa^{(\alpha)}$ of some $\alpha \in R$ at the maximum level κ, and outputs 1 if $\alpha = 0$, 0 otherwise, with negligible probability of error (over the choice of $u \in S_\kappa^{(\alpha)}$).

$\mathsf{ext}(\mathsf{pp}, \boldsymbol{p}_{zt}, u)$: the extraction procedure is deterministic and takes as input the public parameters pp, the zero-test parameter $\boldsymbol{p}_{zt}$, and an encoding $u \in S_\kappa^{(\alpha)}$ of some $\alpha \in R$ at the maximum level κ, and outputs a λ-bit string s such that:

1. For $\alpha \in R$ and $u_1, u_2 \in S_\kappa^{(\alpha)}$, $\mathsf{ext}(\mathsf{pp}, \boldsymbol{p}_{zt}, u_1) = \mathsf{ext}(\mathsf{pp}, \boldsymbol{p}_{zt}, u_2)$.
2. The distribution $\{\mathsf{ext}(\mathsf{pp}, \boldsymbol{p}_{zt}, v) | \alpha \leftarrow R, v \in S_\kappa^{(\alpha)}\}$ is nearly uniform over $\{0,1\}^\lambda$.

4.4 The CLT15 Multilinear Map

4.4.1 The CLT15 Multilinear Map over the Integers

Shortly after the multilinear map over ideal lattices by Garg, Gentry and Halevi [GGH13a], another construction over the integers was proposed by Coron, Lepoint and Tibouchi [CLT13]. However a devastating attack was published by Cheon, Han, Lee, Ryu and Stehlé at Eurocrypt 2015 (on ePrint in late 2014). In the wake of this attack, a revised version of their multilinear map over the integers was presented by Coron, Lepoint and Tibouchi at Crypto 2015 [CLT15]. In the remainder, we will refer to the original construction over the integers as CLT13, and to the new version from Crypto 2015 as CLT15.

In this section we recall the CLT15 construction. We omit aspects of the construction that are not relevant to our attack, and refer the reader to [CLT15] for more details. The message space is $R = \mathbb{Z}_{g_1} \times \cdots \times \mathbb{Z}_{g_n}$, for some (relatively small) primes $g_i \in \mathbb{N}$. An encoding of a message $(m_1, \ldots, m_n) \in \mathbb{Z}_{g_1} \times \cdots \times \mathbb{Z}_{g_n}$ at level $k \leq \kappa$ has the following form:

$$e = \mathsf{CRT}_{(p_i)}\left(\frac{r_i g_i + m_i}{z^k} \bmod p_i\right) + a x_0 \tag{4.2}$$

where:

- The p_i's are n large secret primes.
- The r_i's are random noise such that $|r_i g_i + m_i| \ll p_i$.
- $x_0 = \prod_{i \leq n} p_i$.
- z is a fixed secret integer modulo x_0.
- a is random noise.

The scheme relies on the following parameters:

$$\begin{aligned} \lambda &: \text{the security parameter.} \\ \kappa &: \text{the multilinearity level.} \\ n &: \text{the number of primes } p_i. \\ \eta &: \text{the bit length of secret primes } p_i. \\ \gamma = n\eta &: \text{the bit length of } x_0. \\ \rho &: \text{the bit length of the } g_i\text{'s and initial } r_i\text{'s.} \end{aligned}$$

Addition, negation and multiplication of encodings is exactly addition, negation and multiplication over the integers. Indeed, m_i is recovered from e as $m_i = (e \bmod p_i) \bmod g_i$, and as long as $r_i g_i + m_i$ does not go over p_i, addition and multiplication will go through both moduli. Thus we have defined encodings and how to operate on them.

Regarding the sampling procedure from Section 4.3.2, for our purpose, it suffices to know that it is realized by publishing a large number of level-0 encodings of random elements. Users can then sample a new random element as a subset sum of published elements. Likewise, the rerandomization procedure is achieved by publishing a large number of encodings of zero at each level, and an element is re-randomized by adding a random subset sum of encodings of zero at the same level. The encoding procedure is realized by publishing a single level-1 encoding y of 1 (by which we mean $(1,\dots,1) \in \mathbb{Z}_{g_1} \times \cdots \times \mathbb{Z}_{g_n}$): any encoding can then be promoted to an encoding of the same element at a higher level by multiplying by y.

Zero-testing in CLT13.

We now move on to the crucial zero-testing procedure. This is where CLT13 and CLT15 differ. We begin by briefly recalling the CLT13 approach.

In CLT13, the product x_0 of the p_i's is public. In particular, every encoding can be reduced modulo x_0, and every value below should be regarded as being modulo x_0. Let $p_i^* = \prod_{j\neq i} p_j$. Using (4.1), define:

$$\boldsymbol{p}_{zt} \triangleq \sum_{i\leq n} \Big(\frac{h_i z^\kappa}{g_i} \bmod p_i\Big) p_i^* = \mathsf{CRT}_{(p_i)}\Big(\frac{h_i z^\kappa}{g_i} p_i^* \bmod p_i\Big) \qquad \bmod x_0.$$

where the h_i's are some relatively small numbers with $|h_i| \ll p_i$. Now take a level-κ encoding of zero:

$$e = \mathsf{CRT}_{(p_i)}\Big(\frac{r_i g_i}{z^\kappa} \bmod p_i\Big) \qquad \bmod x_0.$$

Since multiplication acts coordinate-wise on the CRT components, using (4.1) again, we have:

$$\omega \triangleq e\boldsymbol{p}_{zt} = \mathsf{CRT}_{(p_i)}(h_i r_i p_i^*) = \sum_i h_i r_i p_i^* \qquad \bmod x_0.$$

Since $p_i^* = x_0/p_i$, as long as we set our parameters so that $|h_i r_i| \ll p_i$, we have $|\omega| \ll x_0$.

Thus the zero-testing procedure is as follows: for a level-κ encoding e, compute $\omega = e\boldsymbol{p}_{zt} \bmod x_0$. Output 1, meaning we expect e to encode zero, iff the ν most significant bits of ω are zero, for an appropriately chosen ν. In [CLT13], multiple $\boldsymbol{p}_{zt}$'s can be defined in order to avoid false positives; we restrict our attention to a single $\boldsymbol{p}_{zt}$.

Zero-testing in CLT15.

In CLT13, an encoding at some fixed level is entirely defined by its vector of associated values $c_i = r_i g_i + m_i$. Moreover, addition and multiplication of encodings act coordinate-wise on these values, and the value of the encoding itself is $\mathbb{Z}_{x_0}$-linear as a function of these values. Likewise, ω is $\mathbb{Z}_{x_0}$-linear as a function of the r_i's. This nice structure is an essential part of what makes the devastating attack by Cheon *et al.* [CHL+15] possible. In CLT15, the authors set out to break this structure by introducing a new noise component a.

For this purpose, the public parameters include a new prime number $N \gg x_0$, with $\mathrm{size}(N) = \gamma + 2\eta + 1$. Meanwhile x_0 is kept secret, and no longer part of the public parameters. Encodings are thus no longer reduced modulo x_0, and take the general form given in (4.2), including a new noise value a. Equivalently, we can write an encoding e of (m_i) at level k as:

$$e = \sum_i \left(r_i + m_i(g_i^{-1} \bmod p_i)\right) u_i + a x_0 \tag{4.3}$$

$$\text{with } u_i \triangleq \left(g_i z^{-k} (p_i^*)^{-1} \bmod p_i\right) p_i^*.$$

That is, we fold the $g_i z^{-k}$ multiplier of r_i with the CRT coefficient into u_i.

The zero-testing parameter $\boldsymbol{p}_{zt}$ is now defined modulo N in such a way that:

$$\begin{array}{lll} & v_0 \triangleq x_0 \boldsymbol{p}_{zt} \bmod N & \forall i, v_i \triangleq u_i \boldsymbol{p}_{zt} \bmod N \\ \text{satisfy:} & |v_0| \ll N & |v_i| \ll N \end{array} \tag{4.4}$$

To give an idea of the sizes involved, $\text{size}(v_0) \approx \gamma$ and $\text{size}(v_i) \approx \gamma + \eta$ for $i > 0$. We refer the reader to [CLT15] for how to build such a $\boldsymbol{p}_{zt}$. The point is that if e is an encoding of zero at level κ, then we have:

$$\omega = e\boldsymbol{p}_{zt} \bmod N = \sum r_i v_i + a v_0 \bmod N.$$

In order for this quantity to be smaller than N, the size of a must be somehow controlled. Conversely as long as a is small enough and the noise satisfies $|r_i| \ll p_i$ then $|\omega| \ll N$. We refer the reader to [CLT15] for an exact choice of parameters.

Thus the size of a must be controlled. The term ax_0 will be dominant in (4.3) in terms of size, so decreasing a is the same as decreasing the size of the encoding as a whole. The scheme requires a way to achieve this without altering the encoded value (and without publishing x_0).

For this purpose, inspired by [VDGHV10], a *ladder* $(X_i^{(k)})_{i \leq \ell}$ of encodings of zero of increasing size is published for each level $k \leq \kappa$. The size of an encoding e at level k can then be reduced without altering the encoded value by recursively subtracting from e the largest ladder element smaller than e, until e is smaller than X_0. More precisely we can choose X_0 small enough that the previous zero-testing procedure goes through, and then choose X_ℓ twice the size of X_0, so that the product of any two encodings smaller than X_0 can be reduced to an encoding smaller than X_0. After each addition and multiplication, the size of the resulting encoding is reduced via the ladder.

In the end, the zero-testing procedure is very similar to CLT13: given a (ladder-reduced) level-κ encoding e, compute $\omega = e\boldsymbol{p}_{zt} \bmod N$. Then output 1, meaning we expect e to encode zero, iff the ν high-order bits of ω are zero.

Extraction.

The extraction procedure simply outputs the ν high-order bits of ω, computed as above. For both CLT13 and CLT15, it can be checked that they only depend on the m_i's (as opposed to the noises a and the r_i's).

4.5 Cheon *et al.*'s Attack on CLT13

In this section we provide a short description of Cheon *et al.*'s attack on CLT13 [CHL+15], as elements of this attack appear in our own. We actually present (a close variant of) the slightly simpler version in [CGH+15].

Assume we have access to a level-0 encoding a of some random value, n level-1 encodings (b_i) of zero, and a level-1 encoding y of 1. This is the case for one-round multi-party Diffie-Hellman (see previous section). Let $a_i = a \bmod p_i$, *i.e.* a_i is the i-th value "$r_i g_i + m_i$" associated with a. For $i \leq n$, define $r_{i,j} = b_i z / g_j \bmod p_j$, *i.e.* $r_{i,j}$ is the j-th value "r_j" associated with b_i (recall that b_i is an encoding of zero, so $m_j = 0$). Finally let $y_k = yz \bmod p_k$.

Now compute:

$$\begin{array}{ll} e_{i,j} = a \cdot b_i \cdot b_j \cdot y^{\kappa-2} \bmod x_0 & \omega_{i,j} = e_{i,j} \boldsymbol{p}_{zt} \bmod x_0 \\ e'_{i,j} = \quad b_i \cdot b_j \cdot y^{\kappa-2} \bmod x_0 & \omega'_{i,j} = e'_{i,j} \boldsymbol{p}_{zt} \bmod x_0 \end{array}$$

Note that:

$$\begin{aligned}\omega_{i,j} &= \sum_k \left(a_k \frac{r_{i,k} g_k}{z} \frac{r_{j,k} g_k}{z} \frac{y_k^{\kappa-2}}{z^{\kappa-2}} \frac{h_k z^\kappa}{g_k} \bmod p_k\right) p_k^* \\ &= \sum_k a_k r_{i,k} r_{j,k} c_k \qquad \text{with } c_k = g_k y_k^{\kappa-2} h_k p_k^*. \end{aligned} \tag{4.5}$$

Crucially, in the second line, the modulo p_k disappears and the equation holds over the integers, because $e_{i,j}$ is a valid encoding of zero, so the correctness of the scheme requires $|e_{i,j} z^\kappa / g_k \bmod p_k| \ll p_k$.

Equation (4.5) may be seen as a matrix multiplication. Indeed, define Ω, resp. Ω', as the $n \times n$ matrix with entries $\omega_{i,j}$, resp. $\omega'_{i,j}$, and likewise R with entries $r_{i,j}$. Moreover let A, resp. C, be the diagonal matrix with diagonal entries a_i, resp. c_i. Then (4.5) may be rewritten:

$$\begin{aligned}\Omega &= R \cdot A \cdot C \cdot R^{\mathrm{T}} \\ \Omega' &= R \cdot C \cdot R^{\mathrm{T}} \\ \Omega \cdot (\Omega')^{-1} &= R \cdot A \cdot R^{-1}.\end{aligned}$$

Here matrices are viewed over $\mathbb{Q}$ for inversion (they are invertible whp).

Once $\Omega \cdot (\Omega')^{-1}$ has been computed, the (diagonal) entries of A can be recovered as its eigenvalues. In practice this can be achieved by computing the characteristic polynomial, and all computations can be performed modulo some prime p larger than the a_i's (which are size 2ρ).

Thus we recover the a_i's, and by definition $a_i = a \bmod p_i$, so p_i can be recovered as $p_i = \gcd(a - a_i, x_0)$. From there it is trivial to recover all other secret parameters of the scheme.

4.6 Main Attack

4.6.1 On the Impact of Recovering x_0

If x_0 is known, CLT15 essentially collapses to CLT13. In particular, all encodings can be reduced modulo x_0 so ladders are no longer needed. What is more, all $\omega_{i,j}$'s from Cheon *et al.*'s attack can be reduced modulo $v_0 = x_0 \boldsymbol{p}_{zt} \bmod N$, which effectively removes the new noise a. As a direct consequence Cheon *et al.*'s attack goes through and all secret parameters are recovered (cf. [CLT15, Section 3.3]). Moreover ladder elements reduced by x_0 provide low-level encodings of zero even if the scheme itself does not. Also note that Cheon *et al.*'s attack is quite efficient as it can be performed modulo any prime larger than the values we are trying to recover, *i.e.* larger than $2^{2\rho}$.

Our attack recovers x_0. As a first step, we introduce *integer extraction*.

4.6.2 Integer Extraction

Integer extraction essentially removes the extra noise induced by ladder reductions when performing computations on encodings. In addition, as we shall see in Section 4.6.4, this step is enough to recover x_0 when an exact multiple is known, as is the case in the optimized variant proposed and implemented in [CLT15].

Integer Extraction of Level-κ Encodings of Zero.

In the remainder we say that an encoding at level k is small iff it is less than $X_0^{(k)}$ in absolute value. In particular, any ladder-reduced encoding is small.

Definition 22 (integer extraction of an encoding)**.** *Let $e \in \mathbb{Z}$, and write:*

$$\begin{aligned} e &= \sum_{i=1}^{n} r_i u_i + a x_0 \\ \text{with: } u_i &= \left(g_i z^{-k} (p_i^*)^{-1} \bmod p_i \right) p_i^* \text{ as in (4.3)} \\ r_i &\in \mathbb{Z} \cap (-p_i/2, p_i/2]. \end{aligned}$$

Note that r_i is uniquely defined as $r_i = e g_i^{-1} z^k \bmod p_i$, and $a = (e - \sum r_i u_i)/x_0$. Hence the following map is well-defined over $\mathbb{Z}$:

$$\phi : e \mapsto \sum_i r_i v_i + a v_0$$

$$\text{with: } v_0 = x_0 \boldsymbol{p}_{zt} \bmod N, \text{ and } \forall i > 0, v_i = u_i \boldsymbol{p}_{zt} \bmod N \text{ as in (4.4).}$$

We call $\phi(e)$ the integer extraction *of e.*

Remark. ϕ is defined over the integers, and not modulo N. Indeed the v_i's are seen as integers: recall from Section 4.2 that throughout this paper $x \bmod N$ denotes an integer in $\mathbb{Z} \cap (-N/2, N/2]$.

The point is that if e is a small encoding of zero at level κ, then $\phi(e) = e\boldsymbol{p}_{zt} \mod N$. In that case $\phi(e)$ matches the extraction in the sense of the ext procedure of Section 4.3.2 (more precisely ext returns the high-order bits of $\phi(e)$).

However we want to compute $\phi(e)$ even when e is larger. For this purpose, the crucial point is that ϕ is actually $\mathbb{Z}$-linear as long as for all encodings involved, the associated r_i's do not go over $p_i/2$, *i.e.* reduction modulo p_i does not interfere. More formally:

Lemma 15. *Let $e, a, r_1, \ldots, r_n \in \mathbb{Z}$ with $-p_i/2 < r_i \leq p_i/2$ such that $e = \sum r_i u_i + a x_0$ as in Definition 22. Define $e' = \sum r'_i u_i + a' x_0$ in the same manner. Let $k \in \mathbb{Z}$.*

1. *If $\forall i, -p_i/2 < r_i + r'_i \leq p_i/2$, then:* $\phi(e + e') = \phi(e) + \phi(e')$
2. *If $\forall i, -p_i/2 < k r_i \leq p_i/2$, then:* $\phi(ke) = k\phi(e)$

An important remark is that the conditions on the r_i's above are also required for the correctness of the scheme to hold. In other words, as long as we perform valid computations from the point of view of the multilinear map (*i.e.* there is no reduction of the r_i's modulo p_i, and correctness holds), then the $\mathbb{Z}$-linearity of ϕ also holds.

Using this observation, we can recursively compute the integer extraction of every ladder element $X_i^{(\kappa)}$. Indeed $\phi(X_0^{(\kappa)}) = X_0^{(\kappa)} \boldsymbol{p}_{zt} \bmod N$. Then assume we know $\phi(X_0^{(\kappa)}), \ldots, \phi(X_i^{(\kappa)})$ for some $i < \ell$. Reduce X_{i+1} by the previous elements of the ladder. We get:

$$\begin{aligned} & Y_{i+1} \triangleq X_{i+1}^{(\kappa)} - \alpha_i X_i^{(\kappa)} - \cdots - \alpha_0 X_0^{(\kappa)} \\ \text{with:} \quad & |Y_{i+1}| < |X_0^{(\kappa)}| \\ \text{whence:} \quad & \phi(X_{i+1}^{(\kappa)}) = \phi(Y_{i+1}) + \sum_{j \leq i} \alpha_j \phi(X_j^{(\kappa)}) \end{aligned}$$

Since $|Y_{i+1}| < |X_0|$ we can compute $\phi(Y_{i+1}) = Y_{i+1}\boldsymbol{p}_{zt} \bmod N$, and deduce $\phi(X_{i+1}^{(\kappa)})$.

In exactly the same manner, we can compute $\phi(e)$ for any valid level-κ encoding of zero, by first reducing via the ladder and then compensating using $\mathbb{Z}$-linearity. Here, by valid we mean of size up to X_ℓ, and such that the corresponding r_i's are within the limit imposed by the correctness of the multilinear map.

4.6.3 Integer Extraction of Products

In this section, we show that it is in fact possible to compensate ladder reductions at intermediate levels for any computation on encodings, *e.g.* compute $\phi(abc)$ for a three-way product abc. This extends integer extraction beyond products of two elements. Note however that this will not be necessary for our main attack.

Using Section 4.6.2, if a, b are two small encodings at levels s and $\kappa - s$ respectively, and b encodes zero, we know how to compute $\phi(ab)$, because the size of ab is at most that of X_ℓ.

We now consider larger products. Let $a_1, \ldots, a_m$, be small encodings at level $s_1, \ldots, s_m$, with $t_j \triangleq \sum_{i \leq j} s_i$, $t_m = \kappa$, and with a_m an encoding of zero. We would like to compute $\phi(a_1 \cdots a_m)$. Note that $a_1 \cdots a_m$ may be much larger than $X_\ell^{(\kappa)}$ in the absence of ladder reduction, so our previous technique is not enough.

Instead, a valid computation is to compute the product $\pi \triangleq a_1 \cdots a_m$ pairwise from the left, and reduce at each step. That is, let $\pi_1 \triangleq a_1$, and recursively define the ladder-reduced partial product $\pi_{i+1} \triangleq \pi_i a_{i+1} - \sum_j \alpha_j^{i+1} X_i^{(t_{i+1})} < X_0^{(t_{i+1})}$ for $i < m$. Thus $\pi_m < X_0^{(\kappa)}$ encodes the same element as π, and $\phi(\pi_m) = \pi_m \boldsymbol{p}_{zt} \bmod N$. In order to compute $\phi(\pi)$, observe:

$$\pi = \Big(\big((a_1 a_2 - \sum \alpha_i^{(2)} X_i^{(t_2)})\ldots\big)a_{m-1} - \sum \alpha_i^{m-1} X_i^{(t_{m-1})}\Big)a_m - \sum \alpha_i^{(m)} X_i^{(\kappa)} + \sum_{2 \leq k \leq m} \sum_i \alpha_i^{(k)} X_i^{(t_k)} a_{k+1} \cdots a_m$$

Hence:

$$\phi(a_1 \cdots a_m) = \phi(\pi_m) + \sum_{2 \leq k \leq m} \sum_i \alpha_i^{(k)} \phi(X_i^{(t_k)} a_{k+1} \cdots a_m)$$

In the above equation, $\phi(\pi_m)$ is known since π_m is small, so we are reducing the computation of a product π of m elements to a sum of products of $m-1$ elements, of the form $X_i^{(t_k)} a_{k+1} \cdots a_m$. As mentioned earlier we already know how to compute ϕ for products of 2 small elements, so by induction we are done.

To be more precise, the induction is carried out on the hypothesis: we know how to compute ϕ for products of up to m small encodings (with the last being an encoding of zero so that the overall product encodes zero). In order to apply the induction hypothesis on $X_i^{(t_k)} a_{k+1} \cdots a_m$, the term $X_i^{(t_k)}$ would need to be small, which is not the case. However it can be reduced by previous ladder elements, *i.e.* first compute $X_0^{(t_k)} a_{k+1} \cdots a_m$, then define $Y_1 = X_1^{(t_k)} - \alpha_0 X_0^{t_k} < X_0^{(t_k)}$, whence $\phi(X_1^{(t_k)} a_{k+1} \cdots a_m) = \phi(Y_1 a_{k+1} \cdots a_m) + \alpha_0 \phi(X_0^{t_k} a_{k+1} \cdots a_m)$, and so forth as in the previous section. Thus the induction goes through and we know how to compute $\phi(\pi)$.

All in all, while the above formalism may obfuscate the process somewhat, the idea is simple: ϕ is ($\mathbb{Z}$-)linear as long as we are performing valid computations from the point of view of the scheme. As a consequence every ladder reduction involved during a computation can be compensated for at its last stage, when the level-κ encoding is multiplied by the zero-testing parameter. The payback is that we will be able to ignore ladder reductions in the rest of the attack.

Remark. While the above reasoning is concerned with products only, ladder reductions coming from additions can also be compensated in a similar way. In summary, we can actually compute $\phi(F(a_1, \ldots, a_n))$ for *any* computation F on encodings a_i, as long as the computation is valid for the scheme itself (*i.e.* noises are within acceptable bounds for the correcntess of the multilinear map to hold).

A note on complexity. It may seem that computing $\phi(a_1 \cdots a_m)$ using the previous approach has a huge complexity, but actually most of the computation overlaps. In fact we only ever need to compute the $\phi(X_i^{(t_k)} a_{k+1} \cdots a_m)$'s for each i, k. Memorizing intermediate results yields a complexity in ℓm, where ℓ is the size of the longest ladder. The time required for each term is quite close to using the multi-party Diffie-Hellman scheme.

4.6.4 Recovering x_0 when an Exact Multiple is Known

The authors of [CLT15] propose an optimized version of their scheme, where a multiple qx_0 of x_0 is provided in the public parameters. The size of q is chosen such that qx_0 is about the same size as N. Ladders at levels below κ are no longer necessary: every encoding can be reduced modulo qx_0 without altering encoded values or increasing any noise. The ladder at level κ is still needed as a preliminary to zero-testing, however it does not need to go beyond qx_0, which makes it much smaller. In the end this optimization greatly reduces the size of the public key and speeds up computations, making the scheme much more practical (cf. Section 4.6.6).

In this scenario, note that qx_0 may be regarded as an encoding of 0 at level κ (and indeed every level). Moreover by construction it is small enough to be reduced by the ladder at level κ with a valid computation (*i.e.* with low enough noise for every intermediate encoding involved that the scheme operates as desired and zero-extraction is correct). As a direct consequence we have:

$$\phi(qx_0) = qv_0$$

and so we can recover q as $q = \gcd(qx_0, \phi(qx_0))$, and get $x_0 = qx_0/q$. This attack has been verified on the reference implementation, and recovers x_0 instantly.

Remark. qv_0 is larger than N by design, so that it cannot be computed simply as $qx_0\boldsymbol{p}_{zt} \bmod N$ due to modular reductions (cf. [CLT15, Section 3.4]). The point is that our computation of ϕ is over the integers and not modulo N.

4.6.5 Recovering x_0 in the General Case

We now return to the non-optimized version of the scheme, where no exact multiple of x_0 is provided in the public parameters.

The second step of our attack recovers x_0 using a matrix product similar to Cheon *et al.*'s (cf. Section 4.5), except we start with families of $n+1$ encodings rather than n. That is, assume that for some t we have $n+1$ level-t small encodings (a_i) of any value, and $n+1$ level-$(\kappa - t)$ small encodings (b_i) of zero. This is easily achievable for one-round multi-party Diffie-Hellman (cf. Section 4.3.2), e.g. choose $t = 1$, then pick $(n+1)$ level-1 encodings (a_i) of zero from the public parameters, and let $b_i = a'_i y^{\kappa-2}$ for a'_i another family of $(n+1)$ level-1 encodings of zero and y any level-1 encoding, where the product is ladder-reduced at each level. In other applications of the multilinear map, observe that ladder elements provide plenty of small encodings of zero, as each ladder element can be reduced by the elements below it to form a small encoding of zero. Thus the necessary conditions to perform both our attack to recover x_0, and the follow-up attack by Cheon *et al.* to recover other secret parameters once x_0 is known, are very lax. In this respect CLT15 is weaker than CLT13.

Let $a_{i,j} = a_i z \bmod p_j$, *i.e.* $a_{i,j}$ is the j-th value "$r_j g_j + m_j$" associated with a_i. Likewise for $i \leq n$, let $r_{i,j} = b_i z^{\kappa-1}/g_j \bmod p_j$, *i.e.* $r_{i,j}$ is the j-th value "r_j" associated with b_i (recall that b_i is an encoding of zero, so $m_j = 0$). Now compute:

$$\omega_{i,j} \triangleq \phi(a_i b_j).$$

If we look at the $\omega_{i,j}$'s modulo v_0 (which is unknown for now), everything behaves as in CLT13 since the new noise term av_0 disappears, and the ladder reduction at level κ is negated by the integer extraction procedure. Hence, similar to Section 4.5, we have:

$$\omega_{i,j} \bmod v_0 = \sum_k a_{i,k} r_{j,k} v_k \bmod v_0. \tag{4.6}$$

Again, equation (4.6) may be seen as a matrix product. Indeed, define Ω as the $(n+1)\times(n+1)$ integer matrix with entries $\omega_{i,j}$, let A be the $(n+1)\times n$ matrix with entries $a_{i,j}$, let R be the $(n+1)\times n$ matrix with entries $r_{i,j}$, and finally let V be the $n \times n$ diagonal matrix with diagonal entries v_i. Then (4.6) may be rewritten modulo v_0:

$$\Omega = A \cdot V \cdot R^{\mathrm{T}} \qquad \text{in } \mathbb{Z}_{v_0}.$$

Since A and R are $(n+1)\times n$ matrices, this implies that Ω is not full-rank when embedded into $\mathbb{Z}_{v_0}$. As a consequence v_0 divides $\det(\Omega)$, where the determinant is computed over the integers. Now we can build a new matrix Ω' in the same way using a different choice of b_i's, and recover v_0 as $v_0 = \gcd(\det(\Omega), \det(\Omega'))$. Finally we get $x_0 = v_0/\boldsymbol{p}_{zt} \bmod N$ (note that $N \gg x_0$ by construction).

The attack has been verified on the reference implementation with reduced parameters. In Section 4.7 we propose two different techniques to recover v_0 while avoiding the determinant computation step. Section 4.7.2 in particular proposes a probabilistic attack quite different from our main attack.

Remark. As pointed out above, Ω cannot be full-rank when embedded into $\mathbb{Z}_{v_0}$. Our attack also requires that it *is* full-rank over $\mathbb{Q}$ (whp). This holds because while Ω can be nicely decomposed as a product when viewed modulo v_0, the "remaining" part of Ω, that is $\Omega - (\Omega \bmod v_0)$ is the matrix of the terms av_0 for each $\omega_{i,j}$, and the value a does have the nice structure of $\omega_{i,j} \bmod v_0$. This is by design, since the noise a was precisely added in CLT15 in order to defeat the matrix product structure of Cheon *et al.*'s attack.

4.6.6 Attack Complexity

It is clear that the attack is polynomial, and asymptotically breaks the scheme. In this section we provide an estimate of its practical complexity. When an exact multiple of x_0 is known, the attack is instant as mentioned in Section 4.6.4, so we focus on the general case from Section 4.6.5.

In the general case, a ladder of encodings of size $\ell \approx \gamma$ is published at every level[30]. Using the scheme requires κ ladder reductions, *i.e.* $\kappa\ell$ additions of integers of size γ. Since there are κ users, this means the total computation incurred by using the scheme is close to $\kappa^2\gamma^2$. For the smallest 52-bit instance, this is already $\approx 2^{46}$. Thus using the scheme a hundred times is above

[30] As the level increases, it is possible to slightly reduce the size of the ladder. Indeed the acceptable level of noise increases with each level, up to ρ_f at level κ. As a consequence it is possible to leave a small gap between ladder elements as the level increases. For instance if the base level of noise is 2ρ for ladder elements, then at level κ it is possible to leave a gap of roughly $\rho_f - 2\rho - \log \ell$ bits between ladder elements. We disregard this effect, although it slighly improves our complexity.

the security parameter. This highlights the importance of the optimization based on publishing qx_0, which makes the scheme much more practical. More importantly for our current purpose, this makes it hard to propose an attack below the security parameters.

As a result, what we propose in terms of complexity evaluation is the following. For computations that compare directly to using the multilinear scheme, we will tally the complexity as the number of operations equivalent to using the scheme, in addition to the bit complexity. For unrelated operations, we will count the number of bit operations as usual.

There are two steps worth considering from a complexity point of view: computing Ω and computing its determinant. In practice both steps happen to have comparable complexity. Computing the final gcd is negligible in comparison using a subquadratic algorithm [Möl08], which is practical for our parameter size.

Computing Ω.

As a precomputation, in order to compute ϕ, the integer extraction of ladder elements at level κ needs to be computed. This requires ℓ integer extractions, where $\ell \leq \gamma$. Computing Ω itself requires $(n+1)^2$ integer extractions of a single product. Each integer extraction requires 1 multiplication, and 2ℓ additions (as well as ℓ multiplications by small scalars). For comparison, using the multilinear scheme for one user requires 1 multiplication and ℓ additions on integers of similar size. Thus overall computing Ω costs about $\gamma + n^2$ times as much as simply *using* the multilinear scheme. For the 52-bit instance proposed in [CLT15] for instance, this means that if it is practical to use the scheme about a million times, then it is practical to compute Ω. Here by using the scheme we mean one (rather than κ^2) ladder reduction, so the bit complexity is $\mathcal{O}(\gamma^3 + n^2\gamma^2)$.

Computing the Determinant.

Let n denote the size of a matrix Ω (it is $(n+1)$ in our case but we will disregard this), and β the number of bits of its largest entry. When computing the determinant of an integer matrix, one has to carefully control the size of the integers appearing in intermediate computations. It is generally possible to ensure that these integers do not grow past the size of the determinant. Using Hadamard's bound this size can be upper bounded as $\log(\det(\Omega)) \leq n(\beta + \frac{1}{2}\log n)$, which can be approximated to $n\beta$ in our case, since β is much larger than n.[31]

As a result, computing the determinant using "naive" methods requires $\mathcal{O}(n^3)$ operations on integers of size up to $n\beta$, which results in a complexity $\widetilde{\mathcal{O}}(n^4\beta)$ using fast integer multiplication (but slow matrix multiplication). The asymptotic complexity is known to be $\widetilde{\mathcal{O}}(n^\omega\beta)$ [Sto05]; however we are interested in the complexity of practical algorithms. Computing the determinant can be reduced to solving the linear system associated with Ω with a random target vector: indeed the determinant can then be recovered as the least common denominator of the (rational) solution vector[32]. In this context the fastest algorithms use p-adic lifting [Dix82], and an up-to-date analysis using fast arithmetic in [MS04] gives a complexity $\mathcal{O}(n^3\beta\log^2\beta\log\log\beta)$ (with $\log n = o(\beta)$).[33]

For the concrete instantiations of one-round multipartite Diffie-Hellman implemented in [CLT15], this yields the following complexities:

[31] This situation is fairly unusual, and in the literature the opposite is commonly assumed; algorithms are often optimized for large n rather than large β.

[32] In general extra factors may appear, but this is not relevant for us.

[33] This assumes a multitape Turing machine model, which is somewhat less powerful than a real computer.

Security parameter:	52	62	72	80
Building Ω:	2^{60}	2^{66}	2^{74}	2^{82}
Determinant:	2^{57}	2^{66}	2^{74}	2^{81}

Thus, beside being polynomial, the attack is actually coming very close to the security parameter as it increases to 80 bits.[34]

4.7 Recovering x_0 without Computing a Determinant

In this section we describe two alternate approaches to recovering x_0 in the general case. The first approach is very similar to our main attack, and simply replaces the determinant computation with solving a rational system. The complexity is essentially equivalent.

The second approach is quite different. It relies on finding and exploiting divisors of v_0. The attack is probabilistic in the sense that it works on roughly 90% of instances: the probabilistic aspect is purely dependent on instance generation.

4.7.1 First approach: Rational System Solving

As mentioned earlier, this approach is very similar to our main attack. We begin by computing matrices Ω, Ω' exactly as in Section 4.6.5. From there recovering v_0 can be roughly modeled as the following problem:

Problem 2. *Let n, β, v be positive integers, with $v \ll \beta$. Let A, B be two integer matrices chosen uniformly at random among $(n+1) \times (n+1)$ matrices with the following properties:*

1. *All entries are in $\mathbb{Z} \cap [-2^\beta, 2^\beta]$*
2. *A, B are full-rank.*
3. *A, B are rank n when embedded into $\mathbb{Z}_v$.*

The problem is to find (the largest possible) v given A and B.

Remark. As it it stated above, the problem provides two matrices A, B satisfying the three conditions. However the problem makes sense even with just one such matrix.

In our main attack, we solve this problem by observing $v | \det(A)$, so $v = \gcd(\det(A), \det(B))$. However it is not clear that this is the best approach. We think that this is an interesting problem and there may be more efficient solutions.

In this section we will simply sketch another solution based on solving a rational system, whose complexity is essentially the same as computing a determinant in our main attack.

Decompose A as a block matrix in the following way:

$$A = \left[\begin{array}{c|c} A' & b \\ \hline c & d \end{array}\right]$$

[34] We may note in passing that in a random-access or log-RAM computing model [Für14], which is more realistic than the multitape model, the estimated determinant complexity would already be slightly lower than the security parameter.

In this representation, A' is the submatrix containing the first n rows and columns of A, and b, c, d are respectively $n \times 1$, $1 \times n$ and 1×1 matrices. Whp A' is invertible both over $\mathbb{Z}_v$ and over $\mathbb{Q}$. Then over $\mathbb{Z}_v$ we have $c(A')^{-1}b = d$ since A is not full-rank. Hence if we compute $d' = c(A')^{-1}b$ over the rationals, then $d - d' \bmod v = 0$. If we write $d' = p/q$ with p, q coprime integers, this means that $v|qd - p$. We can repeat this process with B (or using another decomposition of A), and recover v as the gcd of $qd - p$ computed as above for each matrix.

The complexity of this method is essentially the same as that of a determinant, because it amounts to solving a rational system, namely $A'x = b$, which is also the bottleneck for the most efficient determinant algorithms. Thus the real challenge would be to solve Problem 2 without somehow resorting to solving a rational system with parameters close to those of A.

4.7.2 Second Approach: Probabilistic Algorithm

If we try to apply Cheon *et al.*'s attack directly to CLT15, there are two obstacles: ladder reductions, and the noise term av_0. Using integer extraction, the effect of ladder reductions is negated. Then, if v_0 is somehow recovered, Cheon *et al.*'s attack can be applied modulo v_0, since this makes the term av_0 disappear. However the same is true for any divisor π of v_0.

Based on this idea our second approach proceeds in two steps: first, we find a prime $\pi|v_0$. Second, we exploit π to recover all secret parameters using a variant of Cheon *et al.*'s attack, modulo increasing powers of π.

Step 1: Finding π.

Observe that v_0 has no reason to be prime. In fact, it is defined as a large sum of various parameters with no common denominator, and heuristically we expect it to behave like a uniformly random number modulo π for any sufficiently small π (*e.g.* $\pi \ll 2^\gamma$). In particular, choose some bound b and define:

$$B(b) \triangleq \prod_{p \in \mathbb{P}, p < b} p$$
$$p(b) \triangleq \Pr[x \leftarrow^{\$} \mathbb{Z} \cap [0, B(b)] : \exists p \in \mathbb{P}, p < b, p|x]$$

where $\mathbb{P}$ denotes the set of prime numbers. Then as long as $B(b) \ll v_0$ we expect that the probability that some prime $\pi < b$ divides v_0 is very close to $p(b)$.

If we take *e.g.* $b = 100$ then already $p(b) \approx 0.88$. In general Merten's third theorem provides an asymptotic formula which is quite tight for our purpose:

$$1 - p(b) \sim \frac{e^{-\gamma}}{\ln b}$$

where $\gamma \approx 0.58$ is Euler's constant.

We could simply try to guess π and carry out the rest of the attack. However a better method in order to find $\pi < b, \pi|v_0$, is to compute a matrix Ω and its determinant as in Section 4.6.5, except all computations are carried out modulo $B(b)$. Thus determinant computation in particular is much cheaper (for a reasonable choice of b). Once we know $\det(\Omega) \bmod B(b)$, we can check whether $p|\det(\Omega)$ for each prime $p < b$. If so then with high probability $p|v_0$. False positives can be eliminated by repeating this process a few times. The overall probability of success is $p(b)$.

Step 2: Rationale.

Before presenting the attack itself, we highlight the main ideas. At this point we know some small $\pi|v_0$. This means we can reduce every equation modulo π and perform Cheon *et al.*'s attack. Thus for some level-0 encodings $c_i \triangleq \mathsf{CRT}_{(p_j)}(c_{i,j})$ we can recover the $c_{i,j}$'s modulo π. We want to recover the $c_{i,j}$'s in their entirety.

However we are only able to recover secret information modulo π. If we consider, say, the $c_{i,j}$'s as integers in base π, a natural idea in order to recover the next digit of the $c_{i,j}$'s is to somehow divide them by π, in order to shift the second digit into the first position. Thus we create a linear combination $d \triangleq \mathsf{CRT}_{(p_j)}(d_j)$ of the encodings c_i such that $d_j \bmod \pi = 0$. Then we divide d by π over the integers and apply Cheon *et al.*'s attack. In the end we find some linear information on the $c_{i,j}$'s modulo π^2.

We could repeat this process for increasing powers of π. However we only recover information on some linear combinations of the c_i's, and the size of these linear combinations increases as we go on. Moreover we would lose n degrees of freedom at each step, which implies we would need a large number of c_i's at the start, which the scheme may not provide.

Instead, we use a second idea, which is to recover information modulo π not directly on the c_i's, but on their pairwise products. This gives us much more information, which makes it possible to recover information modulo π^m on the $c_{i,j}$'s themselves, and not just a subspace of codimension n spanned by some linear combinations. As a result the number of c_i's required does not grow with the number of induction steps.

We now move on to the actual attack.

Step 2.1: Computing the Matrices $\Omega_{x,y}$.

Let $\alpha = \lceil \sqrt{2n+1} \rceil$, and pick:

$$\begin{array}{ll} a_i \triangleq \mathsf{CRT}_{(p_j)}(a_{i,j}/z^t) & : \text{a sequence of } n \text{ encodings at some level } t. \\ b_i \triangleq \mathsf{CRT}_{(p_j)}(b_{i,j}g_j/z^{\kappa-t}) & : \text{a sequence of } n \text{ encodings of zero at level } \kappa - t. \\ c_x \triangleq \mathsf{CRT}_{(p_j)}(c_{x,j}) & : \text{a sequence of } \alpha \text{ encodings at level 0.} \end{array}$$

Now define:

$$\begin{array}{ll} \Omega & : \text{the } n \times n \text{ integer matrix with entries } \phi(a_i b_j). \\ \Omega_{x,y} & : \text{the } n \times n \text{ integer matrix with entries } \phi(c_x c_y a_i b_j). \\ \Omega'_{x,y} & \triangleq \Omega_{x,y}\Omega^{-1} \text{ over the rationals.} \end{array}$$

That is, we compute the matrix from Cheon *et al.*'s attack for all encodings $c_x c_y$. When $\Omega'_{x,y}$ is viewed modulo π, its eigenvalues are the $c_{x,i}c_{y,i}$'s modulo π as in Section 4.5 (this assumes Ω is invertible modulo π, which holds whp). In particular this enables us to recover the $c_{x,i}$'s modulo π.

Step 2.2: Main Induction.

Assume we know all $c_{x,i}$'s modulo π^m for some $m > 0$. We want to recover the $c_{x,i}$'s modulo π^{m+1}. To this end, decompose $c_{x,i}$ as:

$$c_{x,i} = c'_{x,i}\pi^m + c''_{x,i}, \text{ with } |c''_{x,i}| < \pi^m$$

So far we know $c''_{x,i}$, and we are looking for $c'_{x,i} \bmod \pi$.

Because there are $\alpha^2 > 2n+1$ encodings $c_x c_y$, we can create $n+1$ linear combinations:

$$d_k \triangleq \sum \lambda_{x,y}^{(k)} c_x c_y, \text{ with } \lambda_{x,y}^{(k)} \in \mathbb{Z} \cap (-\pi^m/2, \pi^m/2], \text{ for } k \leq n+1$$

such that if we let $d_{k,i} \triangleq d_k \bmod p_i = \sum \lambda_{x,y}^{(k)} c_{x,i} c_{y,i}$ be the i-th component of the encoding d_k, then $d_{k,i} \bmod \pi^m = 0$.

In addition, spending one more degree of freedom, we can create n linear combinations of the d_k's that are zero modulo π^m. Without loss of generality, we assume $d_k \bmod \pi^m = 0$ for $k \leq n$. We now restrict our attention to the first n d_k's.

Let $d'_k \triangleq d_k/\pi^m$, where the division is exact over the integers. Then $d'_{k,i} \triangleq d'_k \bmod p_i$, when regarded modulo π^{m+1}, satisfies:

$$\begin{aligned} d'_{k,i} &= \frac{1}{\pi^m} \sum \lambda_{x,y}^{(k)} c_{x,i} c_{y,i} \\ &= \frac{1}{\pi^m} \sum \lambda_{x,y}^{(k)} (c'_{x,i}\pi^m + c''_{x,i})(c'_{y,i}\pi^m + c''_{y,i}) \\ &= \sum \lambda_{x,y}^{(k)} (c'_{x,i} c''_{y,i} + c''_{x,i} c'_{y,i}) + \frac{1}{\pi^m} \sum \lambda_{x,y}^{(k)} c''_{x,i} c''_{y,i} \end{aligned}$$

where the last division is exact by construction of the $\lambda_{x,y}^{(k)}$'s.

Recall that the $c''_{x,i}$'s are known by induction hypothesis, so what knowledge of the $d'_{k,i}$'s really gives us is n linear equations over the $c'_{x,i}$'s (and $c'_{y,i}$'s). Hence whp knowing the $d'_{k,i}$'s modulo π allows us to recover the $c'_{x,i}$'s modulo π we are looking for.

Thus the induction comes down to recovering the $d'_{k,i}$'s modulo π. Observe that the d'_k's are valid encodings at level 0. Moreover if we apply Cheon *et al.*'s attack modulo π on the d'_k's, then due to the $\mathbb{Z}$-linearity of ϕ, the associated matrix will be $\frac{1}{\pi^m} \sum \lambda_{x,y}^{(k)} \Omega'_{x,y}$. Thus in the end the $d'_{k,i}$'s modulo π are the eigenvalues of $\frac{1}{\pi^m} \sum \lambda_{x,y}^{(k)} \Omega'_{x,y}$ modulo π,[35] and we are done.

Final Step.

Once $\pi^m > c_{x,i}$, *i.e.* $m > 2\rho/\log \pi$, we know $c_{x,i}$. Then $p_i | c_x - c_{x,i}$ and we can recover p_i *e.g.* as $\gcd(c_1 - c_{1,i}, c_2 - c_{2,i})$. From there all other secret parameters of the scheme are easily computed.

Noise Growth.

In its application to multipartite Diffie-Hellman, the multilinear map needs to support multiplication of one level-0 encoding with κ level-1 encodings. The noise of the final encoding at level κ (by which we mean the size of its value modulo p_i) will be close to $2\rho(\kappa+1)$. Meanwhile our attack induces a noise at level κ equivalent to multiplying *three* level-0 encodings: namely c_x, c_y and multiplication by $\lambda_{x,y}^{(k)}$; with an encoding at level t and another at level $\kappa - t$. By using ladder elements as encodings, the resulting noise will be close to $2\rho \cdot 5$. Thus the attack is applicable for $\kappa \geq 4$. This assumes that encodings with the base level of noise 2ρ are available at each level: this is provided by ladder elements.

[35] The matrix $M = \frac{1}{\pi^m} \sum \lambda_{x,y}^{(k)} \Omega'_{x,y}$ may seem ill-defined modulo π due to the division by π^m. However the division is over the rationals before embedding into $\mathbb{Z}_\pi$, and it is exact on the numerator of each entry. Indeed $M = (\frac{1}{\pi^m} \sum \lambda_{x,y}^{(k)} \Omega_{x,y}) \Omega^{-1}$, and the sum on the left-hand side yields an integer matrix, where each element is an image by ϕ of some product $d'_k a_i b_j$, by construction of the λ's and by $\mathbb{Z}$-linearity. Thus all we ever need is for the single matrix Ω to be invertible modulo π.

Complexity.

It is clear that the attack is polynomial. We provide a quick look at its complexity. The main steps of the attack are as follows. We can choose *e.g.* $b = 100$ as this already ensures a high probability of success.

1. **Step 1: Finding π.** This step is similar to our main attack, except the determinant computation is much cheaper, as it is performed modulo $B(b)$. If we let $\beta \triangleq \log B(b)$ then the complexity given in Section 4.6.6 yields $\mathcal{O}(n^3\beta \log^2 \beta \log\log \beta)$. For $b = 100$, $\beta \approx 121$, and this step will be negligible relative to the next ones.

2. **Step 2.1: Computing the Matrices $\Omega_{x,y}$.** This requires $\alpha^2 n^2 \approx n^3$ calls to ϕ for a four-way product. This is more or less equivalent to using the scheme $n^3\gamma$ times. For the smallest instance implemented in [CLT15], $n^3\gamma \approx 2^{47}$.

3. **Step 2.2: Main Induction:** We repeat the induction $m \approx 2\rho/\log(\pi)$ times. Each induction step involves:

   (a) Inverting a matrix of dimension n with entries of size up to 2ρ. This can be evaluated to $\mathcal{O}(n^3\rho \log \rho \log\log \rho)$ bit operations.

   (b) Recovering the eigenvalues of (a linear combination over $\mathbb{Q}$ of) $\Omega'_{x,y}$'s modulo π. Since π is quite small, this essentially costs n^3 operations, which is negligible (moreover the eigenvectors are always the same, and can be recovered once for a further speedup).

Conclusion

In this thesis we have focused mostly on cryptanalyses of some recent cryptographic schemes. Part of the goal of cryptanalysis is to identify the most relevant attacks against a given type of cryptographic scheme or problem. In this respect, a large part of our work pertains to schemes where the picture of the most relevant attacks still appears incomplete, or at least not fully delineated. As such many open questions remain.

In Chapter 1, we have presented a unified cryptanalysis of block ciphers Robin, iSCREAM and Zorro. Our analysis was based on self-similarity and invariant subspace properties. We also provided generic tools for detecting this type of attack.

Since our work, invariant subspaces have made a few more appearances [GJN+15, Røn16, GRR16]. In the first two cases [GJN+15, Røn16], invariant subspaces can be regarded as a simple oversight by the designers, and could be easily prevented by tweaking round constants. This would suggest a view of invariant subspaces similar to slide attacks: devastating in some cases, but easy to prevent for designers who are aware of this type of attack. However [GRR16] presents a generalization of invariant subspaces applied to AES, and drops the requirement that the key schedule be trivial. This hints that the future of invariant subspaces may be more complex, and more work is needed in that direction. My two coauthors from [LMR15] will certainly continue to investigate. It is also interesting to note that [GRR16] has a strong integral attack flavor. Together with recent work on the division property such as [Tod15], this suggests that new types of integral attacks may still be waiting to be uncovered.

Meanwhile there is no other recorded occurence of commuting maps, such as the ones that appear in Robin and Zorro. It is an open question whether weaker (*e.g.* probabilistic) variants of these properties could be useful for the analysis of other ciphers.

In Chapter 2 we proposed a structural cryptanalysis of ASASA. The attack was applied to both black-box and white-box schemes, including a multivariate trapdoor permutation. A very interesting follow-up work by Biryukov and Khovratovich shows that our structural attack on ASASA can be extended to longer structures, even SASASASAS for some parameters [BK15]. The main obstacle is the degree of the overall function, which is bounded using results by Boura and Canteaut on the degree of composite functions [BC13]. On the other hand no technique is known as soon as the ASASA... construction reaches full degree. A distinguisher that would be successful on this type of construction beyond full degree would be very interesting, as it might be usable to attack SPN block ciphers. Another interesting new avenue of applications for structural cryptanalysis is a recent line of work that has looked at reverse-engineering S-boxes [BPU16], or other permutations whose structure may not be fully understood [PUB16].

Going back to white-box cryptography, which was the motivation behind ASASA [BBK14], in Chapter 3, we build two efficient white-box schemes with provable security guarantees. This follows several works on incompressible white-box constructions [DLPR13, BBK14, BI15], which either relied on asymmetric primitives or offered no provable security. So far, previous symmetric white-box constructions [BBK14, BI15] aimed to resist standard symmetric attacks, as well

as structural attacks in the case of [BBK14]. However when approached from the provable security angle (as is the case in our work), new parallells with other cryptographic models appear. This includes local extractors [Vad04] and intrusion resilience [Dzi06]. Most notably, our strong model of incompressibility is quite close to a very recent independent work on big-key encryption [BKR16]. In this respect, at least from a theoretical perspective, the problem of white-box incompressibility seems less isolated, and may be regarded as one of the more practical ends of a larger spectrum of models. Techniques from [BKR16] may be especially helpful for future work.

Finally, in Chapter 4, we described a polynomial attack on a multilinear map candidate construction. The current picture of multilinear map is as follows. Three major schemes have been proposed [GGH13a, CLT13, GGH15]. Two variants have been proposed for the CLT scheme [CLT13, CLT15], however our contribution is precisely to show that the CLT15 variant can be discarded entirely. Of the other three schemes, every one has been broken when used to instantiate non-interactive multipartite key exchange [HJ15, CHL$^+$15, CLLT16], and GGH13 has also been broken for some constructions of indistinguishability obfuscation (iO) [MSZ16a]. While new constructions of multilinear maps may still be possible, all these attacks appear to have shaken the faith of the community in building a generic multilinear map (or graded encoding scheme). Instead, recent work adapts and restricts the usage of existing multilinear map constructions, in such a way that a large class of attack becomes impossible. This class is designed to include all known attacks [GMS16, MSZ16b]. The point is that even in the presence of these restrictions, multilinear maps can still be used to achieve indistinguishability obfuscation. These constructions are dedicated to iO, and the status of other constructions relying on multilinear maps is unknown. On the cryptanalytic side, important open questions include whether CLT13 is secure when used to build iO, *e.g.* using [Zim15], and whether the aforementioned recent construtions of iO that structurally avoid known attacks are secure.

Tables

1 Robin and iSCREAM S-Box

	*0	*1	*2	*3	*4	*5	*6	*7	*8	*9	*a	*b	*c	*d	*e	*f
0*	00	85	65	d2	5b	ff	7a	ce	4d	e2	2c	36	92	15	bd	ad
1*	57	f3	37	2d	88	0d	ac	bc	18	9f	7e	ca	41	ee	61	d6
2*	59	ec	78	d4	47	f9	26	a3	90	8b	bf	30	0a	13	6f	c0
3*	2b	ae	91	8a	d8	74	0b	12	cc	63	fd	43	b2	3d	e8	5d
4*	b6	1c	83	3b	c8	45	9d	24	52	dd	e4	f4	ab	08	77	6d
5*	f5	e5	48	c5	6c	76	ba	10	99	20	a7	04	87	3f	d0	5f
6*	a5	1e	9b	39	b0	02	ea	67	c6	df	71	f6	54	4f	8d	2e
7*	e7	6a	c7	de	35	97	55	4e	22	81	06	b4	7c	fb	1a	a1
8*	d5	79	fc	42	84	01	e9	5c	14	93	33	29	c1	6e	a8	b8
9*	28	32	0c	89	b9	a9	d9	75	ed	58	cd	62	f8	46	9e	19
a*	cb	7f	a2	27	d7	60	fe	5a	8e	95	e3	4c	16	0f	31	be
b*	64	d3	3c	b3	7b	cf	40	ef	8f	94	56	f2	17	0e	af	2a
c*	2f	8c	f1	e1	dc	53	68	72	44	c9	1b	a0	38	9a	07	b5
d*	5e	d1	03	b1	23	80	1f	a4	34	96	e0	f0	c4	49	73	69
e*	da	c3	09	aa	4a	51	f7	70	3e	86	66	eb	21	98	1d	b7
f*	db	c2	bb	11	4b	50	6b	e6	9c	25	fa	7d	82	3a	a6	05

2 Well-Behaved Affine Spaces for the Robin and iSCREAM S-Box

Only spaces whose direction contains 1 are listed.

Values in $u + A$								Basis of A			Values in $v + B = S(u + A)$								Basis of B			$\dim(A + B)$
00	01	26	27	84	85	a2	a3	01	26	84	00	01	26	27	84	85	a2	a3	01	26	84	3
18	19	7c	7d	9e	9f	fa	fb	01	64	86	18	19	7c	7d	9e	9f	fa	fb	01	64	86	3
28	29	32	33	8a	8b	90	91	01	1a	a2	90	91	8a	8b	32	33	28	29	01	1a	a2	3
3c	3d	5e	5f	b2	b3	d0	d1	01	62	8e	b2	b3	d0	d1	3c	3d	5e	5f	01	62	8e	3
44	45	66	67	c8	c9	ea	eb	01	22	8c	c8	c9	ea	eb	44	45	66	67	01	22	8c	3
4e	4f	54	55	6c	6d	76	77	01	1a	22	77	76	6d	6c	55	54	4f	4e	01	1a	22	3
28	29	32	33	6c	6d	76	77	01	1a	44	90	91	8a	8b	54	55	4e	4f	01	1a	c4	4
28	29	32	33	4e	4f	54	55	01	1a	66	90	91	8a	8b	76	77	6c	6d	01	1a	e6	4
2e	2f	38	39	8c	8d	9a	9b	01	16	a2	6f	6e	63	62	cd	cc	c1	c0	01	0c	a2	4
08	09	2e	2f	8c	8d	aa	ab	01	26	84	4d	4c	6f	6e	c1	c0	e3	e2	01	22	8c	5
08	09	38	39	9a	9b	aa	ab	01	30	92	4d	4c	63	62	cd	cc	e3	e2	01	2e	80	5
0a	0b	12	13	c6	c7	de	df	01	18	cc	2c	2d	36	37	68	69	72	73	01	1a	44	5
0e	0f	16	17	c2	c3	da	db	01	18	cc	bd	bc	ad	ac	f1	f0	e1	e0	01	10	4c	5
20	21	3e	3f	86	87	98	99	01	1e	a6	59	58	5d	5c	e9	e8	ed	ec	01	04	b0	5
22	23	34	35	80	81	96	97	01	16	a2	78	79	74	75	d8	d9	d4	d5	01	0c	a0	5
24	25	3a	3b	82	83	9c	9d	01	1e	a6	47	46	43	42	fd	fc	f9	f8	01	04	ba	5
4a	4b	50	51	8e	8f	94	95	01	1a	c4	e4	e5	f4	f5	a8	a9	b8	b9	01	10	4c	5

3 Commuting Linear Map and Invariant Subspace for Zorro

The commuting linear map M is represented as a 16×16 matrix over $\mathbb{F}_{2^8}$, using the AES representation of $\mathbb{F}_{2^8}$ as $\mathbb{F}_2[x]/(x^8 + x^4 + x^3 + x + 1)$.

$$\begin{bmatrix}
0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 34 & 101 & 35 & 101 & 50 & 249 & 50 & 249 & 249 & 116 & 249 & 116 \\
0 & 0 & 0 & 0 & 101 & 35 & 101 & 34 & 249 & 50 & 249 & 50 & 116 & 249 & 116 & 249 \\
0 & 0 & 0 & 0 & 35 & 101 & 34 & 101 & 50 & 249 & 50 & 249 & 249 & 116 & 249 & 116 \\
0 & 0 & 0 & 0 & 101 & 34 & 101 & 35 & 249 & 50 & 249 & 50 & 116 & 249 & 116 & 249 \\
0 & 0 & 0 & 0 & 17 & 86 & 17 & 86 & 1 & 0 & 0 & 0 & 249 & 50 & 249 & 50 \\
0 & 0 & 0 & 0 & 86 & 17 & 86 & 17 & 0 & 0 & 0 & 1 & 50 & 249 & 50 & 249 \\
0 & 0 & 0 & 0 & 17 & 86 & 17 & 86 & 0 & 0 & 1 & 0 & 249 & 50 & 249 & 50 \\
0 & 0 & 0 & 0 & 86 & 17 & 86 & 17 & 0 & 1 & 0 & 0 & 50 & 249 & 50 & 249 \\
0 & 0 & 0 & 0 & 51 & 190 & 51 & 190 & 86 & 17 & 86 & 17 & 35 & 101 & 34 & 101 \\
0 & 0 & 0 & 0 & 190 & 51 & 190 & 51 & 17 & 86 & 17 & 86 & 101 & 34 & 101 & 35 \\
0 & 0 & 0 & 0 & 51 & 190 & 51 & 190 & 86 & 17 & 86 & 17 & 34 & 101 & 35 & 101 \\
0 & 0 & 0 & 0 & 190 & 51 & 190 & 51 & 17 & 86 & 17 & 86 & 101 & 35 & 101 & 34
\end{bmatrix}$$

The invariant subspace $\ker(M + Id)$ is generated by the following 12 row vectors, in the same representation.

```
(1   0   1   0   0   0   0   0   0   0   0   0   0   0   0   0)
(0   1   0   0   0   0   0   0   0   0   0   0   0   0   0   0)
(0   0   0   1   0   0   0   0   0   0   0   0   0   0   0   0)
(0   0   0   0   1   0   0   0   0   0   0  38   0   0 159   0)
(0   0   0   0   0   1   0   0   0   0   0   1   0   0   0   3)
(0   0   0   0   0   0   1   0   0   0   0  38   0   0 159   0)
(0   0   0   0   0   0   0   1   0   0   0   1   0   0   0   3)
(0   0   0   0   0   0   0   0   1   0   0  79   0   0  38   1)
(0   0   0   0   0   0   0   0   0   1   0   1   0   0   0   0)
(0   0   0   0   0   0   0   0   0   0   1  79   0   0  38   1)
(0   0   0   0   0   0   0   0   0   0   0   0   1   0   1   0)
(0   0   0   0   0   0   0   0   0   0   0   0   0   1   0   1)
```

Bibliography

[ADK+14] Martin R. Albrecht, Benedikt Driessen, Elif Bilge Kavun, Gregor Leander, Christof Paar, and Tolga Yalçin. Block ciphers - focus on the linear layer (feat. PRIDE). In *Advances in Cryptology - CRYPTO 2014 - 34th Annual Cryptology Conference, Santa Barbara, CA, USA, August 17-21, 2014, Proceedings, Part I*, pages 57–76, 2014.

[ADW09] Joël Alwen, Yevgeniy Dodis, and Daniel Wichs. Survey: Leakage resilience and the bounded retrieval model. In *Information Theoretic Security*, pages 1–18. Springer, 2009.

[AFF+14] Martin R. Albrecht, Jean-Charles Faugère, Robert Fitzpatrick, Ludovic Perret, Yosuke Todo, and Keita Xagawa. Practical Cryptanalysis of a Public-Key Encryption Scheme Based on New Multivariate Quadratic Assumptions. In Hugo Krawczyk, editor, *Public-Key Cryptography – PKC 2014*, volume 8383 of *Lecture Notes in Computer Science*, pages 446–464. Springer Berlin Heidelberg, 2014.

[ANWOW13] Jean-Philippe Aumasson, Samuel Neves, Zooko Wilcox-O'Hearn, and Christian Winnerlein. Blake2: simpler, smaller, fast as md5. In *Applied Cryptography and Network Security*, pages 119–135. Springer, 2013.

[ARM09] ARM. Security technology building a secure system using TrustZone technology. White paper, available at `infocenter.arm.com/help/topic/com.arm.doc.prd29-genc-009492c/`, 2009.

[BB02] Elad Barkan and Eli Biham. In how many ways can you write rijndael? In Yuliang Zheng, editor, *Advances in Cryptology — ASIACRYPT 2002*, volume 2501 of *Lecture Notes in Computer Science*, pages 160–175. Springer Berlin Heidelberg, 2002.

[BBK14] Alex Biryukov, Charles Bouillaguet, and Dmitry Khovratovich. Cryptographic Schemes Based on the ASASA Structure: Black-Box, White-Box, and Public-Key. In Palash Sarkar and Tetsu Iwata, editors, *Advances in Cryptology–ASIACRYPT 2014*, volume 8873 of *Lecture Notes in Computer Science*, pages 63–84. Springer Berlin Heidelberg, 2014. Full version: `http://eprint.iacr.org/2014/474`.

[BC13] Christina Boura and Anne Canteaut. On the influence of the algebraic degree of on the algebraic degree of. *Information Theory, IEEE Transactions on*, 59(1):691–702, 2013.

[BCG+12] Julia Borghoff, Anne Canteaut, Tim Güneysu, Elif Bilge Kavun, Miroslav Knežević, Lars R. Knudsen, Gregor Leander, Ventzislav Nikov, Christof Paar, Christian

Rechberger, Peter Rombouts, Søren S. Thomsen, and Tolga Yalçın. PRINCE - A Low-Latency Block Cipher for Pervasive Computing Applications - Extended Abstract. In *ASIACRYPT*, volume 7658 of *LNCS*, pages 208–225. Springer, 2012.

[BDK+11] Boaz Barak, Yevgeniy Dodis, Hugo Krawczyk, Olivier Pereira, Krzysztof Pietrzak, François-Xavier Standaert, and Yu Yu. Leftover hash lemma, revisited. In *Advances in Cryptology–CRYPTO 2011*, pages 1–20. Springer, 2011.

[BDLF10] Charles Bouillaguet, Orr Dunkelman, Gaëtan Leurent, and Pierre-Alain Fouque. Another look at complementation properties. In Seokhie Hong and Tetsu Iwata, editors, *Fast Software Encryption*, volume 6147 of *Lecture Notes in Computer Science*, pages 347–364. Springer Berlin Heidelberg, 2010.

[BF01] Dan Boneh and Matt Franklin. Identity-based encryption from the Weil pairing. In *Advances in Cryptology–CRYPTO 2001*, pages 213–229. Springer, 2001.

[BFP11] L. Bettale, J.-C. Faugère, and L. Perret. Cryptanalysis of Multivariate and Odd-Characteristic HFE Variants. In *Public Key Cryptography - PKC 2011*, volume 6571, pages 441–458. Springer-Verlag, 2011.

[BGEC04] Olivier Billet, Henri Gilbert, and Charaf Ech-Chatbi. Cryptanalysis of a white box aes implementation. In *Selected Areas in Cryptography*, pages 227–240. Springer, 2004.

[BGI+01] Boaz Barak, Oded Goldreich, Rusell Impagliazzo, Steven Rudich, Amit Sahai, Salil Vadhan, and Ke Yang. On the (im)possibility of obfuscating programs. In *Advances in cryptology–CRYPTO 2001*, pages 1–18. Springer, 2001.

[Bha14] Arnab Bhattacharyya. Polynomial decompositions in polynomial time. In Andreas S. Schulz and Dorothea Wagner, editors, *Algorithms - ESA 2014 - 22th Annual European Symposium, Wroclaw, Poland, September 8-10, 2014. Proceedings*, volume 8737 of *Lecture Notes in Computer Science*, pages 125–136. Springer, 2014.

[BHT15] Arnab Bhattacharyya, Pooya Hatami, and Madhur Tulsiani. Algorithmic regularity for polynomials and applications. In Piotr Indyk, editor, *Proceedings of the Twenty-Sixth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2015, San Diego, CA, USA, January 4-6, 2015*, pages 1870–1889. SIAM, 2015.

[BI15] Andrey Bogdanov and Takanori Isobe. White-box cryptography revisited: Space-hard ciphers. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security (CCS)*, pages 1058–1069. ACM, 2015.

[Bih00] Eli Biham. Cryptanalysis of Patarin's 2-Round Public Key System with S-Boxes (2R). In Bart Preneel, editor, *Advances in Cryptology – EUROCRYPT 2000*, volume 1807 of *Lecture Notes in Computer Science*, pages 408–416. Springer Berlin Heidelberg, 2000.

[BK15] Alex Biryukov and Dmitry Khovratovich. Decomposition attack on SASASASAS. Cryptology ePrint Archive, Report 2015/646, 2015. http://eprint.iacr.org/.

[BKL+07] Andrey Bogdanov, Lars R. Knudsen, Gregor Leander, Christof Paar, Axel Poschmann, Matthew J. B. Robshaw, Yannick Seurin, and Charlotte Vikkelsø. PRESENT: An ultra-lightweight block cipher. In Pascal Paillier and Ingrid Verbauwhede, editors, *Cryptographic Hardware and Embedded Systems - CHES 2007*, volume 4727 of *Lecture Notes in Computer Science*, pages 450–466. Springer Berlin Heidelberg, 2007.

[BKR16] Mihir Bellare, Daniel Kane, and Phillip Rogaway. Big-key symmetric encryption: Resisting key exfiltration. Cryptology ePrint Archive, Report 2016/541, to appear in the proceedings of CRYPTO 2016, 2016. http://eprint.iacr.org/.

[BKW03] Avrim Blum, Adam Kalai, and Hal Wasserman. Noise-tolerant learning, the parity problem, and the statistical query model. *Journal of the ACM (JACM)*, 50(4):506–519, jul 2003.

[BLB04] Stéphane Boucheron, Gábor Lugosi, and Olivier Bousquet. Concentration inequalities. In *Advanced Lectures on Machine Learning*, pages 208–240. Springer, 2004.

[BODD+] Achiya Bar-On, Itai Dinur, Orr Dunkelman, Virginie Lallemand, Nathan Keller, and Boaz Tsaban. Cryptanalysis of SP networks with partial non-linear layers. EUROCRYPT 2015, to appear. Available at http://eprint.iacr.org/2014/228.

[BPU16] Alex Biryukov, Léo Perrin, and Aleksei Udovenko. Reverse-engineering the S-box of Streebog, Kuznyechik and STRIBOBr1. In *EUROCRYPT 2016*, pages 372–402. Springer, 2016.

[BS01] Alex Biryukov and Adi Shamir. Structural Cryptanalysis of SASAS. In Birgit Pfitzmann, editor, *Advances in Cryptology–EUROCRYPT 2001*, volume 2045 of *Lecture Notes in Computer Science*, pages 395–405. Springer Berlin Heidelberg, 2001.

[BS03] Dan Boneh and Alice Silverberg. Applications of multilinear forms to cryptography. *Contemporary Mathematics*, 324(1):71–90, 2003.

[BSS+] Ray Beaulieu, Douglas Shors, Jason Smith, Stefan Treatman-Clark, Bryan Weeks, and Louis Wingers. The SIMON and SPECK families of lightweight block ciphers. Cryptology ePrint Archive, Report 2013/404. http://eprint.iacr.org/2013/404.

[CD16] Victor Costan and Srinivas Devadas. Intel SGX explained. Technical report, Cryptology ePrint Archive, Report 2016/086, 20 16. http://eprint. iacr. org, 2016.

[CDD+07] David Cash, Yan Zong Ding, Yevgeniy Dodis, Wenke Lee, Richard Lipton, and Shabsi Walfish. Intrusion-resilient key exchange in the bounded retrieval model. In *Theory of Cryptography*, pages 479–498. Springer, 2007.

[CDK09] Christophe De Cannière, Orr Dunkelman, and Miroslav Knezevic. KATAN and KTANTAN - A family of small and efficient hardware-oriented block ciphers. In *Cryptographic Hardware and Embedded Systems - CHES 2009, 11th International Workshop, Lausanne, Switzerland, September 6-9, 2009, Proceedings*, pages 272–288, 2009.

[CE85] David Chaum and Jan-Hendrik Evertse. Crytanalysis of des with a reduced number of rounds: Sequences of linear factors in block ciphers. In *Advances in Cryptology - CRYPTO '85, Santa Barbara, California, USA, August 18-22, 1985, Proceedings*, volume 218 of *Lecture Notes in Computer Science*, pages 192–211. Springer, 1985.

[CEJO02a] Stanley Chow, Phil Eisen, Harold Johnson, and Paul C. van Oorschot. White-box cryptography and an AES implementation. In *Selected Areas in Cryptography*, pages 250–270. Springer, 2002.

[CEJO02b] Stanley Chow, Phil Eisen, Harold Johnson, and Paul C. van Oorschot. A white-box DES implementation for DRM applications. In *Digital Rights Management*, pages 1–15. Springer, 2002.

[CFL+16] Jung Hee Cheon, Pierre-Alain Fouque, Changmin Lee, Brice Minaud, and Hansol Ryu. Cryptanalysis of the new CLT multilinear map over the integers. In *Advances in Cryptology–EUROCRYPT 2016*, pages 509–536. Springer, 2016.

[CGH+15] Jean-Sébastien Coron, Craig Gentry, Shai Halevi, Tancrède Lepoint, Hemanta K Maji, Eric Miles, Mariana Raykova, Amit Sahai, and Mehdi Tibouchi. Zeroizing without low-level zeroes: New attacks on multilinear maps and their limitations. In *Advances in Cryptology–CRYPTO 2015*, pages 247–266. Springer, 2015.

[CHL+15] Jung Hee Cheon, Kyoohyung Han, Changmin Lee, Hansol Ryu, and Damien Stehlé. Cryptanalysis of the multilinear map over the integers. In *Advances in Cryptology–EUROCRYPT 2015*, pages 3–12. Springer, 2015.

[CL15] Craig Costello and Patrick Longa. Four$\mathbb{Q}$: Four-Dimensional Decompositions on a $\mathbb{Q}$-curve over the Mersenne Prime. In Tetsu Iwata and Jung Hee Cheon, editors, *Advances in Cryptology - ASIACRYPT 2015 - 21st International Conference on the Theory and Application of Cryptology and Information Security, Auckland, New Zealand, November 29 - December 3, 2015, Proceedings, Part I*, volume 9452 of *Lecture Notes in Computer Science*, pages 214–235. Springer, 2015.

[CLLT16] Jean-Sebastien Coron, Moon Sung Lee, Tancrede Lepoint, and Mehdi Tibouchi. Cryptanalysis of ggh15 multilinear maps. Technical report, Cryptology ePrint Archive, Report 2015/1037, 2015. To appear in the proceedings of CRYPTO 2016., 2016.

[CLR15] Jung Hee Cheon, Changmin Lee, and Hansol Ryu. Cryptanalysis of the new CLT multilinear maps. Cryptology ePrint Archive, Report 2015/934, 2015. http://eprint.iacr.org/.

[CLT13] Jean-Sébastien Coron, Tancrède Lepoint, and Mehdi Tibouchi. Practical multilinear maps over the integers. In *Advances in Cryptology–CRYPTO 2013*, pages 476–493. Springer, 2013.

[CLT15] Jean-Sebastien Coron, Tancrède Lepoint, and Mehdi Tibouchi. New multilinear maps over the integers. In *Advances in Cryptology–CRYPTO 2015*, pages 267–286. Springer, 2015.

[CMNT11] Jean-Sébastien Coron, Avradip Mandal, David Naccache, and Mehdi Tibouchi. Fully homomorphic encryption over the integers with shorter public keys. In *Advances in Cryptology–CRYPTO 2011*, pages 487–504. Springer, 2011.

[CNT12] Jean-Sébastien Coron, David Naccache, and Mehdi Tibouchi. Public key compression and modulus switching for fully homomorphic encryption over the integers. In *Advances in Cryptology–EUROCRYPT 2012*, pages 446–464. Springer, 2012.

[Com13] CAESAR Committee. CAESAR– Competition for Authenticated Encryption: Security, Applicability, and Robustness. General secretary Daniel J. Bernstein, information available at http://competitions.cr.yp.to/caesar.html, 2013.

[Dae95] Joan Daemen. Cipher and hash function design strategies based on linear and differential cryptanalysis. PhD thesis, Katholieke Universiteit Leuven, Leuven, Belgium, 1995.

[DDKL15a] Itai Dinur, Orr Dunkelman, Thorsten Kranz, and Gregor Leander. Personal communication, 2015.

[DDKL15b] Itai Dinur, Orr Dunkelman, Thorsten Kranz, and Gregor Leander. Decomposing the ASASA block cipher construction. Cryptology ePrint Archive, Report 2015/507, 2015. http://eprint.iacr.org/2015/507/.

[DFKYZD99] Ye Ding-Feng, Lam Kwok-Yan, and Dai Zong-Duo. Cryptanalysis of "2R" Schemes. In Michael Wiener, editor, *Advances in Cryptology – CRYPTO' 99*, volume 1666 of *Lecture Notes in Computer Science*, pages 315–325. Springer Berlin Heidelberg, 1999.

[DFSS07] Vivien Dubois, Pierre-Alain Fouque, Adi Shamir, and Jacques Stern. Practical Cryptanalysis of SFLASH. In Alfred Menezes, editor, *Advances in Cryptology - CRYPTO 2007*, volume 4622 of *Lecture Notes in Computer Science*, pages 1–12. Springer Berlin Heidelberg, 2007.

[DGS07] Vivien Dubois, Louis Granboulan, and Jacques Stern. Cryptanalysis of HFE with Internal Perturbation. In Tatsuaki Okamoto and Xiaoyun Wang, editors, *Public Key Cryptography – PKC 2007*, volume 4450 of *Lecture Notes in Computer Science*, pages 249–265. Springer Berlin Heidelberg, 2007.

[DH76a] W. Diffie and M. E. Hellman. Multiuser cryptographic techniques. In *AFIPS 1976 National Computer Conference*, pages 109–112. ACM, 1976.

[DH76b] Whitfield Diffie and Martin E Hellman. Multiuser cryptographic techniques. In *Proceedings of the June 7-10, 1976, national computer conference and exposition*, pages 109–112. ACM, 1976.

[Din04] Jintai Ding. A New Variant of the Matsumoto-Imai Cryptosystem through Perturbation. In Feng Bao, Robert Deng, and Jianying Zhou, editors, *Public Key Cryptography – PKC 2004*, volume 2947 of *Lecture Notes in Computer Science*, pages 305–318. Springer Berlin Heidelberg, 2004.

[Dix82] John D. Dixon. Exact solution of linear equations using P-adic expansions. *Nümerische Mathematik*, 40(1):137–141, 1982.

[DLPR13] Cécile Delerablée, Tancrède Lepoint, Pascal Paillier, and Matthieu Rivain. White-box security notions for symmetric encryption schemes. In *Selected Areas in Cryptography–SAC 2013*, pages 247–264. Springer, 2013.

[DMRP12] Yoni De Mulder, Peter Roelse, and Bart Preneel. Cryptanalysis of the Xiao–Lai white-box AES implementation. In *Selected Areas in Cryptography*, pages 34–49. Springer, 2012.

[DPAR00] Joan Daemen, Michaël Peeters, Gilles Van Assche, and Vincent Rijmen. NESSIE proposal: NOEKEON. Homepage http://gro.noekeon.org/, 2000.

[DR02] Joan Daemen and Vincent Rijmen. *The Design of Rijndael: AES - The Advanced Encryption Standard*. Information Security and Cryptography. Springer, 2002.

[DS09] Itai Dinur and Adi Shamir. Cube Attacks on Tweakable Black Box Polynomials. In Antoine Joux, editor, *Advances in Cryptology – EUROCRYPT 2009*, volume 5479 of *Lecture Notes in Computer Science*, pages 278–299. Springer Berlin Heidelberg, 2009.

[Dzi06] Stefan Dziembowski. Intrusion-resilience via the bounded-storage model. In *Theory of Cryptography*, pages 207–224. Springer, 2006.

[Eve87] Jan-Hendrik Evertse. Linear structures in blockciphers. In *Advances in Cryptology - EUROCRYPT '87, Workshop on the Theory and Application of of Cryptographic Techniques, Amsterdam, The Netherlands, April 13-15, 1987, Proceedings*, volume 304 of *Lecture Notes in Computer Science*, pages 249–266. Springer, 1987.

[FD86] Harriet Fell and Whitfield Diffie. Analysis of a Public Key Approach Based on Polynomial Substitution. In HughC. Williams, editor, *Advances in Cryptology – CRYPTO '85 Proceedings*, volume 218 of *Lecture Notes in Computer Science*, pages 340–349. Springer Berlin Heidelberg, 1986.

[FJ03] Jean-Charles Faugère and Antoine Joux. Algebraic Cryptanalysis of Hidden Field Equation (HFE) Cryptosystems Using Gröbner Bases. In Dan Boneh, editor, *Advances in Cryptology - CRYPTO 2003*, volume 2729 of *Lecture Notes in Computer Science*, pages 44–60. Springer Berlin Heidelberg, 2003.

[FKKM16] Pierre-Alain Fouque, Pierre Karpman, Paul Kirchner, and Brice Minaud. Efficient and provable white-box primitives. To appear in the proceedings of Asiacrypt 2016, 2016.

[FM14] Thomas Fuhr and Brice Minaud. Match box meet-in-the-middle attack against KATAN. In *Fast Software Encryption*, pages 61–81. Springer, 2014.

[FP06] Jean-Charles Faugère and Ludovic Perret. Cryptanalysis of 2R- Schemes. In Cynthia Dwork, editor, *Advances in Cryptology - CRYPTO 2006*, volume 4117 of *Lecture Notes in Computer Science*, pages 357–372. Springer Berlin Heidelberg, 2006.

[FP09a] J.-C. Faugère and L. Perret. An Efficient Algorithm for Decomposing Multivariate Polynomials and its Applications to Cryptography. *Journal of Symbolic Computation*, 44(12):1676–1689, 2009.

[FP09b] J.-C. Faugère and L. Perret. High Order Derivatives and Decomposition of Multivariate Polynomials. In *ISSAC '09: Proceedings of the 2009 international symposium on Symbolic and algebraic computation*, pages 207–214. ACM, 2009.

[Für14] Martin Fürer. How fast can we multiply large integers on an actual computer? In *LATIN 2014: Theoretical Informatics*, pages 660–670. Springer, 2014.

[FvzGP10] J.-C. Faugère, J. von zur Gathen, and L. Perret. Decomposition of Generic Multivariate Polynomials. In *ISSAC '10: Proceedings of the 2010 international symposium on Symbolic and algebraic computation*, pages 131–137. ACM, 2010. isbn: 0747-7171 (updated version).

[GGH13a] Sanjam Garg, Craig Gentry, and Shai Halevi. Candidate multilinear maps from ideal lattices. In *Eurocrypt*, volume 7881, pages 1–17. Springer, 2013.

[GGH+13b] Sanjam Garg, Craig Gentry, Shai Halevi, Mariana Raykova, Amit Sahai, and Brent Waters. Candidate indistinguishability obfuscation and functional encryption for all circuits. In *Foundations of Computer Science (FOCS), 2013 IEEE 54th Annual Symposium on*, pages 40–49. IEEE, 2013.

[GGH15] Craig Gentry, Sergey Gorbunov, and Shai Halevi. Graph-induced multilinear maps from lattices. In *Theory of Cryptography*, pages 498–527. Springer, 2015.

[GGNPS13] Benoît Gérard, Vincent Grosso, María Naya-Plasencia, and François-Xavier Standaert. Block ciphers that are easier to mask: How far can we go? In Guido Bertoni and Jean-Sébastien Coron, editors, *Cryptographic Hardware and Embedded Systems - CHES 2013*, volume 8086 of *Lecture Notes in Computer Science*, pages 383–399. Springer Berlin Heidelberg, 2013.

[Gil16] Henri Gilbert. On white-box cryptography. invited talk, Fast Software Encryption 2016, slides available at https://fse.rub.de/slides/wbc_fse2016_hg_2pp.pdf, 2016.

[GJN+15] Jian Guo, Jérémy Jean, Ivica Nikolić, Kexin Qiao, Yu Sasaki, and Siang Meng Sim. Invariant subspace attack against full Midori64. Technical report, Cryptology ePrint Archive, Report 2015/1189, 2015.

[GLS+14a] Vincent Grosso, Gaëtan Leurent, François-Xavier Standaert, Kerem Varici, François Durvaux, Lubos Gaspar, and Stéphanie Kerckhof. Addendum to the CAESAR submission for SCREAM and iSCREAM. Posted on the official CAESAR submission list, available at http://competitions.cr.yp.to/round1/scream-ordering.txt, 2014.

[GLS+14b] Vincent Grosso, Gaëtan Leurent, François-Xavier Standaert, Kerem Varici, François Durvaux, Lubos Gaspar, and Stéphanie Kerckhof. SCREAM & iSCREAM. Entry in the CAESAR competition [Com13], available at http://competitions.cr.yp.to/round1/screamv1.pdf, 2014.

[GLSV14] Vincent Grosso, Gaëtan Leurent, François-Xavier Standaert, and Kerem Varici. LS-designs: Bitslice encryption for efficient masked software implementations. To appear in the proceedings of FSE 2014, available at http://www.uclouvain.be/crypto/people/show/382, 2014.

[GMQ07] Louis Goubin, Jean-Michel Masereel, and Michaël Quisquater. Cryptanalysis of white box DES implementations. In *Selected Areas in Cryptography*, pages 278–295. Springer, 2007.

[GMS16] Sanjam Garg, Pratyay Mukherjee, and Akshayaram Srinivasan. Obfuscation without the vulnerabilities of multilinear maps. Technical report, Cryptology ePrint Archive, Report 2016/390, 2016.

[GNPW13] Jian Guo, Ivica Nikolić, Thomas Peyrin, and Lei Wang. Cryptanalysis of Zorro. Cryptology ePrint Archive, Report 2013/713, 2013. http://eprint.iacr.org/2013/713.

[GPPR11] Jian Guo, Thomas Peyrin, Axel Poschmann, and Matt Robshaw. The LED block cipher. In Bart Preneel and Tsuyoshi Takagi, editors, *Cryptographic Hardware and Embedded Systems – CHES 2011*, volume 6917 of *Lecture Notes in Computer Science*, pages 326–341. Springer Berlin Heidelberg, 2011.

[GPT15] Henri Gilbert, Jérôme Plût, and Joana Treger. Key-Recovery Attack on the ASASA Cryptosystem With Expanding S-Boxes. In *CRYPTO 2015*. Springer, 2015.

[GRR16] Lorenzo Grassi, Christian Rechberger, , and Sondre Rønjom. Subspace trail cryptanalysis and its applications to AES. Cryptology ePrint Archive, Report 2016/592, http://eprint.iacr.org/2016/592, to appear in the proceedings of CRYPTO 2016, 2016.

[Hal15a] Shai Halevi. Cryptographic graded-encoding schemes: Recent developments. TCS+ online seminar, available at https://sites.google.com/site/plustcs/past-talks/20150318shaihaleviibmtjwatson, 2015.

[Hal15b] Shai Halevi. Graded encoding, variations on a scheme. Technical report, Cryptology ePrint Archive, Report 2015/866, 2015. http://eprint. iacr. org, 2015.

[HJ15] Yupu Hu and Huiwen Jia. Cryptanalysis of GGH map. Technical report, Cryptology ePrint Archive, Report 2015/301, 2015.

[HLY12] Yun-Ju Huang, Feng-Hao Liu, and Bo-Yin Yang. Public-Key Cryptography from New Multivariate Quadratic Assumptions. In Marc Fischlin, Johannes A. Buchmann, and Mark Manulis, editors, *Public Key Cryptography - PKC 2012 - 15th International Conference on Practice and Theory in Public Key Cryptography, Darmstadt, Germany, May 21-23, 2012. Proceedings*, volume 7293 of *Lecture Notes in Computer Science*, pages 190–205. Springer, 2012.

[HSW13] Susan Hohenberger, Amit Sahai, and Brent Waters. Full domain hash from (leveled) multilinear maps and identity-based aggregate signatures. In *Advances in Cryptology–CRYPTO 2013*, pages 494–512. Springer, 2013.

[Imp95] Russell Impagliazzo. A personal view of average-case complexity. In *Structure in Complexity Theory Conference, 1995., Proceedings of Tenth Annual IEEE*, pages 134–147. IEEE, 1995.

[JNP14] Jérémy Jean, Ivica Nikolić, and Thomas Peyrin. Tweaks and keys for block ciphers: the TWEAKEY framework. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 274–288. Springer, 2014.

[Jou00] Antoine Joux. A one round protocol for tripartite Diffie–Hellman. In *Algorithmic number theory*, pages 385–393. Springer, 2000.

[KDH13] Ferhat Karakoç, Hüseyin Demirci, and Emre Harmancı. ITUbee: A Software Oriented Lightweight Block Cipher. In *Second International Workshop on Lightweight Cryptography for Security and Privacy (LightSec)*, 2013.

[Kra10] Hugo Krawczyk. Cryptographic extraction and key derivation: The hkdf scheme. In *Advances in Cryptology–CRYPTO 2010*, pages 631–648. Springer, 2010.

[LAAZ11] Gregor Leander, Mohamed Ahmed Abdelraheem, Hoda AlKhzaimi, and Erik Zenner. A cryptanalysis of PRINTcipher: The invariant subspace attack. In Phillip Rogaway, editor, *Advances in Cryptology — CRYPTO 2011*, volume 6841 of *Lecture Notes in Computer Science*, pages 206–221. Springer Berlin Heidelberg, 2011.

[LF06] Éric Levieil and Pierre-Alain Fouque. An Improved LPN Algorithm. In Roberto De Prisco and Moti Yung, editors, *Security and Cryptography for Networks*, volume 4116 of *Lecture Notes in Computer Science*, pages 348–359. Springer Berlin Heidelberg, 2006.

[LMR15] Gregor Leander, Brice Minaud, and Sondre Rønjom. A generic approach to invariant subspace attacks: Cryptanalysis of Robin, iSCREAM and Zorro. In *Advances in Cryptology–EUROCRYPT 2015*, pages 254–283. Springer, 2015.

[LRW02] Moses Liskov, Ronald L. Rivest, and David Wagner. Tweakable block ciphers. In Moti Yung, editor, *Advances in Cryptology — CRYPTO 2002*, volume 2442 of *Lecture Notes in Computer Science*, pages 31–46. Springer Berlin Heidelberg, 2002.

[Mat94] Mitsuru Matsui. Linear Cryptanalysis Method for DES Cipher. In Tor Helleseth, editor, *Advances in Cryptology – EUROCRYPT '93*, volume 765 of *Lecture Notes in Computer Science*, pages 386–397. Springer Berlin Heidelberg, 1994.

[MDFK15] Brice Minaud, Patrick Derbez, Pierre-Alain Fouque, and Pierre Karpman. Key-recovery attacks on ASASA. In *Advances in Cryptology–ASIACRYPT 2015*, pages 3–27. Springer, 2015.

[MF15] Brice Minaud and Pierre-Alain Fouque. Cryptanalysis of the new multilinear map over the integers. *IACR-ePrint (http://eprint. iacr. org/2015/941)*, 2015.

[MI88] Tsutomu Matsumoto and Hideki Imai. Public Quadratic Polynomial-Tuples for Efficient Signature-Verification and Message-Encryption. In D. Barstow, W. Brauer, P. Brinch Hansen, D. Gries, D. Luckham, C. Moler, A. Pnueli, G. Seegmüller, J. Stoer, N. Wirth, and Christoph G. Günther, editors, *Advances in Cryptology – EUROCRYPT '88*, volume 330 of *Lecture Notes in Computer Science*, pages 419–453. Springer Berlin Heidelberg, 1988.

[Min14] Brice Minaud. Linear biases in AEGIS keystream. In *Selected Areas in Cryptography–SAC 2014*, pages 290–305. Springer, 2014.

[Möl08] Niels Möller. On Schönhage's algorithm and subquadratic integer GCD computation. *Mathematics of Computation*, 77(261):589–607, 2008.

[MS04] Thom Mulders and Arne Storjohann. Certified dense linear system solving. *Journal of Symbolic Computation*, 37(4):485–510, 2004.

[MS15] Brice Minaud and Yannick Seurin. The iterated random permutation problem with applications to cascade encryption. In *Advances in Cryptology–CRYPTO 2015*, pages 351–367. Springer, 2015.

[MSZ16a] Eric Miles, Amit Sahai, and Mark Zhandry. Annihilation attacks for multilinear maps: Cryptanalysis of indistinguishability obfuscation over GGH13. Technical report, Cryptology ePrint Archive, Report 2016/147, 2016.

[MSZ16b] Eric Miles, Amit Sahai, and Mark Zhandry. Secure obfuscation in a weak multilinear map model: A simple construction secure against all known attacks. Technical report, Cryptology ePrint Archive, Report 2016/588, 2016.

[Pat95] Jacques Patarin. Cryptanalysis of the Matsumoto and Imai Public Key Scheme of Eurocrypt'88. In Don Coppersmith, editor, *Advances in Cryptology – CRYPTO' 95*, volume 963 of *Lecture Notes in Computer Science*, pages 248–261. Springer Berlin Heidelberg, 1995.

[Pat96] Jacques Patarin. Hidden Fields Equations (HFE) and Isomorphisms of Polynomials (IP): Two New Families of Asymmetric Algorithms. In Ueli Maurer, editor, *Advances in Cryptology – EUROCRYPT '96*, volume 1070 of *Lecture Notes in Computer Science*, pages 33–48. Springer Berlin Heidelberg, 1996.

[PG97] Jacques Patarin and Louis Goubin. Asymmetric Cryptography with S-Boxes. In *ICICS'97*, volume 1334 of *Lecture Notes in Computer Science*, pages 369–380. Springer, 1997.

[PGC01] Jacques Patarin, Louis Goubin, and Nicolas Courtois. Quartz, 128-bit long digital signatures. CT-RSA Conference, 2001.

[PR13] Emmanuel Prouff and Matthieu Rivain. Masking against side-channel attacks: A formal security proof. In Thomas Johansson and Phong Q. Nguyen, editors, *Advances in Cryptology – EUROCRYPT 2013*, volume 7881 of *Lecture Notes in Computer Science*, pages 142–159. Springer Berlin Heidelberg, 2013.

[PUB16] Léo Perrin, Aleksei Udovenko, and Alex Biryukov. Cryptanalysis of a theorem: Decomposing the only known solution to the big APN problem. *ePrint report, to appear in the proceedings of CRYPTO 2016*, 2016.

[RASA14] Shahram Rasoolzadeh, Zahra Ahmadian, Mahmood Salmasizadeh, and Mohammad Reza Aref. Total break of Zorro using linear and differential attacks. *The ISC International Journal of Information Security*, 6(1), 2014. Available at `http://isecure-journal.com/index.php/isecure/article/view/14-215/104`.

[Reg05] Oded Regev. On Lattices, Learning with Errors, Random Linear Codes, and Cryptography. In *STOC'05*, pages 84–93. ACM Press, 2005.

[RM85] J.A. Reeds and J.L. Manferdelli. Des has no per round linear factors. In GeorgeRobert Blakley and David Chaum, editors, *Advances in Cryptology*, volume 196 of *Lecture Notes in Computer Science*, pages 377–389. Springer Berlin Heidelberg, 1985.

[Røn16] Sondre Rønjom. Invariant subspaces in Simpira. Technical report, Cryptology ePrint Archive, Report 2016/248, 2016.

[RP97] Vincent Rijmen and Bart Preneel. A family of trapdoor ciphers. In Eli Biham, editor, *Fast Software Encryption*, volume 1267 of *Lecture Notes in Computer Science*, pages 139–148. Springer Berlin Heidelberg, 1997.

[Sha85] Adi Shamir. Identity-based cryptosystems and signature schemes. In *Advances in cryptology*, pages 47–53. Springer, 1985.

[Sha11] Ronen Shaltiel. An introduction to randomness extractors. In *Automata, languages and programming*, pages 21–41. Springer, 2011.

[Sol14] Hadi Soleimany. Probabilistic slide cryptanalysis and its applications to LED-64 and Zorro. To appear in the proceedings of FSE 2014, available at http://research.ics.aalto.fi/publications/bibdb2014/pdf/fse2014.pdf, 2014.

[Sti02] Douglas Robert Stinson. Universal hash families and the leftover hash lemma, and applications to cryptography and computing. *Journal of Combinatorial Mathematics and Combinatorial Computing*, 42:3–32, 2002.

[Sto05] Arne Storjohann. The shifted number system for fast linear algebra on integer matrices. *Journal of Complexity*, 21(4):609 – 650, 2005. Festschrift for the 70th Birthday of Arnold Schonhage.

[Tea] The Sage Development Team. Sage Mathematics Software. http://www.sagemath.org.

[Tod15] Yosuke Todo. Integral cryptanalysis on full MISTY1. In *CRYPTO 2015*, pages 413–432. Springer, 2015.

[Vad04] Salil P Vadhan. Constructing locally computable extractors and cryptosystems in the bounded-storage model. *Journal of Cryptology*, 17(1):43–77, 2004.

[VDGHV10] Marten Van Dijk, Craig Gentry, Shai Halevi, and Vinod Vaikuntanathan. Fully homomorphic encryption over the integers. In *Advances in cryptology–EUROCRYPT 2010*, pages 24–43. Springer, 2010.

[WBDY98] Hongjun Wu, Feng Bao, RobertH. Deng, and Qin-Zhong Ye. Cryptanalysis of Rijmen-Preneel Trapdoor Ciphers. In Kazuo Ohta and Dingyi Pei, editors, *Advances in Cryptology – ASIACRYPT'98*, volume 1514 of *Lecture Notes in Computer Science*, pages 126–132. Springer Berlin Heidelberg, 1998.

[WMGP07] Brecht Wyseur, Wil Michiels, Paul Gorissen, and Bart Preneel. Cryptanalysis of white-box DES implementations with arbitrary external encodings. In *Selected Areas in Cryptography*, pages 264–277. Springer, 2007.

[WWGY14] Yanfeng Wang, Wenling Wu, Zhiyuan Guo, and Xiaoli Yu. Differential cryptanalysis and linear distinguisher of full-round Zorro. In Ioana Boureanu, Philippe Owesarski, and Serge Vaudenay, editors, *Applied Cryptography and Network Security*, volume 8479 of *Lecture Notes in Computer Science*, pages 308–323. Springer International Publishing, 2014.

[Wys09] Brecht Wyseur. White-box cryptography. PhD thesis, KU Leuven, 2009.

[XL09] Yaying Xiao and Xuejia Lai. A secure implementation of white-box AES. In *Computer Science and its Applications, 2009. CSA'09. 2nd International Conference on*, pages 1–6. IEEE, 2009.

[Zim15] Joe Zimmerman. How to obfuscate programs directly. In *Advances in Cryptology-EUROCRYPT 2015*, pages 439–467. Springer, 2015.

Résumé

Dans cette thèse, nous nous intéressons à la sécurité de quelques primitives cryptographiques récentes, d'abord symétriques puis asymétriques, en passant par le modèle en boîte blanche, qui est à certains égards intermédiaire.

Dans un premier temps, nous montrons l'existence de fonctions linéaires non triviales commutant avec la fonction de tour de certains chiffrements par bloc, dont découlent des attaques par autosimilarité et sous-espace invariant. Nous nous intéressons ensuite à la cryptanalyse de la structure ASASA, où deux couches non linéaires S sont imbriquées dans des couches affines A. Notre cryptanalyse structurelle permet de casser des instances de chiffrement symétrique, multivarié et en boîte blanche. En nous concentrant sur le modèle d'incompressibilité en boîte blanche, nous montrons ensuite comment réaliser un chiffrement par bloc et un générateur de clef efficaces dont la sécurité est prouvable. Finalement, du côté purement asymétrique, nous décrivons une attaque polynomiale contre une construction récente d'application multilinéaire.

Mots-clés: Cryptanalyse symétrique, cryptanalyse structurelle, sécurité prouvable, boîte blanche, applications multilinéaires.

Abstract

In this thesis, we study the security of some recent cryptographic primitives, both symmetric and asymmetric. Along the way we also consider white-box primitives, which may be regarded as a middle ground betwen symmetric and asymmetric cryptography.

We begin by showing the existence of non-trivial linear maps commuting with the round function of some recent block cipher designs, which give rise to self-similarity and invariant subspace attacks. We then move on to the structural cryptanalysis of ASASA schemes, where nonlinear layers S alternate with affine layers A. Our structural cryptanalysis applies to symmetric, multivariate, as well as white-box instances. Focusing on the white-box model of incompressibility, we then build an efficient block cipher and key generator that offer provable security guarantees. Finally, on the purely asymmetric side, we describe a polynomial attack against a recent multilinear map proposal.

Keywords: Symmetric Cryptanalysis, Structural Cryptanalysis, Provable Security, White-Box Cryptography, Multilinear Maps.