Techniques of cryptanalysis for symmetric-key primitives

Daniel Coggia

▶ **To cite this version:**

Daniel Coggia. Techniques of cryptanalysis for symmetric-key primitives. Cryptography and Security [cs.CR]. Sorbonne Université, 2021. English. NNT : 2021SORUS217 . tel-03515131

HAL Id: tel-03515131
https://theses.hal.science/tel-03515131

Submitted on 6 Jan 2022

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Sorbonne Université

École doctorale Informatique, Télécommunications et Électronique
ED130, Paris

Inria de Paris / Équipe-projet COSMIQ

Techniques de cryptanalyse dédiées au chiffrement à bas coût

Thèse de doctorat d'informatique

présentée par

Daniel Coggia

soutenue publiquement le 8 octobre 2021

devant un jury composé de :

Christina Boura	Université de Versailles Saint-Quentin	Directrice
Anne Canteaut	Inria	Directrice
Pierre-Alain Fouque	Université Rennes 1	Rapporteur
Jérémy Jean	ANSSI	Examinateur
Gregor Leander	Ruhr-Universität Bochum	Examinateur
Marine Minier	Université de Lorraine	Rapporteure
Gaël Thomas	DGA	Examinateur
Damien Vergnaud	Sorbonne Université	Examinateur

Remerciements

Pour découvrir le monde de la recherche, il me fallait un guide et j'en eus deux formidables. Merci Anne, merci Christina pour vos idées, votre patience et votre humanité. Merci aussi à Alain, qui me prit sous son aile le temps d'une cryptanalyse.

Je suis également très reconnaissant à Marine Minier et Pierre-Alain Fouque d'avoir rapporté cette thèse, ainsi qu'à Jérémy Jean, Gregor Leander, Gaël Thomas et Damien Vergnaud d'avoir accepté de faire partie du jury.

Les membres de l'équipe SECRET puis COSMIQ éclairèrent ces trois années de leur bonne humeur, de leurs récits et de leur sagesse. Merci à Pascale, Jean-Pierre, Nicolas, Anthony, María, Gaëtan, André, Léo et Christelle, véritables piliers sur lesquels nous nous appuyons pour travailler, comprendre ou explorer. Merci à ceux qui m'ont précédé, Yann, Sébastien, Vivien, Antoine, Kevin, Thomas, Andrea, Rémi, André, et à ceux qui me succèdent, Augustin, Pierre, Aurélie, Simona, Paul, Lucien, Johanna, Rocco et Clara. Merci Matthieu, Ferdinand, Valentin, Xavier, Antonio, Nicolas et Clémence, car avec chacun de vous, partager un bureau fut un réel plaisir. Merci Son et Denis, pour votre présence familière à l'étage et pour avoir élargi ma culture du domaine.

Je tiens à remercier les équipes du restaurant Sully de l'AGRAF qui me régalèrent pendant deux ans ainsi que les équipes de propreté du centre Inria de Paris dont j'appréciais quotidiennement l'impeccable service.

Cette thèse n'aurait pas été possible sans l'adhésion de la Direction Générale de l'Armement et le précieux soutien d'Olivier Ducable et Chantal Keryjaouen ; je les en remercie. Merci également à mon parrain DGA, Raphaël Bost, pour sa disponibilité et ses bons conseils.

N'échapperont pas à ces remerciements Louise, Éloïse, Thomas, Raphaël, Thibaut, Jean et Jean-Yves, ou encore Janique, Paul, Jeanne et Clotilde, sans oublier Jean-Dominique, Odile, Laurent, Véronique, Pascal, Flaminia, Annie, Alban, Clémence, Geoffrey, Caroline, Maxime et Thomas. Enfin, merci Marianne.

Contents

Notations

- $\mathbb{F}_2$ is $\mathbb{Z}/2\mathbb{Z}$, the field exclusively composed of the two elements 0 and 1 which we sometimes refer to as *bits*.
- For an integer n, $\mathbb{F}_2^n$ is the $\mathbb{F}_2$–vector space with n coordinates in $\mathbb{F}_2$. In general, a vector will be written with a bold symbol and the indexing will start from 0: $\boldsymbol{x} = (x_0, \ldots, x_{n-1})$.
- Polynomial variables are denoted by uppercase letters like X, Y, K. When there are several polynomial variables $X_0, \ldots, X_{n-1}$, X is the entire tuple of variables $X_i,\ i \in [0, n-1]$ and $X_{\mathcal{I}}$ is the tuple of variables $X_i,\ i \in \mathcal{I}$.
- $\mathbb{F}_2[X_0, \ldots, X_{n-1}]$ is the ring of polynomials over $\mathbb{F}_2$ with n variables.
- $\boldsymbol{x} \preceq \boldsymbol{y}$ when $\boldsymbol{x}, \boldsymbol{y} \in \mathbb{F}_2^n$ means that $\forall i \in [0, n-1],\ y_i = 0 \Rightarrow x_i = 0$. $\boldsymbol{x} \prec \boldsymbol{y}$ means that $\boldsymbol{x} \preceq \boldsymbol{y}$ and $\boldsymbol{x} \neq \boldsymbol{y}$.
- $\mathrm{Prec}\,(\boldsymbol{u})$ is the set $\{\, \boldsymbol{x} \in \mathbb{F}_2^n \mid \boldsymbol{x} \preceq \boldsymbol{u} \,\}$.
- $\mathrm{supp}\,(\boldsymbol{x})$ is the set $\{\, i \in [0, n-1] \mid x_i = 1 \,\}$, called the *support* of the vector $\boldsymbol{x} \in \mathbb{F}_2^n$.
- $\mathrm{wt}\,(\boldsymbol{x})$ is the *Hamming weight* of $\boldsymbol{x}$, i.e. the integer $\#\mathrm{supp}\,(\boldsymbol{x})$.
- $\boldsymbol{e}_i$ denotes the vector $(0, \ldots, 0, \overset{i}{\underset{\downarrow}{1}}, 0, \ldots, 0)$ where the length depends on the context. Similarly, $\boldsymbol{e}_{\mathcal{S}}$ where $\mathcal{S}$ is a set of indices is the vector such that $\mathrm{supp}\,(\boldsymbol{e}_{\mathcal{S}}) = \mathcal{S}$.
- $(\boldsymbol{u}|\boldsymbol{v})$ denotes the concatenation of the vectors $\boldsymbol{u}$ and $\boldsymbol{v}$.
- $\mathbf{0}$ and $\mathbf{1}$ are the vectors $(0, 0, \ldots, 0)$ and $(1, 1, \ldots, 1)$. If needed, the size of the vector, n, is added: $\mathbf{1}_n$.
- $X \sim \mathcal{L}$ means that the random variable X follows the law of probability $\mathcal{L}$.
- $\mathcal{U}\,(\mathcal{X})$ is the uniform law of probability on the set $\mathcal{X}$.

Présentation des travaux

La cryptographie est une science qui fournit des composants algorithmiques, appelés cryptosystèmes, essentiels à la sécurité de toute infrastructure numérique. La diversité des objectifs cryptographiques — confidentialité, intégrité et authentification — implique naturellement une grande diversité dans les prérequis et le fonctionnement des cryptosystèmes. Un objectif très ancien est par exemple l'échange de messages chiffrés à travers un canal de communication peu sûr entre deux interlocuteurs qui partagent une clef de chiffrement. Les cryptosystèmes qui remplissent cet objectif sont appelés algorithmes de chiffrement symétrique. Certains cryptosystèmes, dits asymétriques, permettent par ailleurs le partage de clefs de chiffrements via un canal de communication public, ce qui est très utile pour chiffrer le traffic entre deux ordinateurs connectés à Internet.

La sécurité des cryptosystèmes repose sur celle de composants algorithmiques plus simples, les primitives cryptographiques. En particulier, de nombreux chiffrements symétriques sont construits autour d'une primitive appelée chiffrement par bloc. Il est cependant souvent impossible de prouver mathématiquement la sécurité d'une primitive. Ainsi, la confiance accordée à une primitive a un prix, celui de son analyse constante et transparente par une communauté académique dédiée. Cette thèse est une modeste contribution à l'effort de cryptanalyse de primitives symétriques comme les chiffrements par bloc ou les générateurs pseudo-aléatoires.

J'ai étudié dans un premier temps une famille de distingueurs fondés sur la propagation de sous-espaces vectoriels différentiels dans les chiffrements par bloc de construction SPN. Je me suis ensuite tourné vers la création de méthodes permettant aux cryptographes de modéliser un problème de cryptanalyse de primitive symétrique en problème MILP, afin d'exploiter certains logiciels solutionneurs de problèmes MILP très performants. Je me suis aussi intéressé à l'analyse algébrique des primitives symétriques, notamment au calcul d'une partie de leur forme algébrique normale, utile dans les attaques de type *cube*. Enfin, j'ai travaillé vers le début de ma thèse sur un sujet relativement éloigné des primitives symétriques avec l'étude d'une primitive de chiffrement asymétrique fondée sur les codes correcteurs d'erreur et la métrique rang.

Chapitres 1 et 2 – Introduction à la cryptographie moderne et à la cryptanalyse différentielle

Je commence par introduire les concepts centraux de la cryptographie moderne au chapitre 1, avec la relation entre preuve de sécurité et cryptanalyse comme fil conducteur, et avec le chiffrement symétrique comme sujet principal.

Sécurité calculatoire. Pour prétendre qu'un cryptosystème est sûr, il faut d'abord définir contre quelles menaces l'analyse est valide. Dans le modèle calculatoire, on considère l'adversaire comme

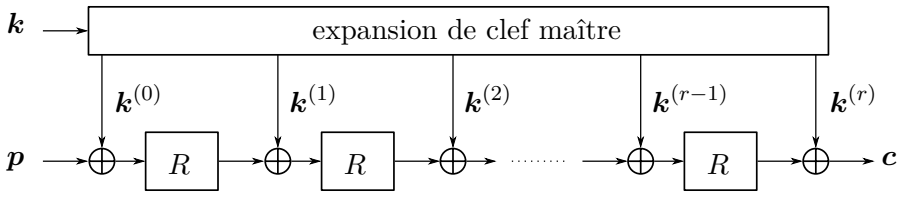

FIGURE 0.1 – Chiffrement par bloc itératif.

ayant une certaine capacité de calcul et un accès bien défini à la donnée produite par le cryptosystème. D'autres modèles considèrent que l'adversaire peut mesurer certaines caractéristiques physiques de l'exécution d'un algorithme cryptographique, comme le temps de calcul ou la consommation d'énergie, ou que l'adversaire peut altérer l'exécution d'un cryptosystème. On parle alors d'attaques par canaux cachés ou par injection de fautes. Dans cette thèse, je n'étudie la sécurité des cryptosystèmes que dans le cadre du modèle calculatoire.

Dans ce modèle, les cryptosystèmes ont souvent une preuve de sécurité qui repose sur une ou plusieurs conjectures. Parmi ces conjectures, on peut trouver le fait qu'un chiffrement par bloc ne peut pas être distingué d'une permutation aléatoire plus rapidement qu'en devinant la clef, ou encore qu'il est impossible d'inverser une fonction à trappe (sans la connaissance de la trappe) en-deçà d'une certaine complexité de calcul.

Chiffrements par bloc. Un chiffrement par bloc est une primitive cryptographique utilisée dans les chiffrements symétriques. C'est une application

$$B : \begin{cases} \mathbb{F}_2^m \times \mathbb{F}_2^n & \longrightarrow & \mathbb{F}_2^n \\ (\boldsymbol{k}, \boldsymbol{p}) & \longmapsto & \boldsymbol{c} = B_{\boldsymbol{k}}(\boldsymbol{p}), \end{cases}$$

où $\boldsymbol{k}$ est appelée la clef, $\boldsymbol{p}$ le message clair et $\boldsymbol{c}$ le message chiffré.

Pour que l'exécution de cette application soit rapide en pratique tout en assurant le niveau de sécurité désiré, ces primitives sont souvent construites selon une structure itérative : des clefs de tour sont dérivées de la clef maître puis s'appliquent en alternance sur le message clair une fonction de tour R et l'addition d'une clef de tour (voir figure 0.1).

La fonction de tour R est elle-même souvent construite selon les principes de confusion et de diffusion de Shannon [Sha49], qui stipulent respectivement que la relation entre l'entrée et la sortie d'une telle fonction doit être complexe et que chaque bit de sortie doit être fonction d'un maximum de bits d'entrée. En particulier, dans les chiffrements dits SPN et toujours dans un souci d'équilibre entre légèreté et sécurité, la fonction de tour possède deux couches : la première applique de petites fonctions complexes et non-linéaires en parallèle, les *Sbox*, pour apporter de la confusion et la seconde applique une fonction linéaire sur tout l'état pour apporter de la diffusion efficacement (voir figure 0.2).

Le chiffrement par bloc par excellence, l'*Advanced Encryption Standard* (AES) [Nisa], probablement la primitive cryptographique qui chiffre le plus important volume de données sur Internet, suit une structure SPN. Sa taille de bloc n est de 128 bits et il en existe trois variantes pour des clefs de 128, 192 ou 256 bits. Chaque variante a respectivement 10, 12 et 14 tours.

Le nombre de tours d'un chiffrement itératif sera, encore une fois, choisi pour l'équilibre qu'il promet en termes d'efficacité et de sécurité. Il existe par exemple des attaques sur chaque variante

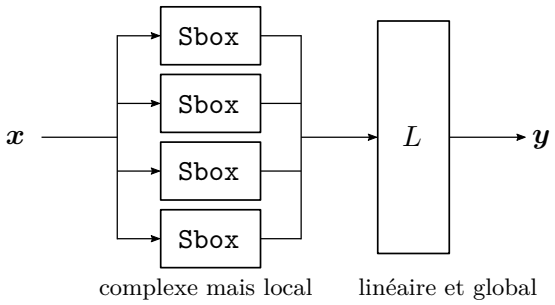

FIGURE 0.2 – Fonction de tour d'un chiffrement par bloc SPN.

de l'AES réduite respectivement à 7 (/10), 8 (/12) et 9 (/14) [Jea13] tours mais l'AES conserve encore une marge de sécurité de plusieurs tours dans chacune de ses variantes.

Cryptanalyse différentielle. Le but de la cryptanalyse différentielle est de distinguer un chiffrement par bloc d'une permutation aléatoire en étudiant les statistiques de propagation d'une différence d'entrée $\boldsymbol{a}$ vers une différence de sortie $\boldsymbol{b}$. On cherche en particulier un couple $(\boldsymbol{a}, \boldsymbol{b})$ tel que la probabilité sur $\boldsymbol{x}$ et $\boldsymbol{k}$ d'avoir

$$B_{\boldsymbol{k}}(\boldsymbol{x} + \boldsymbol{a}) + B_{\boldsymbol{k}}(\boldsymbol{x}) = \boldsymbol{b}$$

est grande.

Dans une fonction linéaire L, typiquement la couche de diffusion d'une fonction de tour, une différence $\boldsymbol{a}$ se propage de façon déterministe vers la différence $\boldsymbol{b} = L(\boldsymbol{a})$. Au contraire dans une fonction non-linéaire F, comme une *Sbox*, une différence d'entrée $\boldsymbol{a}$ peut donner plusieurs différences de sortie, et toutes ces possibilités sont regroupées au sein de la DDT (*Difference Distribution Table*) :

$$\mathrm{DDT} : (\boldsymbol{a}, \boldsymbol{b}) \in \mathbb{F}_2^{2n} \mapsto \left|\{\, \boldsymbol{x} \in \mathbb{F}_2^n \mid F(\boldsymbol{x} + \boldsymbol{a}) + F(\boldsymbol{x}) = \boldsymbol{b} \,\}\right|.$$

Étudier les propriétés différentielles d'une primitive symétrique, et d'un chiffrement par bloc SPN en particulier, revient souvent à étudier les interactions entre la couche linéaire et les DDT des *Sbox*.

La cryptanalyse différentielle connaît de nombreuses variantes comme les différentielles tronquées [Knu95] ou les différentielles impossibles [BBS99]. Ces notions sont abordées au chapitre 2.

Chapitre 3 – Propagation de sous-espaces différentiels dans les chiffrements SPN

Ce chapitre reprend un travail effectué avec Anne Canteaut et Christina Boura et publié dans ToSC 2019, numéro 1 [BCC19].

Nous nous sommes intéressés au distingueur sur 5 tours d'AES présenté par Grassi, Rechberger et Rønjom à Eurocrypt 2017 [GRR17]. Ce distingueur repose sur l'existence de deux sous-espaces vectoriels V et W de $\mathbb{F}_2^{128}$ tels que pour tout espace affine de direction V, le nombre de paires d'éléments $\boldsymbol{x}$, $\boldsymbol{x}'$ de cet espace affine qui vérifient

$$\mathrm{AES}^{(5)}(\boldsymbol{x}) + \mathrm{AES}^{(5)}(\boldsymbol{x}') \in W$$

est un multiple de 8, où $\text{AES}^{(5)}$ désigne cinq tours d'AES. La structure de V et W ne permettant pas d'étendre ce distingueur en un recouvrement de clef, Grassi proposa peu de temps après [Gra18] des distingueurs sur 4 tours, inspirés de la preuve du distingueur *multiple-de-8*, et extensibles en un recouvrement de clef sur 5 tours très performant.

Les preuves originales de ces distingueurs étaient cependant peu satisfaisantes par leur redondance et par la difficulté d'en tirer des enseignements utiles à la cryptanalyse sur plus de tours, ou sur d'autres chiffrements SPN, avec une technique similaire. Nous avons donc proposé des preuves alternatives plus compactes et mieux structurées qui s'adaptent à d'autres chiffrements SPN.

Plus précisément, ces distingueurs sont des conséquences de deux faits indépendants. Le premier est qu'il existe des propagations de sous-espaces vectoriels sur 2 tours d'AES. Le second — et c'est là qu'est la nouveauté et que nous introduisons une nouvelle formulation — est que pour au moins un espace W en fin de propagation et pour chacun des espaces affines qu'il dirige, la fonction

$$\left\{\boldsymbol{p}^0, \boldsymbol{p}^1\right\} \mapsto R\left(\boldsymbol{p}^0\right) + R\left(\boldsymbol{p}^1\right),$$

qui à chaque paire de vecteurs dans cet espace affine, associe la différence des images par la fonction de tour, est invariante par une relation d'équivalence entre paires de vecteurs. À partir de cette formulation par relation d'équivalence, nous pouvons prouver ce second point pour l'AES et, pour tout SPN, caractériser les espaces W pour lesquels la propriété est vraie.

En conjonction avec le travail de Leander, Tezcan et Wiemer sur la recherche de propagation de sous-espaces [LTW18], nos travaux donnent une méthode pour analyser la menace posée par ce type de distingueurs pour tout chiffrement SPN, comme nous le montrons sur les exemples de `Midori`, `KLEIN`, `LED` et `SKINNY` [Ban+15 ; GNL12 ; Guo+11 ; Bei+16].

Chapitre 4 – Modèles MILP efficaces pour la cryptographie symétrique

Ce chapitre reprend un travail effectué avec Christina Boura et publié dans ToSC 2020, numéro 3 [BC20]. Nous nous sommes intéressés à la modélisation de problèmes de cryptanalyse différentielle en problème MILP (pour *Mixed-Integer Linear Programming*).

En effet, conduire une cryptanalyse peut nécessiter d'effectuer des calculs spécifiques à la primitive étudiée et donc d'écrire les programmes qui effectueront ces calculs efficacement. L'élaboration de tels programmes efficaces peut se révéler extrêmement chronophage, si bien que de plus en plus de cryptographes se tournent vers l'utilisation de logiciels solutionneurs de problèmes MILP très performants et déjà disponibles. La condition d'utililisation d'un tel solutionneur est donc de pouvoir traduire un problème de cryptanalyse en problème MILP efficacement résoluble.

Problème MILP. Un problème MILP est de la forme

$$\text{maximiser } \sum_{i=0}^{n} c_i x_i \text{ sous les contraintes } \sum_{i=0}^{n} a_{i,j} x_i \leqslant b_j,\ j \in [0, m-1],$$

où $a_{i,j}, b_j, c_i$ sont des nombres réels et où les variables x_i doivent prendre des valeurs entières ; n est le nombre de variables et m le nombre d'inégalités ou contraintes. Nous nous intéresserons essentiellement à des variables x_i dans $\{0,1\} \subseteq \mathbb{Z}$ et notre but sera, en première approximation dans cette présentation, de minimiser le nombre de variables et de contraintes nécessaires à la modélisation d'un problème cryptographique donné.

Modéliser au niveau des bits. Le premier problème de cryptanalyse à être modélisé en problème MILP [Mou+11] fut le calcul du nombre minimum de *Sbox* actives dans un SPN défini par des opérations sur des mots (de 4 ou 8 bits), comme l'AES par exemple, et nécessitait peu de variables et une modélisation élémentaire. Néanmoins, certaines spécifications manipulant chaque bit individuellement, ou certains problèmes plus compliqués comme la recherche de différentielles impossibles, ont besoin de modèles beaucoup plus volumineux en variables et en contraintes. Plus généralement, pour que la modélisation en problème MILP puisse couvrir un large champ d'applications en cryptographie symétrique, il est important de pouvoir modéliser le comportement des primitives au niveau des bits. C'est précisément dans ce cadre que nos travaux prennent place.

Modélisation des *Sbox*. En particulier, la modélisation des propriétés différentielles ou linéaires d'une *Sbox* est un problème ardu pour obtenir des modèles de SPN entiers efficacement résolubles. En effet, dans le cas de la cryptanalyse différentielle et de ses variantes, il faut pouvoir modéliser la DDT. Par exemple, pour minimiser un nombre de *Sbox* actives ou rechercher des différentielles impossibles, il faut pouvoir modéliser la contrainte booléenne

$$(\boldsymbol{a}, \boldsymbol{b}) \mapsto \begin{cases} 0 & \text{si } \mathrm{DDT}(\boldsymbol{a}, \boldsymbol{b}) = 0, \\ 1 & \text{sinon,} \end{cases}$$

par des inégalités à coefficients réels et des variables entières.

Sun et al. [Sun+14b ; Sun+14a] furent les premiers à proposer des modèles MILP pour l'analyse des SPN au niveau des bits, avec notamment deux méthodes de modélisation pour les *Sbox* de 4 bits. Pour modéliser les *Sbox* de 8 bits, Abdekhalek et al. [Abd+17] utilisèrent un algorithme de minimisation de la représentation en produit de sommes d'une fonction booléenne. Si leur technique fut un succès pour la *Sbox* de `SKINNY-128` avec 372 inégalités, elle fut moins probante pour celle de l'AES avec 8302 inégalités, ce qui est souvent trop pour construire des modèles utiles en pratique.

Nous proposons donc trois nouvelles méthodes de modélisation MILP de contrainte booléenne qui peuvent servir, entre autres, à la modélisation des DDT. Soit $F : \mathbb{F}_2^n \to \mathbb{F}_2$ une contrainte booléenne à modéliser. Chaque élément de $\mathbb{F}_2^n$ est aussi vu comme un point de l'hypercube $\{0,1\}^n \subset \mathbb{R}^n$.

Notre première méthode, inspirée de celle de Sun et al., repose sur le calcul de l'enveloppe convexe des antécédents de 1 par F, qui dans notre exemple correspondent aux transitions possibles de la DDT. En effet, l'enveloppe convexe prend la forme d'une liste d'inégalités qui caractérise cet ensemble de points. En revanche, cette liste contient des inégalités redondantes car nous avons aussi pour contrainte que les variables sont à valeurs dans $\{0,1\}$. Si Sun et al. calculent directement un sous-ensemble d'inégalités minimal, nous choisissons de grossir cette liste avec de nouvelles inégalités issues des premières avant minimisation. Plus précisément, k inégalités

$$\begin{gathered} a_{0,0}x_0 + \cdots + a_{0,n-1}x_{n-1} \leqslant b_0, \\ \vdots \\ a_{(k-1),0}x_0 + \cdots + a_{(k-1),n-1}x_{n-1} \leqslant b_{k-1}, \end{gathered}$$

issues de l'enveloppe convexe et qui partagent un antécédent de 1 sur leur bord, sont susceptibles de fournir, par leur somme

$$\left(\sum_j a_{j,0}\right) x_0 + \cdots + \left(\sum_j a_{j,n-1}\right) x_{n-1} \leqslant \sum_j b_j,$$

une nouvelle inégalité correcte et discriminant un ensemble de points différent. Cette méthode toute simple donne des résultats nettement meilleurs que les méthodes précédentes pour les *Sbox* de 4 bits mais elle est limitée aux *Sbox* de 6 bits par le calcul de l'enveloppe convexe.

Notre seconde méthode, inspirée de celle d'Abdelkhalek et al., est basée sur la couverture des antécédents de 0 par des ensembles de la forme $\boldsymbol{a} \oplus \mathrm{Prec}(\boldsymbol{u})$, où

$$\mathrm{Prec}(\boldsymbol{u}) \stackrel{\text{def}}{=} \{\, \boldsymbol{x} \in \mathbb{F}_2^n \mid \forall i \in [0, n-1],\ u_i = 0 \Rightarrow x_i = 0 \,\}.$$

En effet, l'ensemble $\boldsymbol{a} \oplus \mathrm{Prec}(\boldsymbol{u})$ (avec $\mathrm{supp}(\boldsymbol{a}) \cap \mathrm{supp}(\boldsymbol{u}) = \emptyset$) peut être éliminé par la seule inégalité

$$-\sum_{i \in \mathrm{supp}(\boldsymbol{a})} x_i + \sum_{i \notin \mathrm{supp}(\boldsymbol{a}+\boldsymbol{u})} x_i \geqslant 1 - \mathrm{wt}(\boldsymbol{a}).$$

Cette méthode offre des résultats légèrement meilleurs que ceux d'Abdelkhalek et al. mais son utilisation est plus rapide sans nécessiter d'utiliser en « boîte noire » un programme de minimisation d'une représentation en produit de sommes.

Notre troisième méthode couvre les antécédents de 0 par des boules en métrique de Hamming. En effet, la boule de centre $\boldsymbol{c}$ et de rayon d, $\mathcal{B}(d, \boldsymbol{c})$, peut être éliminée par la seule inégalité

$$\sum_{i=0}^{n} (1 - c_i) x_i + c_i (1 - x_i) \geqslant d + 1.$$

De plus, on peut généraliser cette inégalité dans le cas où quelques antécédents de 1 rapprochés appartiennent au bord de la boule. Enfin, à partir d'inégalités éliminant trois boules voisines de rayon 1, on peut obtenir une nouvelle inégalité correcte. L'utilisation conjointe des méthodes 2 et 3 donne de très bons résultats, avec par exemple un nombre d'inégalités de 2882 pour la DDT de la *Sbox* de l'AES.

Modélisation des couches linéaires. Modéliser les propriétés différentielles d'une couche linéaire revient dans de nombreux cas à modéliser une contrainte $\boldsymbol{A} \cdot \boldsymbol{x} = \boldsymbol{0}$ où $\boldsymbol{A}$ est une matrice à coefficients dans $\mathbb{F}_2$. La méthode naturelle est de modéliser les contraintes données par chaque ligne de la matrice :

$$a_{j,0} x_0 \oplus \cdots \oplus a_{j,n-1} x_{n-1} = 0.$$

Cependant, nous montrons qu'une contrainte de la forme $x_0 \oplus \cdots \oplus x_{n-1} = 0$ nécessite 2^{n-1} inégalités à coefficients dans $\mathbb{R}$ pour être modélisée. Notre méthode consiste donc, en s'inspirant de la théorie des codes correcteurs d'erreur, à changer la matrice $\boldsymbol{A}$ pour diminuer le poids de ses lignes. Les résultats sont assez inégaux d'une couche linéaire à l'autre.

Recherche de différentielles impossibles. Afin de démontrer les capacités de nos nouvelles méthodes de modélisation MILP sur des problèmes complexes, nous avons poursuivi le travail de Sasaki et Todo [ST17b] sur la recherche de différentielles impossibles. Nous prouvons en particulier que toutes les différentielles $(\boldsymbol{a}, \boldsymbol{b})$ avec un octet actif en entrée et un octet actif en sortie sont possibles pour 5 tours d'AES et 13 tours de `SKINNY-128`.

Chapitre 5 – Analyse des formes algébriques normales

Dans ce chapitre, je m'intéresse à la représentation d'un chiffrement comme polynôme à coefficients dans $\mathbb{F}_2$. En effet, toute fonction booléenne $F : \mathbb{F}_2^n \to \mathbb{F}_2$ admet une représentation de la forme

$$F(X) = \sum_{\boldsymbol{u} \in \mathbb{F}_2^n} a_{\boldsymbol{u}} X^{\boldsymbol{u}}, \quad \text{où } X^{\boldsymbol{u}} \stackrel{\text{def}}{=} \prod_{i=0}^{n-1} X_i,$$

appelée forme algébrique normale ou ANF. De plus, le coefficient $a_{\boldsymbol{u}}$ peut être calculé par la transformée de Möbius, exponentielle en le poids de $\boldsymbol{u}$, $\mathrm{wt}\,(\boldsymbol{u})$:

$$a_{\boldsymbol{u}} = \sum_{\boldsymbol{x} \mathrm{Prec}(\boldsymbol{u})} F(\boldsymbol{x}).$$

Une coordonnée d'une primitive de chiffrement peut alors être de la forme $F(X, K)$ où $X_0, \ldots, X_{n-1}$ sont des variables publiques et $K_0, \ldots, K_{m-1}$ sont des variables secrètes. Dans ce cas, l'ANF de F s'écrira

$$F(X, K) = \sum_{\boldsymbol{u} \in \mathbb{F}_2^n} P_{\boldsymbol{u}}(K) X^{\boldsymbol{u}}$$

où les polynômes $P_{\boldsymbol{u}}(K)$ sont appelés *superpoly* et les ensembles $\mathrm{Prec}\,(\boldsymbol{u})$ leurs cubes respectifs, car

$$P_{\boldsymbol{u}}(K) = \sum_{\boldsymbol{x} \in \mathrm{Prec}(\boldsymbol{u})} F(\boldsymbol{x}, K).$$

Attaques *cube*. Certaines attaques algébriques exploitent des structures ou des faiblesses dans les ANF à différents niveaux dans un chiffrement. Parmi elles, les attaques de type *cube* [DS09], surtout appliquées aux générateurs pseudo-aléatoires, cherchent à construire des systèmes polynomiaux de la forme

$$P_{\boldsymbol{u}}(K) = c_{\boldsymbol{u}}, \ \boldsymbol{u} \in \mathcal{U}$$

tels que pour tout $\boldsymbol{u} \in \mathcal{U}$, $c_{\boldsymbol{u}} = P_{\boldsymbol{u}}(\boldsymbol{k})$ où $\boldsymbol{k}$ est la clef secrète, et tels que l'on puisse résoudre ce système, ou à défaut en tirer une information sur la clef secrète. Ces attaques connurent quelques développements avec les testeurs de cubes et les attaques cube dynamiques [Aum+09 ; DS11], mais leur dépendance à la transformée de Möbius limite l'analyse aux cubes de dimension atteignable par les moyens de calcul actuels.

Chemins de monômes. L'élaboration du concept de chemin de monômes, à la suite des travaux de Todo [Tod15] sur les distingueurs intégraux, permirent à Todo et al. [Tod+17] de proposer des attaques cube théoriques bien au-delà des dimensions atteignables en pratique par la transformée de Möbius. Les chemins de monômes suivent l'existence d'un monôme $X^{\boldsymbol{u}} K^{\boldsymbol{v}}$ tout au long des opérations d'une primitive et leur énumération complète permet de calculer la valeur du coefficient de $X^{\boldsymbol{u}} K^{\boldsymbol{v}}$ dans l'ANF de $F(X, K)$.

Dans ce chapitre, après un état de l'art sur les attaques cube et les chemins de monômes, je propose une idée alternative pour analyser la structure d'une ANF. L'idée de base est la suivante : dans un circuit qui permet d'évaluer une fonction booléenne $F : \mathbb{F}_2^n \to \mathbb{F}_2$, le remplacement de l'addition et de la multiplication sur $\mathbb{F}_2$ par les mêmes opérations sur $\mathbb{Z}$ crée un nouvel objet, une

fonction $F^{\mathbb{Z}} : \mathbb{Z}^n \to \mathbb{Z}$, dont l'analyse peut réveler des informations sur l'ANF de F. Je propose en particulier un algorithme pour calculer une ANF avec peu de coefficients non nuls quand on a accès à un circuit d'évaluation.

Chapitre 6 – Étude d'un code à trappe en métrique rang

Ce chapitre, en marge de mon sujet de thèse, reprend un travail effectué avec Alain Couvreur et publié dans *Designs, Codes and Cryptography* en 2020 [CC20]. Il propose une analyse de sécurité d'une fonction à trappe à la McEliece publiée par Pierre Loidreau [Loi17]. Cette fonction à trappe est construite à partir des codes de Gabidulin et leur décodage en métrique rang.

Codes de Gabidulin et métrique rang. Soient m, n tels que $m \geqslant n$ et q une puissance d'un nombre premier. Dans ce contexte, le poids rang d'un vecteur $\boldsymbol{x} \in \mathbb{F}_{q^m}^n$ est la dimension sur $\mathbb{F}_q$ de l'espace engendré par ses coordonnées dans $\mathbb{F}_{q^m}$:

$$|\boldsymbol{x}|_{\mathrm{R}} = \dim \operatorname{vect}_{\mathbb{F}_q}(x_0, \ldots, x_{n-1}).$$

Un code $\mathscr{C}$ est un sous-espace vectoriel de $\mathbb{F}_{q^m}^n$ et sa distance minimale est le plus petit poids rang parmi ses vecteurs non nuls. Soit $\boldsymbol{a} \in \mathbb{F}_{q^m}^n$ de poids rang maximal (égal à n) et $k \leqslant n$. Le code de Gabidulin de support $\boldsymbol{a}$ et de dimension k, $\mathcal{G}_k(\boldsymbol{a})$, est le code engendré par les lignes de la matrice

$$\begin{pmatrix} a_1 & \ldots & a_n \\ a_1^q & \ldots & a_n^q \\ \vdots & & \vdots \\ a_1^{q^{k-1}} & \ldots & a_n^{q^{k-1}} \end{pmatrix}.$$

Une propriété importante des codes de Gabidulin est qu'il existe un algorithme efficace, dit de décodage, qui, étant donnés $\boldsymbol{a}$ et un vecteur $\boldsymbol{y} \in \mathbb{F}_{q^m}^n$, retrouve la décomposition (unique si elle existe) de $\boldsymbol{y}$ comme somme $\boldsymbol{y} = \boldsymbol{c} + \boldsymbol{e}$ avec $\boldsymbol{c} \in \mathcal{G}_k(\boldsymbol{a})$ et $|\boldsymbol{e}|_{\mathrm{R}} \leqslant (n-k)/2$. Au contraire, pour un code tiré aléatoirement et avec la donnée d'une matrice génératrice, le décodage est un problème difficile.

Fonction à trappe de Pierre Loidreau. La primitive de Loidreau exploite justement cet écart important. Soient $\boldsymbol{b}$ de rang plein et $k \leqslant n$. Soient $\lambda \leqslant m$ un paramètre entier, $\mathcal{V}$ un sous-espace sur $\mathbb{F}_q$ de $\mathbb{F}_{q^m}$ et $\boldsymbol{P}$ une matrice à coefficients dans $\mathcal{V}$ et inversible dans $\mathcal{M}_n(\mathbb{F}_{q^m})$. On définit

$$\boldsymbol{G}_{\mathrm{pub}} \stackrel{\mathrm{def}}{=} \boldsymbol{G} \cdot \boldsymbol{P}^{-1} \qquad \text{et} \qquad t \stackrel{\mathrm{def}}{=} \left\lfloor \frac{n-k}{2\lambda} \right\rfloor.$$

La clef publique est alors la paire $(\boldsymbol{G}_{\mathrm{pub}}, t)$ et la clef privée est la décomposition de $\boldsymbol{G}_{\mathrm{pub}}$, $(\boldsymbol{b}, \boldsymbol{P})$. Pour chiffrer un message $\boldsymbol{m} \in \mathbb{F}_{q^m}^k$, on tire un vecteur d'erreur $\boldsymbol{e}$ de rang inférieur à t et on calcule $\boldsymbol{c} = \boldsymbol{m} \cdot \boldsymbol{G}_{\mathrm{pub}} + \boldsymbol{e}$. Pour déchiffrer, on utilise la décomposition de $\boldsymbol{G}_{\mathrm{pub}}$ pour calculer $\boldsymbol{c} \cdot \boldsymbol{P}$ et le décodage efficace pour $\mathcal{G}_k(\boldsymbol{b})$ donne $\boldsymbol{m}$.

Cryptanalyse pour $\lambda = 2$. En s'inspirant des cryptanalyses d'Overbeck [Ove08] contre les systèmes antérieurs fondés sur les codes de Gabidulin, nous proposons un distingueur pour la primitive de Loidreau quand les paramètres λ et k vérifient

$$n\left(1 - \frac{1}{\lambda}\right) + 1 < k < n - \lambda.$$

Nous montrons ensuite comment forger une clef privée à partir de la clef publique pour les paramètres $\lambda = 2$ et k vérifiant la contrainte ci-dessus.

Chapter 1

Modern cryptography

Cryptography is about trust and freedom in the information era. It empowers people with the choice of who exactly they share information with over a network they cannot control. Cryptography aims at creating an ideal world where information only makes sense for its legitimate recipient, where malicious attempts of distorsion are hopeless and where the bond between an identity in the world of humans and a cryptographic one cannot be suspected. For now, it is a major keystone of our trust in the Internet.

1.1 Trusting the Internet

In this section, we first give a schematic view of how the Internet is organized. This is a useful example to illustrate how cryptography is used today by millions of people.

1.1.1 The basics of Internet

A network is a group of two or more devices which share a connection. For example, a *local area network* (LAN) can be built with a few computers all connected to a *network switch* with Ethernet cables. A switch is a device whose goal is precisely to transfer data to and from the connected devices thanks to their unique identifier, the media access control (MAC) address. If all the devices connected through the switch are mere computers, these computers live in closed LAN. If however one of the connected devices is a *router* connected to another switch, the LAN can communicate with the network of this other switch through the router (Figure 1.1). The standard way for connected devices to transfer data is by cutting this data in small pieces called *packets* before independently sending those packets through switches and routers.

Internet stands for *inter*connected *net*works. The major components the Internet connects are called *autonomous systems* (AS). An AS is a large network, i.e. a large group of routers and switches, controlled by a single organization like an Internet service provider (ISP), a company or a public institution. For example in France, the Renater network connects many French universities and research labs. It is also an AS connected to other ASes, thus being a part of the Internet and allowing French universities to access services provided on other ASes. There are many different kinds of ASes: some like Renater connect end users and servers, others like tier 1 networks specialize in physical infrastructure to rent transit... An AS A can connect with another AS B with a direct physical

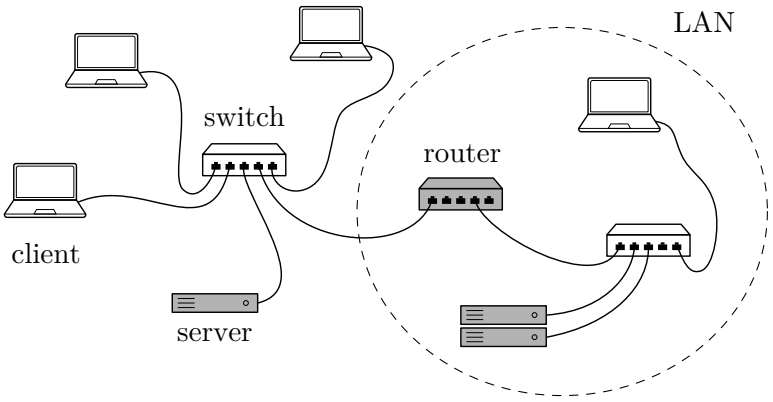

Figure 1.1 – The router creates a network connecting two LANs.

connection or by renting transit or borrowing transit from a third AS C, recursively connected with the AS B. The Internet counted nearly 100 000 ASes worldwide in February 2021 [Mai21]. In order to transfer data from the AS A to the AS B despite this decentralized architecture of the Internet, ASes need to build a map of how they interconnect. They do so with the Border Gateway Protocol (BGP).

A protocol is an agreed upon way of formatting and processing data. It is key to making devices communicate coherently. Every device that connects to the Internet needs an address to locate the device on the map of ASes. The Internet Protocol (IP) is the standard for formatting this address (IP address) and for providing packets sent over a network with source and destination addresses (IP header). Indeed, a router needs the IP addresses to forward a packet to another router in the right direction, but it also needs to know what is the right direction. Edge routers, which connect ASes to others, can get this knowledge through BGP: they publish information about the IP addresses controlled by their AS or about the ASes their AS is connected with. This information helps edge routers from other ASes to build a map of the Internet and to forward packets one step closer to their destination. Another well known protocol is the Transmission Control Protocol (TCP) which is responsible for ordering packets, retransmitting lost packets and checking errors to reconstruct the data after an IP transfer through the network.

Routing denotes the decisions taken by routers to forward packets, to compute a route for them. At the network layer, i.e. the connections between ASes, routing needs the information exchanged with BGP, and routers have no choice but to trust the information is correct. Of course organizations behind ASes can choose to advertize some routes and hide others for commercial, strategic or political purposes, thus distorting the map of ASes. More than that, there are risks of “BGP hijacking” if a router pretends to control IP addresses which belong to another AS. Attackers can reroute traffic to overload a website or to redirect users to a fake website and steal their credentials. This happened in 2018 to steal from cryptocurrency wallets for example [Poi18]. Routing is not the only weak spot of the Internet functioning as other dangers exist: spoofing, DNS leaks...

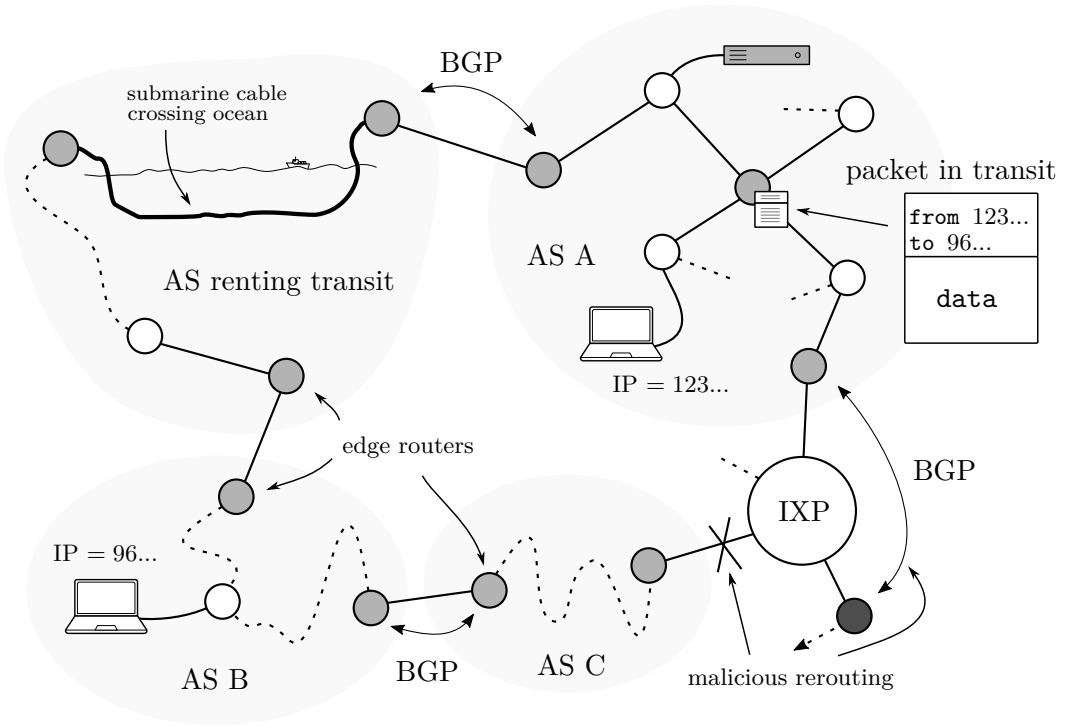

Figure 1.2 – A schematic view of the internet. IXP stands for Internet exchange point, a physical location where ASes can interconnect routers.

1.1.2 Cryptographic protocols

We have just seen that when we send data through the Internet, this data travels through the routers of our own AS, through the routers of other ASes and can even be rerouted to a fake destination. In order to use the Internet with confidence, we want to make sure we connect with the right person or service and we want to make sure our data cannot be read or modified by an attacker who controls the network. Those goals, formalized by the concepts of authentication, confidentiality and integrity, are those achieved by a cryptographic protocol like Transport Layer Security (TLS) [Res18].

Confidentiality is the property that makes the data readable by its legitimate owners and recipients only. It is obtained through *encryption* and algorithms which perform encryption are called *ciphers*. In TLS, the stream should look random to the attacker and should not leak any information but the length of the data. Section 1.3 and Section 1.6 explore ciphers in more details.

Authentication allows users to be confident that no identity has been usurped thanks to *digital signatures*. For instance, TLS always authenticates the server side, which should prevent users from trusting fake websites, and optionnally authenticates the client.

Integrity asserts that the data cannot be modified by an attacker without a legitimate user detecting the modification. Indeed, even if an attacker cannot get any information from an encrypted stream, he could manage to modify the stream and hence its decryption. Integrity is obtained through

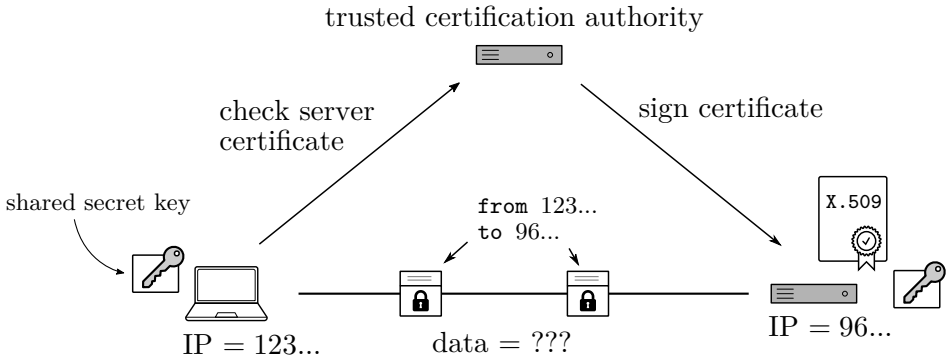

Figure 1.3 – TLS creates a secure channel between the client and the server thanks to the trusted certification authority. Packets of encrypted and authenticated data are transmitted through the network with the classical Internet Protocol.

the authentication *of the data*. It can be performed by algorithms called *message authentication codes* or by digital signatures. Message authentication codes are briefly presented in Section 1.5.2.

The protocol, the cryptosystems and the keys. We see that each goal is fulfilled by an algorithm, a *cryptosystem*. TLS is the protocol which specifies which cryptosystems to use, how to use them and how to handle the data associated with those cryptosystems. Building secure protocols and implementing them without flaws raise difficult questions and form an active field of research. Thanks to a correct use of well-studied cryptosystems, TLS creates a secure channel between a client and a server: it authenticates the server for the client and encrypts and authenticates the data that will be sent through TCP/IP (Figure 1.3). This way, an attacker with complete control of the network cannot inject forged data into the stream nor distinguish the data stream from random bytes. But TLS is an industrial standard used all over the Internet and the attacker is well aware of the tiniest details of both the protocol and the cryptosystems. It might seem counter-intuitive that such a public system can provide any kind of security given that security should require some sort of secrecy somewhere. This problem is tacled with the use of *cryptographic keys*: although cryptosystems have public specifications, they can carry out their cryptographic duties as long as a short parameter, the key, stays secret. We will elaborate on the security of systems with public specifications in the next section.

In its specification [Res18], the state-of-the-art version of TLS (1.3) is cut in two protocols.

The handshake is the first part of TLS where the client and the server are securely introduced to one another and where decisions on how to secure the following data stream are taken. The handshake already needs communication through the network and an attacker should not be able to have any influence or get any meaningful information from this preliminary communication. During the handshake, the client and the server perform a *key exchange*, i.e. they build a shared secret key together (Section 1.6), choose a cipher suite for protecting the data, exchange signed certificates and check them thanks to a trusted third party.

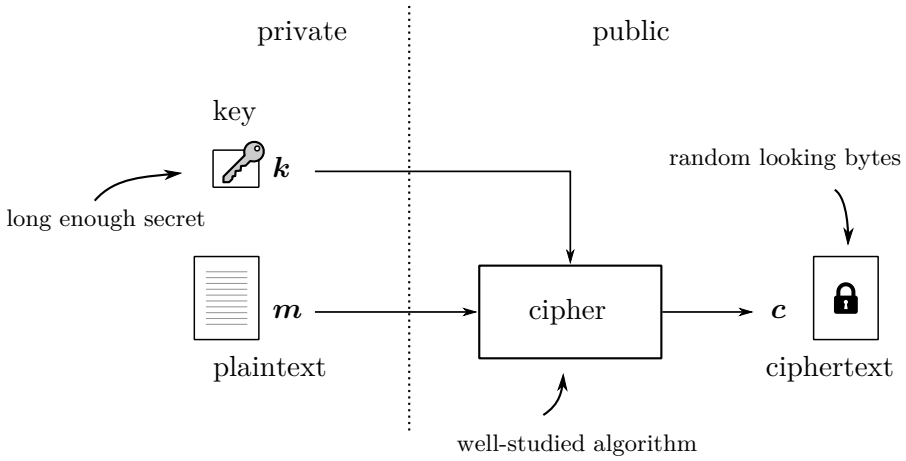

Figure 1.4 – Cryptosystems have public specifications and can be used by anyone. Their security relies on the secret key, only known by legitimate users.

The record protocol encrypts and authenticates the data exchanged between the client and the server with the cipher suit agreed in the handshake. Authenticated encryption is briefly presented in Section 1.5.3.

1.2 Cryptosystems, security notions and primitives

As we saw with the example of TLS, there are different cryptographic needs — confidentiality, integrity and authentication — and each one of those needs will be met with different algorithms. Now the question is how to build these cryptosystems? In this section, we draw a general picture of how modern cyptosystems are designed and analysed by their dedicated scientific community. We first present in Section 1.2.1 the most basic principles about the design of modern cryptosystems. We then explain the central concept of security notion in Section 1.2.2. Finally, we sketch in the other sections how cryptosystems are built on two pillars: primitives and computational assumptions.

1.2.1 Foundations

Kerckhoffs' principles. We explained in the previous section that the security of public cryptosystems relied on secret keys. This idea was already present in the foundational work of Auguste Kerckhoffs in 1883 [Ker83]. Kerckhoffs listed requirements for the design of ciphers, known today as Kerckhoffs' principles. In particular, the first two principles state that a cipher should be mathematically impossible to break even for someone who knows its design. As we saw, cryptographers abide by those principles by moving the needed secrecy from the whole specification of the cipher to a parameter of the cipher called the *key* (Figure 1.4). Other principles insist that the key should be short enough to be remembered and changed, and to make the system easy to use.

Computational security. Claude Shannon gave in 1949 [Sha49] a precise meaning of "mathematically impossible to break" with the introduction of his new *information theory*, now at the heart of modern cryptography. In particular, Shannon proved that a cipher which is perfectly secure needs as much key material as message material, which is very impractical and in contradiction with

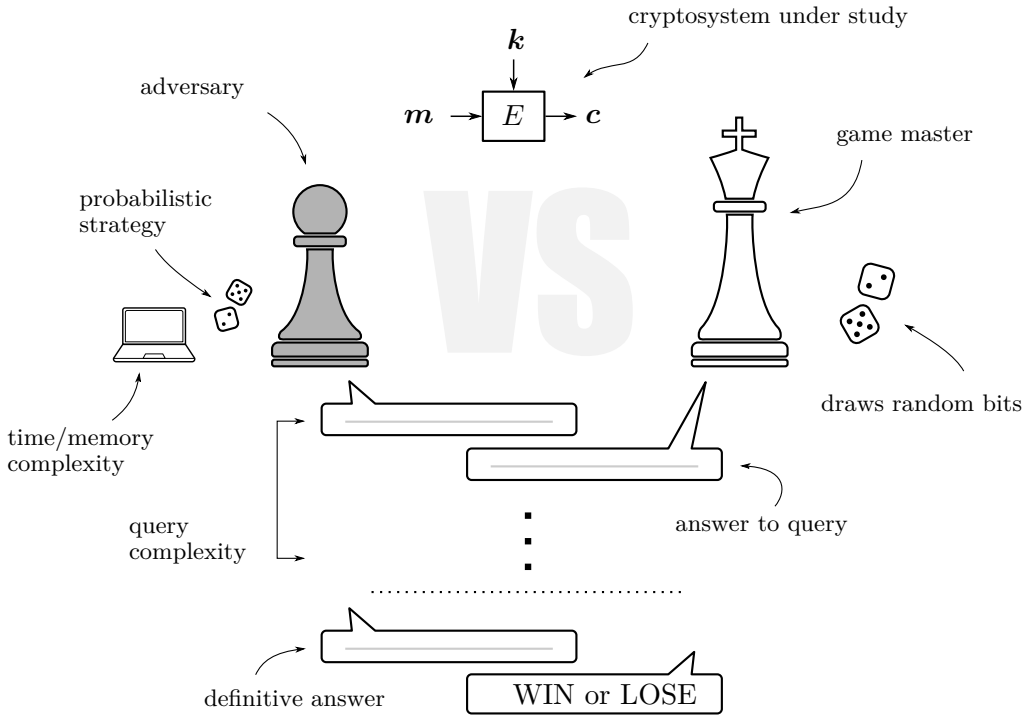

Figure 1.5 – A security game has rules to explain how the game master behaves and how much power the adversary is given, i.e. which queries to the game master are allowed.

Kerckhoffs' principles. As an example, Shannon proved that the one-time pad (see Section 1.3.1) achieves perfect security under this condition. Cryptographers then focus on designing *practically* impossible to break ciphers: an attacker has to use huge computational resources to break such a cipher. For example, if a key is chosen at random among 128-bit words, guessing the key needs an average number of 2^{127} trials with one cipher evaluation each, which is way beyond what the most powerful computers can do. Then a cipher will typically be designed to be unbreakable unless the key has been guessed, which we saw is practically impossible. The expression "break a cryptosystem" we used so far still needs to be defined.

1.2.2 Security notions

To precisely define the security of cryptosystems, cryptographers agree on worst-case scenarios where cryptosystems still have to ensure that the cryptographic goal holds. They then derive criteria and reasonings to build systems which resist a powerful adversary in these worst-case scenarios. In the computational model, where consequences of the system's implementation — like computation times — are not taken into account, a *security notion* is precisely a scenario where an attacker is challenged to threaten a cryptosystem with his computational power only. A security notion is formally defined by the rules of a one player game with only two possible outcomes: win or lose. The player, called the *adversary*, is modeled by a probabilistic Turing machine. The game usually has a game master (or challenger) whose role mainly consists in drawing bits at random, eventually keeping them secret, performing computations with those random bits and calling available oracles (see Section 1.2.4). Finally, the cryptosystem is a parameter of the game (Figure 1.5). This section, which summarizes

important aspects of security notions, is inspired by the teaching of Pointcheval [Poi05; Poi21].

Adversaries. When a cryptographer studies how a system performs against a security notion, he essentially studies the possible strategies to win the game with the given cryptosystem. Since an adversary is a probabilistic algorithm that interacts with the game, she has the following characteristics:

- a probability of success,
- complexities in time and memory,
- complexities in interactions with the game (queries sent to the challenger, answers received from the challenger, queries to available oracles...)

The time complexity is the entire number of operations, including those needed to compute answers to queries, and thus dominates all other types of complexities.

Computational games. Some security games — like key-recovery games — pose a *computational* challenge to the adversary: the adversary has to compute a value — like a secret key — among many possibilities. The success probability is then a good measurement for the performance of the adversary. The maximum success probability an adversary can get for a given complexity is denoted by $\mathrm{Succ}^{\text{name of the game}}_{\text{cryptosystem}}(\text{complexity})$.

Decisional games. Other security games — like distinguishing games — pose a *decisional* challenge to which the adversary has to answer "yes" or "no", knowing that both answers have the same probability $\frac{1}{2}$. In that case, a trivial attack with success probability $\frac{1}{2}$ is to answer "yes" or "no" uniformly at random. Therefore, to measure the performance of an adversary, cryptographers estimate the distance between her success probability and $\frac{1}{2}$. This distance is called the *advantage* of the adversary:

$$\mathrm{Adv}^{\text{name of the game}}_{\text{cryptosystem}}(\text{adversary}) \stackrel{\text{def}}{=} \left|2 \cdot \mathbb{P}\left(\text{adversary outputs the right answer}\right) - 1\right|.$$

We can always turn an algorithm with success probability lower than $\frac{1}{2}$ to an algorithm with success probability higher than $\frac{1}{2}$ by systematically reversing its output. The absolute value in the definition of the advantage is therefore superficial. The maximum advantage an adversary can get for a given complexity is denoted by $\mathrm{Adv}^{\text{name of the game}}_{\text{cryptosystem}}(\text{complexity})$.

Probabilities and complexities. The success probability (or the advantage) and the complexities can balance: an adversary can repeat the same strategy (with independent randomness) to increase the success probability but this will also increase the complexities. This is not a problem in practice and in general, when cryptographers only talk about the complexity of a strategy, its success probability is implicitly close to 1.

Names for security notions. As hinted in the informal definitions above, security notions have names. In fact, they are usually named after two characteristics.

- The goal of the adversary: what she has to compute or decide.

- The allowed queries of the adversary to the challenger.

For example, when studying encryption schemes, a typical goal is to recover the decryption key and it is common to choose whether the adversary can query the decryption of chosen ciphertexts when the game master knows the decryption key. In this setting, the name could be "key-recovery under adaptively-chosen ciphertexts", condensed to KR-ACC.

Cryptanalysis. For a given cryptosystem and a security notion, the term *cryptanalysis* can refer to the search for a strategy or to the found strategy itself. Obviously, exhibiting a cryptanalysis gives an upper bound on the complexity of the best possible strategy (or a lower bound on the success probability).

Since cryptosystems operate on finite data, there always exists a strategy to win the game with good probability and its complexity depends directly on the size of the data, especially on the size of the key material. This strategy might even be independent from the specific cryptosystem under study, in which case it is called *generic*. For example, if the game master gives a pair plaintext/ciphertext and the goal is to find the key, trying all keys is a generic strategy whose expected complexity is $2^{\text{size of the key}-1}$ queries.

Now for each cryptosystem, or family of cryptosystems, cryptographers agree on a *security threshold*. It defines the complexity under which a cryptanalysis is considered to break the cryptosystem and such a cryptanalysis is called an *attack*. The security threshold can be expressed in two equivalent ways: we can give the plain complexity t but cryptographers like using $\log_2(t)$ which they call the number of *bits of security* of the system. This definition remains theoretical as an attack could still have an unpractical complexity. Indeed, typical numbers of bits of security are 64, 80, 128 and 256.

For a symmetric-key cryptosystem, the threshold is often quite natural as it is the complexity of the best generic strategy (which is often guessing the key). For a public-key cryptosystem, things are a bit more complicated as generic strategies are much less obvious to determine, as well as their link with the key size.

1.2.3 Proofs and conjectures

We saw in the previous section that cryptographers agree on worst-case scenarios and formalize them into security games. Of course what they want to do now is to prove that with the cryptosystems they design, attacks cannot be found and adversaries with good complexities and success probabilities do not exist. Unfortunately, cryptographers never found how to design both practical and *provably* secure cryptosystems in the computational model [1]. However, they have invented design strategies to build practical systems with a *conjectured* security. They built a science for designing real-world cryptosystems for which they both have no complete proof of security and no found attack despite their best efforts to overthrow this status quo.

There are schematically two paradigms for studying the security of a cryptosystem (for a given security notion).

1. Relentlessly try to find attacks to stay convinced the system is secure.

[1] In this sentence, we consider the one-time pad as impractical and other information-theoretically secure schemes — like the one given in [AR99] — as impractical because they need a big shared source of randomness.

2. Prove that an attack on the cryptosystem could be used to solve a simpler problem we relentlessly try to solve.

Of course, complex cryptosystems will follow the second paradigm and the underlying problem will be at the heart of the cryptosystem's design. For example, symmetric systems are often built around a *cryptographic primitive* and asymmetric systems are always built around a *computational assumption*.

1.2.4 Primitives

Some cryptosystems are built with a specific architecture and abstract components. When we want to use these systems, concrete algorithms have to instantiate the abstract components: they are called *primitives*. Among them we find hash functions (Section 1.5.1) and block ciphers (Section 1.4.1).

Security of primitives. Cryptographic primitives have security notions as well. The security of a primitive is conjectured and its analysis usually follows the first paradigm. In fact, the computational complexity of a formal proof of security would be of the same magnitude as the complexity of a generic attack. Therefore, if we want the primitive to be out of reach from practical attacks, we have to design it such that it is out of reach from a formal security proof. For instance, the size of a secret key has to be big enough to avoid exhaustive key search. In a nutshell, the security analysis of a primitive is convincing when all known cryptanalysis techniques fail and when the design of the primitive allows to explicit *how far* the primitive is from an attack. This is usually done by finding attacks on more or less simpler versions of the primitive (see "Security margins" in Section 1.4.1 and Section 1.3.4). Decreasing this distance from an attack for existing primitives is a full domain of cryptography to which this thesis is a modest contribution.

The security proof of a cryptosystem can be an unequivoval consequence of the conjectured security of some primitives and a more heuristic consequence for other primitives.

Security reductions. "Unequivocal consequence" means that the performance (the success probability or the advantage) of an adversary against the system's security is bounded by the performance of the best adversaries against the primitives' security. In this case, the proof that the overall cryptosystem is secure often consists in building an adversary against the security notion of the primitive that uses the adversary of the cryptosystem's security game (see Figure 1.6). This looks like an algorithmic reduction but there is a major difference: cryptographers are mostly interested in the computational efficiency of the reduction rather than its asymptotic complexity. For example, an encryption scheme built with a mode of operation is proved secure under the assumption that the block cipher is indistinguishable from a random family of permutations (Section 1.4.4).

Ideal models. "Heuristic consequence" means that the system is proved secure under the assumption that the primitive behaves *ideally*: in the proof, the primitive is completely replaced with an ideal oracle that the adversary and the challenger can query. This kind of abstraction for primitives in security proofs gives less power to the adversary as she cannot benefit from the potential weaknesses of the primitives. Unfortunately, the security proof for a strong cryptosystem should cover powerful adversaries. To explicit this drawback in their proofs, cryptographers make the distinction between the *standard model*, where no such assumption on primitives is made, and the weaker models like the *random oracle model* [BR93] (for hash functions) or the *ideal cipher*

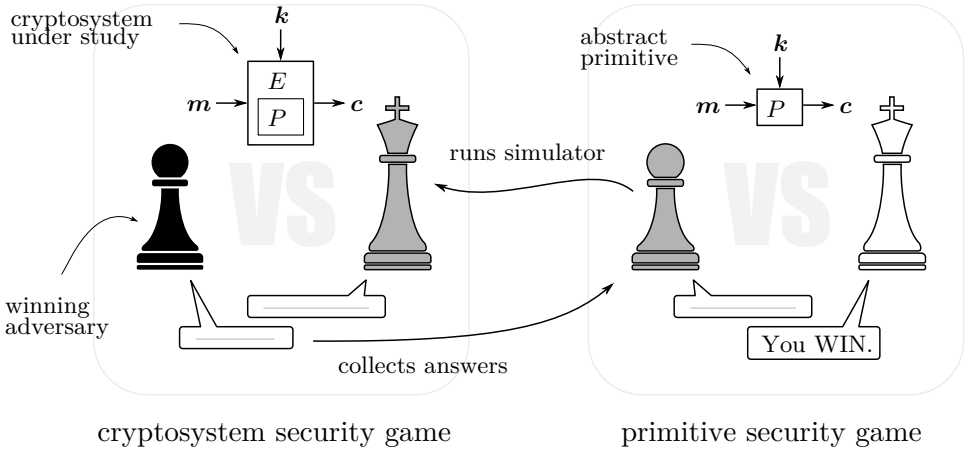

Figure 1.6 – The cryptosystem built with an abstract primitive is studied against a security notion. The corresponding security game is simulated by an adversary of the primitive. The proof that the cryptosystem is secure works by showing that an adversary of the primitive can use an adversary of the cryptosystem to win the primitive security game with a similar advantage. Consequently, if the primitive is conjectured secure, so is the cryptosystem.

model [Sha49] (for block ciphers). However, when a system is only proved secure in a weaker model, its potential weaknesses remain located around the primitives: if an attack is found against the scheme instantiated with a reasonable primitive, changing the primitive for a stronger one may fix it. Using these weaker models essentially comes with trade-offs. Is it worth downgrading to a weaker model to get a security proof for a stronger security notion? Do we prefer an efficient cryptosystem proved secure in a weaker model to a much less efficient one proved secure in the standard model? Those discussions are way beyond the scope of this thesis and a nice overview is given by John Black in [Bla06].

1.2.5 Computational assumptions

The security of asymmetric cryptosystems like signature schemes always relies on an acceptable computational assumption. This assumption may be the fact that a given function is one-way or more frequently the fact that a given bijection is one-way except for those who know a *trapdoor* (see Section 1.6), which is why these systems are called asymmetric. At the basis of the most used public-key cryptosystems like `RSASSA-PSS` [Mor+16] or `ECDSA` [Nisc], notably used in TLS, there are two main families of computational assumptions, each relying on an elementary hard problem.

Factorization. The product $N = p \cdot q$ of two *prime* integers p and q is hard to *factor* when we just know N. In other words, $(p, q) \mapsto N = p \cdot q$ is a one-way function.

Discrete Logarithm. If $\boldsymbol{g}$ generates a group of prime order q and $x \in [0, q-1]$, it is hard to compute the *logarithm* $\boldsymbol{g}^x \mapsto x$: the map $x \mapsto \boldsymbol{g}^x$ is one-way. Of course, the hardness of this problem depends on the particular instantiation of the cyclic group. Common choices are multiplicative subgroups of finite fields and subgroups of elliptic curves.

The proof that the overall cryptosystem is secure then consists in bounding the performance of an adversary of the system's security by the performance of an algorithm solving the computational

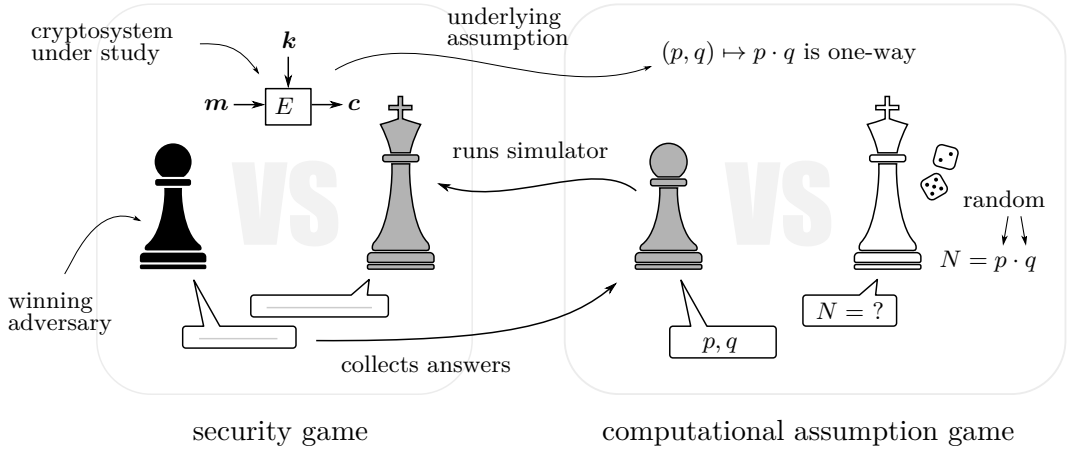

Figure 1.7 – The cryptosystem built upon a computational assumption is studied against a security notion. The corresponding security game is simulated by an adversary of the assumption. The proof that the cryptosystem is secure works by showing that an adversary of the assumption can use an adversary of the cryptosystem to win the game of the assumption with a similar advantage. Consequently, if the assumption holds, the cryptosystem is secure.

problem. This is usually done by building an algorithm that uses the adversary of the security game to solve the computational problem of the assumption (Figure 1.7). This is very similar to the way cryptosystems are proved secure with abstract primitives in the standard model. Again, the computational cost of the reduction is very important and the fact that the computational assumption holds has to be studied with the same dedication as the security of a primitive.

Cryptosystems can use both a computational assumption and abstract primitives, ideal or not. In that case, the reduction of breaking the computational assumption (or breaking a primitive's security) to an attack on the cryptosystem security usually needs several steps to progressively take the different assumptions into account. Such complex proofs in the computational model need elaborate techniques and are a full domain of modern cryptography.

1.3 Symmetric-key encryption

Confidentiality is the oldest and most natural cryptographic goal. Cryptosystems which fill that goal are called *ciphers* or *encryption schemes* as the process of turning a *plaintext* — the meaningful message — into a *ciphertext* — the scrambled message — is called *encryption*. Since legitimate users usually want to get back the plaintext at some point, there is a necessary inverse operation called *decryption*. A major difference between both operations is that encryption may be — and often is — probabilistic whereas decryption has to be deterministic. Consequently, encryption and decryption are not necessarily inverse bijections mathematically speaking but decryption *undoes* encryption.

There are different security notions to define what a secure encryption is and different algorithms which answer different needs. In this section, we explore the traditional yet useful symmetric-key setting. The symmetric-key setting is defined by an assumption: all legitimate users already share a secret key. The word symmetric comes from the fact that encryption and decryption use the same key: the encryption key and the decryption key are symmetric (Figure 1.8).

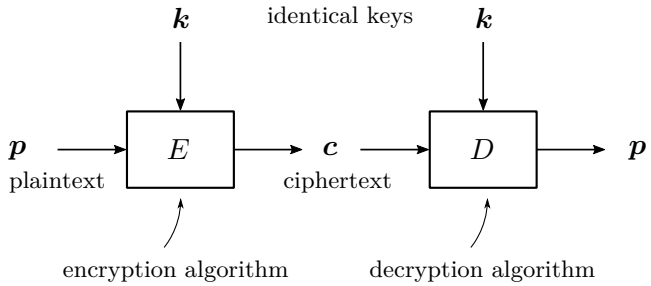

Figure 1.8 – In the symmetric-key setting, the same secret key is used for encryption and decryption.

Definition 1.1 (Symmetric-key cipher)**.** In this setting, a cipher is a pair of algorithms

$$E : \begin{cases} \mathcal{K} \times \mathcal{P} & \longrightarrow & \mathcal{C} \\ (\boldsymbol{k}, \boldsymbol{p}) & \longmapsto & \boldsymbol{c} \end{cases} \quad \text{and} \quad D : \begin{cases} \mathcal{K} \times \mathcal{C} & \longrightarrow & \mathcal{P} \\ (\boldsymbol{k}, \boldsymbol{c}) & \longmapsto & \boldsymbol{p} \end{cases}$$

$$\text{such that} \quad \forall \boldsymbol{k}, \boldsymbol{p},\ D(\boldsymbol{k}, E(\boldsymbol{k}, \boldsymbol{p})) = \boldsymbol{c}.$$

$\mathcal{K}$, $\mathcal{P}$ and $\mathcal{C}$ are respectively the key, plaintext and ciphertext spaces.

The plaintext and the ciphertext spaces are often the same and we will denote them by $\mathcal{M}$, the message space. In this text, we define by default $\mathcal{K} \stackrel{\text{def}}{=} \mathbb{F}_2^m$ and $\mathcal{M} \stackrel{\text{def}}{=} \bigcup_{i=1}^{N} \mathbb{F}_2^i$ where $m, N \in \mathbb{N}$. The key size is fixed to m but messages can have different lengths up to N. Note that E is not a map, in the sense that it can be probabilistic and one input $(\boldsymbol{k}, \boldsymbol{p})$ can give different outputs.

This basic setting gave birth to many ciphers throughout human history but we start with the one-time pad, patented by Vernam in 1919, to introduce stream ciphers and the concept of Initialization Values in Section 1.3.1. Then in Section 1.3.2, we define one-wayness and indistinguishability as the main security notions for symmetric-key encryption (SKE). We finally present dedicated keystream generators, a possible design paradigm for SKE schemes, in Section 1.3.3. Another important design paradigm, based on block ciphers, will be presented in details in Section 1.4.

1.3.1 From the one-time pad to Nonce-based encryption.

The goal of this section is to briefly introduce the role of an Initialization Value IV, or a Nonce N, in a symmetric-key encryption algorithm. We first need to present the one-time pad.

The one-time pad. To encrypt a message $\boldsymbol{p}$, the one-time pad needs a key of the same length. With the above notation, this means that $m \geqslant N$. The ciphertext is then simply $\boldsymbol{c} = \boldsymbol{p} + \boldsymbol{k}_{0,\ldots,|\boldsymbol{p}|-1}$ where the addition is performed in $\mathbb{F}_2$. Decryption and encryption are thus identical. The interesting thing about the one-time pad is that Claude Shannon proved in 1949 that it achieves *perfect secrecy* with *perfectly random* keys [Sha49]. More precisely, this means that knowing the ciphertext cannot improve our knowledge of the plaintext: if the plaintext, the key and the ciphertext are random variables P, K, and C with values in $\mathbb{F}_2^n$ (of same length n) and if the key is chosen uniformly at random — $K \sim \mathcal{U}(\mathbb{F}_2^n)$ — then the plaintext and the ciphertext are independent variables — $\forall \boldsymbol{p}, \boldsymbol{c},\ \mathbb{P}(P = \boldsymbol{p} \,|\, C = \boldsymbol{c}) = \mathbb{P}(P = \boldsymbol{p})$. The problem with the one-time pad is that it needs keys as long as messages themselves and those keys can only be used once, which is very impractical and in contradiction with Kerckhoffs' principles. The idea is then to use the one-time pad with a *keystream generator* rather than a long random key.

Stream ciphers. A keystream generator is an algorithm which turns an initial value IV and a short key $\boldsymbol{k}$ into a *random-looking* keystream $\boldsymbol{s}$ of arbitrary length:

$$S : \left\{ \begin{array}{ccc} \mathcal{K} \times \mathbb{F}_2^\nu & \longrightarrow & \mathcal{M} \\ (\boldsymbol{k}, \mathsf{IV}) & \longmapsto & \boldsymbol{s} \end{array} \right.$$

With $+$ being the addition between vectors of $\mathbb{F}_2$, an encryption algorithm of the form

$$\text{choose } \mathsf{IV}; \quad \boldsymbol{s} \leftarrow S(\boldsymbol{k}, \mathsf{IV}); \quad \boldsymbol{c} \leftarrow \boldsymbol{p} + \boldsymbol{s}; \quad \text{output } (\mathsf{IV}, \boldsymbol{c})$$

is called a *stream cipher*. The IV is chosen first then shared with the rest of the ciphertext. Indeed, since the decryption operation needs to compute the keystream $S(\boldsymbol{k}, \mathsf{IV})$, it needs the IV used for encryption and the encrypted part $\boldsymbol{c}$ has to be associated with its IV. The IV is thus strictly speaking a part of the ciphertext but we will often keep the notation $\boldsymbol{c}$ for the part which depends on the plaintext: $(\mathsf{IV}, \boldsymbol{c}) \in \mathcal{C}$ and $\mathcal{C} = \mathbb{F}_2^\nu \times \mathcal{M}$. Note that the key $\boldsymbol{k}$ is secret and does not change — unless the protocol which uses the cipher specifies otherwise — whereas the IV is public and must change for each encryption.

Indeed, it is important to change IVs for each use of the keystream generator with the same key to avoid scenarios where an attacker tricks the user in encrypting a message of his choice $\boldsymbol{p}^{(1)}$. Indeed, this allows the attacker to collect $(\mathsf{IV}, \boldsymbol{p}^{(1)} + S(\boldsymbol{k}, \mathsf{IV}))$ and hence $\boldsymbol{s} = S(\boldsymbol{k}, \mathsf{IV})$. If the user later encrypts another message $\boldsymbol{p}^{(2)}$ with the same IV, the attacker will know it by comparing IVs and he will remember that this IV gave the keystream $\boldsymbol{s}$. Finally, knowing the keystream will allow him to recover $\boldsymbol{p}^{(2)}$.

Choosing IVs. How to choose the IV should be specified in the cipher specification and, when the cipher follows Definition 1.1, a classical choice is to draw the IV uniformly at random and idependently for each encryption. Some ciphers will need the IV to be chosen this way to be secure but others only need the IV to be never repeated. In this case, the IV is called a Nonce — for number only used once — and Definition 1.1 needs to be adapted.

Definition 1.2 (Nonce-based cipher)**.** A Nonce-based cipher is a pair of *deterministic* algorithms

$$E : \left\{ \begin{array}{ccc} \mathcal{K} \times \mathcal{P} \times \mathcal{N} & \longrightarrow & \mathcal{C} \\ (\boldsymbol{k}, \boldsymbol{p}, \mathsf{N}) & \longmapsto & \boldsymbol{c} \end{array} \right. \quad \text{and} \quad D : \left\{ \begin{array}{ccc} \mathcal{K} \times \mathcal{C} \times \mathcal{N} & \longrightarrow & \mathcal{P} \\ (\boldsymbol{k}, \boldsymbol{c}, \mathsf{N}) & \longmapsto & \boldsymbol{p} \end{array} \right.$$

such that $\quad \forall \boldsymbol{k}, \boldsymbol{p}, \mathsf{N},\ D(\boldsymbol{k}, E(\boldsymbol{k}, \boldsymbol{p}, \mathsf{N}), \mathsf{N}) = \boldsymbol{p}.$

$\mathcal{N}$ is the Nonce space and by default $\mathcal{N} \stackrel{\text{def}}{=} \mathbb{F}_2^\nu$.

1.3.2 Security notions for symmetric-key encryption

Now that we have properly defined SKE schemes, we can present the main security notions against which they will be analyzed: one-wayness and indistinguishability in the chosen-plaintext scenario.

1.3.2.1 One-wayness

One-wayness is one of the most basic security requirements. It means that encryption is hard to invert for someone who does not know the key. In other words, the plaintext should not be recovered

from the ciphertext. One of the security notions which formalizes one-wayness is called OW-CPA, where OW stands for One-Wayness and CPA stands for Chosen Plaintext Attack: in the related security game, the adversary is allowed to query the encryption of plaintexts of his choice to the game master. This power granted to the adversary might seem pessimistic but it captures the above scenario and cryptographers prefer taking extra caution. Moreover, it makes the security notion simpler and easier to study.

In the OW-CPA security game, the challenger first draws a key $\boldsymbol{k}$ and a plaintext $\boldsymbol{p}^*$ uniformly at random. He then encrypts and publishes $\boldsymbol{c}^* = E(\boldsymbol{k}, \boldsymbol{p}^*)$. The goal of the adversary is to compute $\boldsymbol{p}^*$ — this is a computational game — and she is allowed queries $\boldsymbol{p} \mapsto E(\boldsymbol{k}, \boldsymbol{p})$. When the strategy boils down to guessing the key $\boldsymbol{k}$ with the allowed queries to decrypt $\boldsymbol{c}^*$, we call it a *key-recovery* attack.

We dropped the IV in the previous paragraph because when the cipher follows Definition 1.1, the IV management is part of the encryption algorithm and the adversary has no control over the IV. However, in the Nonce variant, the adversary is allowed to choose the Nonce N for each encryption. In this case, the queries will be pairs $(\boldsymbol{p}, \mathsf{N})$ and the adversary will not be allowed to make two queries with the same Nonce and different plaintexts. This rule gives the variant OW-CPA-N and avoids trivial attacks like the one above on the one-time pad with a keystream generator.

The generic attack against OW-CPA is the exhaustive search for the secret key. The adversary sends one query $\boldsymbol{p} \mapsto E(\boldsymbol{k}, \boldsymbol{p}) = (\mathsf{IV}, \boldsymbol{c})$ with $\boldsymbol{p} \in \mathbb{F}_2^n,\ n \geqslant m$ and computes $E(\boldsymbol{k}^{\text{guess}}, \boldsymbol{p}, \mathsf{IV})$ for as many guesses $\boldsymbol{k}^{\text{guess}} \in \mathbb{F}_2^m$ as she can. If the functions $\boldsymbol{p} \in \mathbb{F}_2^n \mapsto E(\boldsymbol{k}, \boldsymbol{p}, \mathsf{IV})$ are different for each $\boldsymbol{k}$, the success probability is 2^{-m} for each guess. Then for any cipher E and for a limited time complexity 2^t:

$$\text{Succ}_E^{\mathsf{OW\text{-}CPA}}\left(2^t\right) \geqslant 2^{t-m}.$$

1.3.2.2 Indistinguishability

Indistinguishability is a much harder requirement for ciphers to fulfill: a cipher satifies indistinguishability when an adversary cannot make the difference between two ciphertexts of two chosen plaintexts. This requires the encryption algorithm to be non deterministic or Nonce-based: when the same message is encrypted twice with the same key, we get different ciphertexts.

Indistinguishability is measured through equivalent decisional games which give the IND-CPA security notion — IND stands for *ind*istinguishability. IND-CPA was formalized for the symmetric-key setting by Bellare, Desai, Jokipii and Rogaway in [Bel+97]. An attack on an IND-CPA game is called a *distinguisher* and its performance is measured through its advantage. We only expose the Real or Random (RoR) flavor but the equivalent security games (Left-or-Right, Find-then-Guess, Semantic Security) and the proofs that they are indeed equivalent can be found in [Bel+97].

The challenger uniformly draws a random bit c (for choice) and a random key $\boldsymbol{k}$ at the beginning of the game. If $c = 0$, the challenger goes into the Real mode (explained below) and if $c = 1$, he goes into the Random mode. The adversary sends requests with a plaintext $\boldsymbol{p}$ to which the challenger answers depending on the chosen mode. In Real mode, the challenger answers queries with a real encryption of $\boldsymbol{p}$. In Random mode, he draws a plaintext $\boldsymbol{p}'$ of the same length as $\boldsymbol{p}$ uniformly at random and answers with the encryption of $\boldsymbol{p}'$. The adversary's goal is to guess the bit c, i.e. whether the game is in real or random mode. The advantage of the distinguisher $\mathcal{D}$ can be computed

with

$$\begin{aligned}\mathrm{Adv}_E^{\mathsf{IND\text{-}CPA}}(\mathcal{D}) &= 2 \cdot \mathbb{P}(\mathcal{D} \text{ outputs } c) - 1 \\ &= \mathbb{P}(\mathcal{D} \text{ outputs } 0 \,|\, c = 0) - \mathbb{P}(\mathcal{D} \text{ outputs } 0 \,|\, c = 1)\,.\end{aligned}$$

There exists a stronger notion called IND$-CPA — $ stands for random — which is not really a requirement but sometimes more convenient for proofs. This notion was first introduced in [Rog+01]. For a given encryption scheme, the IND$-CPA game challenges an adversary to distinguish an encrypted plaintext of his choice from a random vector with the same length as the ciphertext. More precisely, the Real mode is unchanged but in Random mode, the challenger answers with a uniformly random string with the same length as $E(\boldsymbol{k}, \boldsymbol{p})$. Proposition 1.1 formally states that IND$-CPA is stronger than IND-CPA.

Proposition 1.1 (IND$-CPA $\Rightarrow$ IND-CPA)**.**

$$\mathrm{Adv}_E^{\mathsf{IND\$\text{-}CPA}}(t) \geqslant \frac{1}{2} \cdot \mathrm{Adv}_E^{\mathsf{IND\text{-}CPA}}(t)\,.$$

Relations between advantages (or success probabilities) like the one of Proposition 1.1 allow cryptographers to partially order security notions: "IND$-CPA is stronger than IND-CPA", denoted by IND$-CPA $\Rightarrow$ IND-CPA, means that if there exists an attack on IND-CPA then there automatically exists an attack on IND$-CPA. The IND$-CPA security notion is easier to break than IND-CPA and we can get into a situation where the IND$-CPA security of a cipher is broken but IND-CPA is not. The stronger the security notion, the more powerful the adversary is, the easiest it is to break the cipher and the more secure the cipher is when there is no attack.

The generic attack against IND-CPA runs an exhaustive search for the key. If the key is not found, the adversary answers Real or Random uniformly at random.

$$\mathrm{Adv}_E^{\mathsf{IND\text{-}CPA}}\left(2^t\right) \geqslant 2 \cdot \mathrm{Succ}_E^{\mathsf{OW\text{-}CPA}}\left(2^t\right) \geqslant 2^{t-m+1}.$$

The factor 2 comes from the constant in the definition of the advantage.

IND-CPA is stronger than OW-CPA. Indeed, if we can break the one-wayness of an encryption scheme, i.e. recover the plaintext from the ciphertext, we can obviously distinguish the Real mode from the Random mode. Again, we can have the situation where there exists an attack on IND-CPA but not on OW-CPA. In general, decisional games are easier to win (from the adversary's point of view) than computational games and if a cipher has no distinguisher, it can be considered more secure than a cipher with a distinguisher but no key recovery. One might argue that distinguishers break ciphers in rather unrealistic scenarios. First, remember that cryptographers focus on worst-case scenarios and we can imagine a scenario where the adversary does not need to know the exact content of the plaintext but just needs to extract a very small amount of information on the plaintext. Second, security relies on conjectures. Distinguishers usually disclose serious weaknesses that could help to mount key recoveries or other attacks in even more realistic scenarios. Consequently, we can have more confidence in the conjecture that the key-recovery resistance holds when the conjecture on distinguishability resistance also holds.

1.3.3 Dedicated keystream generators

As explained in Section 1.3.1, keystream generators are used to build stream ciphers. Some of them, *dedicated* keystream generators, are designed as *primitives*, in the sense that they have no security proof and that their security analysis follows the first paradigm of Section 1.2.3.

Chosen IV attacks. From the IND$-CPA-N game applied to a stream cipher, we can derive a security notion for keystream generators. More specifically, the chosen plaintext setting — i.e. the allowed queries $(\boldsymbol{p}, \mathsf{IV}) \mapsto E(\boldsymbol{k}, \boldsymbol{p}, \mathsf{IV})$ — and the stream cipher definition — $E(\boldsymbol{k}, \boldsymbol{p}, \mathsf{IV}) = S(\boldsymbol{k}, \mathsf{IV}) + \boldsymbol{p}$ — imply that an adversary has access to the keystream $S(\boldsymbol{k}, \mathsf{IV})$. Besides, the Nonce-based variant implies that the adversary can choose the IV but cannot repeat it. We then get the IND$-CIV game, where the adversary has to distinguish between a given keystream generator S and a random generator which outputs bits uniformly at random with allowed queries $\mathsf{IV} \mapsto S(\boldsymbol{k}, \mathsf{IV})$ in the Real mode. The fact that the adversary cannot repeat the IV can also be modeled in the Random mode with a random oracle which gives the same answer if the same IV is queried. Moreover, we have the weaker notion of the key recovery where the adversary can perform the same chosen IV queries.

In practice. There is a wide variety of design strategies for dedicated keystream generators because of the variety of use cases. Some software-oriented generators like `Salsa20` [Ber08] look like the `CTR` mode (Section 1.4.4.2) and output the keystream in chunks. The differences are that the permutation is unkeyed and public and that the key is mixed with the IV in the counter.

Hardware-oriented generators like `Grain-v1` (Section 1.3.4) usually output keystream bits individually and sequentially. In that case, these keystream bits are computed from the bits of an internal state with a *filtering* function. The internal state is first computed from the IV and the key by an initialization function. During keystream generation, the internal state is updated for each keystream bit by a *transition* function.

Famous stream ciphers with dedicated keystream generators.

- `Salsa20` [Ber08] is a software-oriented member of the eSTREAM portfolio and its variant `ChaCha20` [NL18] is notably used in TLS 1.3.
- `Trivium` [DCP08] from the eSTREAM portfolio for hardware has a 80-bit key and IV and a 288-bit internal state. Notably, `Trivium` has been shown to be a good candidate for homomorphic encryption applications in [Can+18].
- `A5/1` is an encryption algorithm of the GSM standard and its description used to be secret until its reverse-engineering in 1999 by Briceno, Golberg and Wagner [BGW99]. It has a 64-bit key and a 64-bit internal state. These small numbers allow devastating time-memory trade-off attacks [BD00; BSW01] and other attacks exploit some structural weaknesses [EJ03; MJB04; BB06].
- `SNOW 3G` is a standard stream cipher from the 3GPP, an organization in charge of developping protocols for mobile telecommunications [For06]. The key and the IV are both 128-bit long and the internal state is composed of 608 bits.

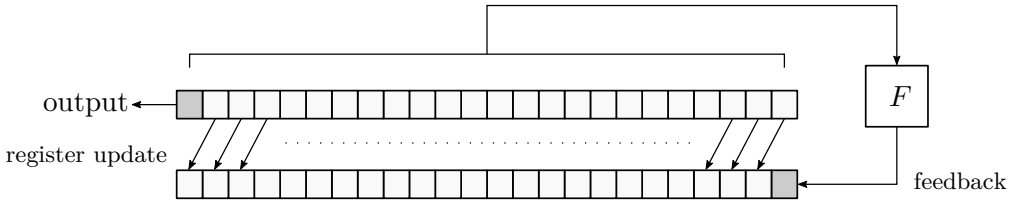

Figure 1.9 – A feedback shift register with a feedback function F.

- `E0` is the stream cipher used to encrypt communications made with the Bluetooth protocol [pro19]. It has a 128-bit key, a 64-bit IV and a 128-bit internal state. The security of `E0` is limited due to several attacks [GBM02; LV04; ZXF13].

Stream ciphers with `CTR`. We will see in Section 1.4.4.2 that we can build a block-cipher-based stream cipher with the `CTR` mode. In that case, the security analysis of the stream cipher follows the second paradigm from Section 1.2.3: we can prove that the security of the scheme holds as long as the underlying primitive, the block cipher, is a good PRP.

1.3.4 Example: the `Grain-v1` stream cipher

The stream ciphers of the `Grain` family have dedicated keystream generators. We give the example of `Grain-v1` which is a hardware-oriented member of the eSTREAM portfolio [Hel+08]. The key size and IV size of `Grain-v1` are respectively 80 bits and 64 bits. The internal state of the generator is composed of two 80-bit feedback shift registers (FSR).

Feedback shift registers. A feedback shift register of length ℓ over $\mathbb{F}_2$ is a ℓ-bit register $\mathtt{r}$ with an output end $\mathtt{r}_0$ and an input end $\mathtt{r}_{\ell-1}$ which is updated with the following steps.

1. Compute a value v depending on the bits of the register with a given *feedback* function $F : \mathbb{F}_2^\ell \to \mathbb{F}_2$: $v = F(\mathtt{r}_0, \ldots, \mathtt{r}_{\ell-1})$;
2. Possibly publish the bit of the output end;
3. Shift all the bits of the register by one towards the output end: $\mathtt{r}_i \leftarrow \mathtt{r}_{i+1}$;
4. Assign the feedback value to the input end bit: $\mathtt{r}_{\ell-1} \leftarrow v$.

Those steps, illustrated by Figure 1.9, define in fact a recurrent sequence $(u_t)_{t\in\mathbb{N}} \in \mathbb{F}_2^{\mathbb{N}}$ whose recurrence relation is given by the feedback function: $u_{t+\ell} = F(u_t, \ldots, u_{t+\ell-1})$. If the feedback function is $\mathbb{F}_2$-linear, we get a linear FSR (LFSR). Otherwise, we have a non-linear FSR (NFSR).

In `Grain-v1`, one register is a LFSR, whose sequence is denoted by $\boldsymbol{s} \in \mathbb{F}_2^{\mathbb{N}}$, and the other one is an NFSR whose sequence is $\boldsymbol{b} \in \mathbb{F}_2^{\mathbb{N}}$. Both registers have to be initialized with the key and the IV before being used for generating the keystream.

Initialization. The register of the NFSR is first filled with the 80-bit key while the register of the LFSR is filled with the 64-bit IV padded with ones.

$$(s_0, \ldots, s_{79}) \leftarrow (\mathsf{IV} \mid \mathbf{1}), \qquad (b_0, \ldots, b_{79}) \leftarrow \boldsymbol{k}.$$

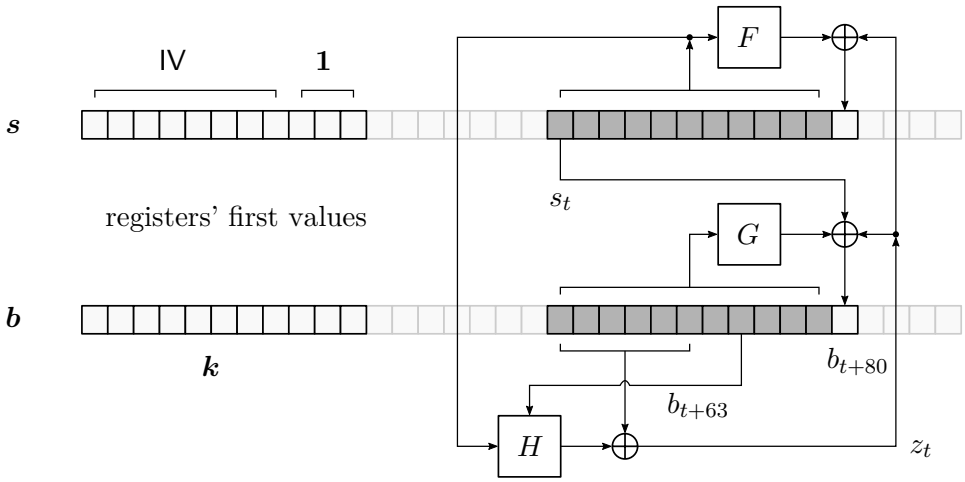

Figure 1.10 – Initialization of `Grain-v1`.

There is a third sequence $\boldsymbol{z} \in \mathbb{F}_2^{\mathbb{N}}$ that we call the output sequence. z_t is derived from $s_t, \ldots, s_{t+79}$ and $b_t, \ldots, b_{t+79}$ with

$$z_t = \sum_{j \in \mathcal{J}} b_{t+j} + H(s_{t+3}, s_{t+25}, s_{t+46}, s_{t+64}, b_{t+63}) \tag{1.1}$$

where $\mathcal{J} = \{1, 2, 4, 10, 31, 43, 56\}$ and $H : \mathbb{F}_2^5 \to \mathbb{F}_2$, called the *filter* function, is defined by:

$$H(\boldsymbol{x}) \stackrel{\text{def}}{=} x_1 + x_4 + x_0x_3 + x_2x_3 + x_3x_4 + x_0x_1x_2 + x_0x_2x_3 + x_0x_2x_4 + x_1x_2x_4 + x_2x_3x_4.$$

The LFSR feedback during this initialization is given by the linear recurrence relation $s_{t+80} = z_t + F(s_t, \ldots, s_{79})$ where

$$F(\boldsymbol{x}) = x_{62} + x_{51} + x_{38} + x_{23} + x_{13} + x_0.$$

The NFSR feedback is given by $b_{t+80} = s_t + z_t + G(b_t, \ldots, b_{t+79})$ where

$$G(\boldsymbol{x}) = \sum_{d=1}^{6} G_d(\boldsymbol{x}), \qquad G_d(\boldsymbol{x}) = \sum_{(i_1,\ldots,i_d) \in \mathcal{I}_d} \prod_{j=1}^{d} x_{i_j}$$

and

$$\begin{aligned}
&\mathcal{I}_1 = \{0, 9, 14, 21, 28, 33, 37, 45, 52, 60, 62\}, && \mathcal{I}_2 = \{(9, 15), (33, 37), (60, 63)\},\\
&\mathcal{I}_4 = \{(15, 21, 60, 63), (33, 37, 52, 60), (9, 28, 45, 63)\}, && \mathcal{I}_3 = \{(21, 28, 33), (45, 52, 60)\},\\
&\mathcal{I}_5 = \{(9, 15, 21, 28, 33), (37, 45, 52, 60, 63)\} \text{ and} && \mathcal{I}_6 = \{(21, 28, 33, 37, 45, 52)\}.
\end{aligned}$$

Both FSRs are clocked 160 times to mix the key and the IV and the output stream (z_t) is not published. After these 160 clocks, the registers are properly initialized with the values $s_{160}, \ldots, s_{239}$ and $b_{160}, \ldots, b_{239}$.

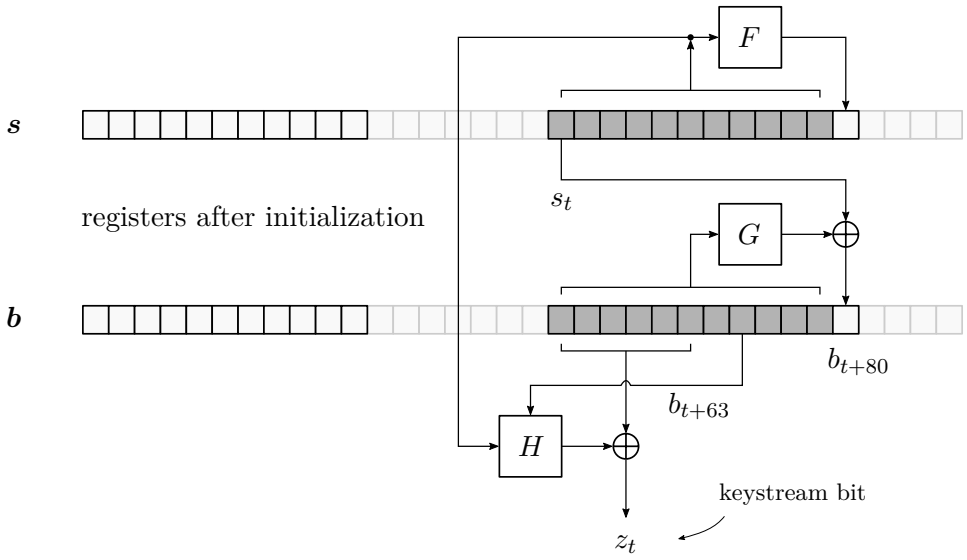

Figure 1.11 – Computing the keystream $(z_t)_{t\in\mathbb{N}}$ for `Grain-v1`.

Keystream generation. Once the registers have been initialized, generating the keystream $\boldsymbol{z}$ is very similar to generating the output stream of the initialization. Indeed, the only change is in the recurrence relations of $\boldsymbol{s}$ and $\boldsymbol{b}$: the output bit z_t does no longer contribute to s_{t+80} and b_{t+80}. More precisely,

$$\begin{aligned} & s_{t+80} = F(s_t, \ldots, s_{79}) \\ \text{and} \quad & b_{t+80} = s_t + G(b_t, \ldots, b_{t+79}). \end{aligned}$$

Security margin. The security of a keystream generator like the one of `Grain-v1` mainly comes from the *number of initialization clocks*. Indeed, the higher this number, the more complicated the first keystream bits. The security margin of a dedicated keystream generator is then given by the difference between the number of initialization clocks of the design and the highest number of clocks after which publishing the output stream allows to attack it. Of course, as for the number of rounds of block ciphers, the performance is impacted by the chosen number of initialization clocks.

1.4 Block-cipher-based encryption

In this section, we present a popular way of building SKE schemes: the combination of a *block cipher* and a *mode of operation*. The block cipher is usually a primitive with a conjectured security and the mode of operation specifies how to use this primitive to get the full SKE scheme as defined in Definitions 1.1 and 1.2.

1.4.1 Block ciphers

A block cipher is a primitive which can be seen as a deterministic (no IV) encryption scheme for fixed-size plaintexts and ciphertexts:

$$B : \left\{ \begin{array}{rcl} \mathbb{F}_2^m \times \mathbb{F}_2^n & \longrightarrow & \mathbb{F}_2^n \\ (\boldsymbol{k}, \boldsymbol{p}) & \longmapsto & \boldsymbol{c}, \end{array} \right.$$

where n is called the *block size* and is typically chosen in $\{64, 128, 256\}$. Once a key $\boldsymbol{k}$ is chosen, the map $\boldsymbol{p} \mapsto B(\boldsymbol{k}, \boldsymbol{p})$, sometimes denoted by $B_{\boldsymbol{k}}$, is a permutation of $\mathbb{F}_2^n$ because the block cipher encryption has to have the inverse operation of decryption:

$$B^{-1}\left(\boldsymbol{k}, B(\boldsymbol{k}, \boldsymbol{p})\right) = \boldsymbol{p}.$$

Ideal ciphers. Mathematically, a block cipher is a family of 2^m permutations indexed by the keys: $(B_{\boldsymbol{k}})_{\boldsymbol{k} \in \mathbb{F}_2^m}$ where $\forall \boldsymbol{k},\ B_{\boldsymbol{k}} : \mathbb{F}_2^n \to \mathbb{F}_2^n$ is bijective. The ideal block cipher with block size n and key size m is then a family taken uniformly at random among the $(2^n!)^{2^m}$ possible families of permutations [Sha49]. The ideal block cipher would then need a description with size

$$\log_2\left((2^n!)^{2^m}\right) = 2^m \sum_{i=2}^{2^n} \log_2(i) \geqslant 2^{m+n} \text{ bits.}$$

However, a real-world block cipher is an algorithm which has to be implemented and thus cannot have such a huge description. When a proof needs the assumption that a block cipher is ideal, which we just saw is a very strong assumption, we say that the proof is valid in the *ideal-cipher model*.

Pseudo-random permutations. For proofs in the standard model, we want block ciphers to be pseudo-random permutations (PRP). A pseudo-random permutation is a keyed family of permutations which satisfies the PRP security notion. The game for a block cipher B is as follows. The challenger draws uniformly at random a Real or Random mode $c \in \mathbb{F}_2$. In Real mode, he draws uniformly at random $\boldsymbol{k} \in \mathbb{F}_2^m$ and answers queries with $\boldsymbol{p} \mapsto B(\boldsymbol{k}, \boldsymbol{p})$. In Random mode, he draws a permutation $R : \mathbb{F}_2^n \to \mathbb{F}_2^n$ uniformly at random and answers queries with $\boldsymbol{p} \mapsto R(\boldsymbol{p})$. This game is similar to IND$-CPA but adapted to permutations. There is a stronger security notion: sPRP where s stands for strong. In the sPRP game, the adversary is also allowed to access the inverse permutation. In other words the additional possible queries in Real and Random modes are respectively $\boldsymbol{c} \mapsto B^{-1}(\boldsymbol{k}, \boldsymbol{c})$ and $\boldsymbol{c} \mapsto R^{-1}(\boldsymbol{c})$. These games are decisional games but we can also study block ciphers against classical key-recovery games.

In practice: iterative ciphers. For a real-world block cipher, security in the PRP game is one primary concern but *efficiency* is also a major requirement. One popular way of achieving efficiency is to apply several key-dependent permutations $T^{(i)}$, called *round transformations*, to the input. Indeed, round transformations are chosen to be efficient and easily implementable and if their individual security is usually very weak, composing several well-chosen round functions should provide strong security. An iterative block cipher is then a construction of the form

$$B(\boldsymbol{k}, \boldsymbol{p}) = T^{(r)}_{\boldsymbol{k}^{(r)}} \circ \cdots \circ T^{(1)}_{\boldsymbol{k}^{(1)}}(\boldsymbol{p}),$$

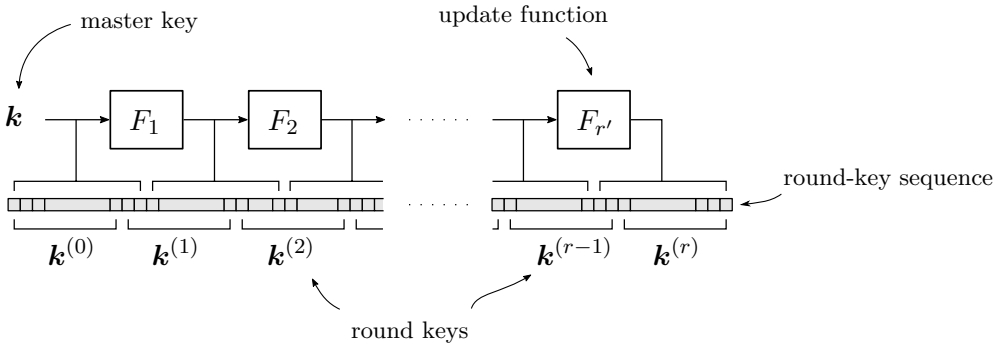

Figure 1.12 – In this key-expansion design, the master key is first loaded in a register. Successive applications of the *update* functions $F_i : \mathbb{F}_2^m \to \mathbb{F}_2^m$ on this register produce a keystream. The bits of the round keys directly use the bits of the keystream.

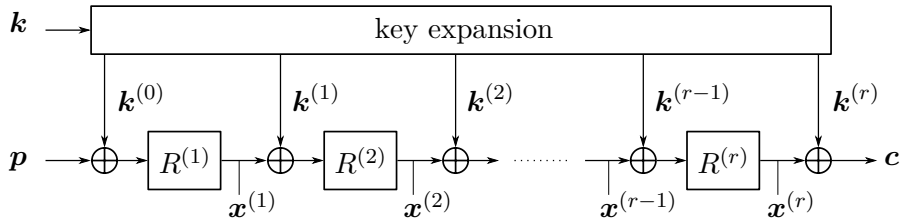

Figure 1.13 – The key-alternating architecture.

where the $\boldsymbol{k}^{(i)}$, $i \in [1, r]$ are called *round keys*. Round keys are derived from the master key $\boldsymbol{k}$ through a *key-expansion* algorithm, also called the *key schedule*. A typical key-expansion architecture is given in Figure 1.12. The *number of rounds* r is usually chosen by the designers after a careful security analysis of the construction. Finally, is important for the maps $\boldsymbol{x} \mapsto T^{(i)}_{\boldsymbol{k}^{(i)}}(\boldsymbol{x})$ to be different to avoid slide attacks [BW99], but the transformations $(\boldsymbol{k}, \boldsymbol{x}) \mapsto T^{(i)}_{\boldsymbol{k}}(\boldsymbol{x})$ can be the same, in which case the block cipher is called *iterated*.

Key-alternating ciphers. Among iterative block ciphers we find the family of key-alternating ciphers. As illustrated in Figure 1.13, a key-alternating block cipher first runs a key-expansion algorithm to derive enough *round keys* $\boldsymbol{k}^{(i)}$ from the master key $\boldsymbol{k}$. It then loads the plaintext $\boldsymbol{p}$ into the *state* $\boldsymbol{x}$ and updates the state alternatively with adding a round key or applying the round functions $R^{(i)}$. The round transformations are then $T^{(i)}(\boldsymbol{x}) = R^{(i)}(\boldsymbol{x}) + \boldsymbol{k}^{(i)}$ — or $R^{(i)}\left(\boldsymbol{x} + \boldsymbol{k}^{(i-1)}\right)$ — and the round keys $\boldsymbol{k}^{(0)}$ and $\boldsymbol{k}^{(r)}$ are called *whitening* keys. It is very common for round functions to be the same — i.e. to combine the key-alternating and the iterated constructions — for better hardware performance and easier security analysis.

SPN ciphers. The heuristic criteria to design round transformations, given by Shannon in [Sha49], are *confusion* and *diffusion*.

- Confusion means that the relation between the output, the input and the key should be as complex as possible.

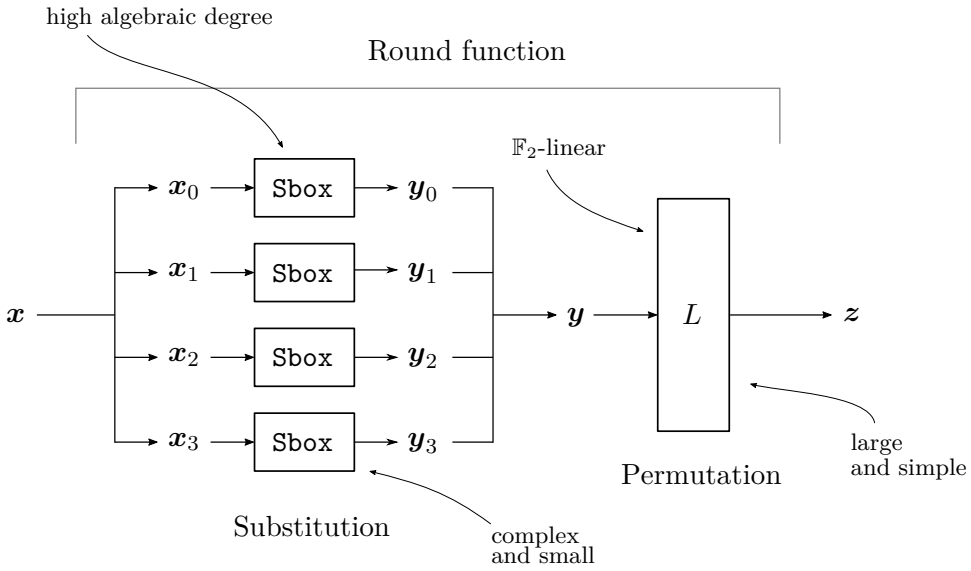

Figure 1.14 – Round function of an SPN cipher.

- Diffusion means that each ciphertext bit should be highly dependent on each plaintext bit and each key bit. Typically, changing only one plaintext bit should output a very different ciphertext.

SPN block ciphers [DR01b] are key-alternating ciphers for which the round functions perform two different steps: one providing confusion and the other providing diffusion. The substitution step applies local confusion thanks to small permutations called Sboxes. The permutation step applies a broad and efficient diffusion with a $\mathbb{F}_2$-linear bijection. The generic round function of an SPN cipher is illustrated in Figure 1.14 and a simple typical SPN block cipher, `PRESENT`, is described in details in Section 1.4.2 as an example.

Famous block ciphers.

- The Data Encryption Standard (DES) [Nisb] was designed by IBM in the 1970s with an iterated Feistel architecture. Its block size is 64 and its key size is 56. This low key size makes it badly vulnerable to brute-force key-recovery attacks: Diffie and Hellman already argued in that sense in 1977 [DH77] and in 1998, the nonprofit Electronic Frontier Foundation built a dedicated machine which could perform key recoveries on DES in less than three days [Fou98]. The DES has been definitively withdrawn by the NIST in 2005 but it is still used as a component of a derived block cipher, Triple DES (3DES or TDEA) [BM17].

- The Advanced Encryption Standard (AES) [Nisa; DR01a] was designed by Joan Daemen and Vincent Rijmen in 1997 as a part of the submission called `Rijndael`. It is designed with an SPN architecture, has block size 128 and exists in three versions for key sizes 128, 192 and 256. At the end of an open competition, in 2000, Rijndael was chosen by the NIST as a successor to the DES among 15 proposals. A description of the AES is given in Section 1.4.3.

Table 1.1 – Value table of the 4-bit Sbox of PRESENT. Values are given in hexadecimal notation: $\mathtt{a} = (0, 1, 0, 1)$ for example.

$\boldsymbol{x}$	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
Sbox($\boldsymbol{x}$)	c	5	6	b	9	0	a	d	3	e	f	8	4	7	1	2

Security margin. The security of a key-alternating block cipher like the AES comes from the complementarity between the strength of the components (Sbox, linear layer, update function in the key schedule) and the *number of rounds* applied to the state where the plaintext is loaded. This implies that after choosing components, designers have to find the right balance for the number of rounds: too many rounds can result in poor performance and not enough rounds can result in broken security. No proof can give an upper bound on the necessary number of rounds. For example, if the round functions were linear, any number of rounds would fail to give any security. Instead, cryptographers establish the *security margin* of the block cipher by exhibiting an attack (i.e. a distinguisher or a key-recovery) on the highest number of rounds possible. If an attack breaks a number of rounds close to the choice of the designers, the cipher should be revised or discarded.

1.4.2 Example: the PRESENT block cipher

The block cipher PRESENT was proposed by Bogdanov et al. in 2007 [Bog+07] as a first attempt to design an ultra-lightweight block cipher. Its block size is 64 bits and its key size is either 80 either 128 bits depending on the variant. It is a key-alternating block cipher which follows an SPN construction.

PRESENT has 31 rounds and the substitution step in the round function applies the same 4-bit Sbox (see Table 1.1) on 16 different chunks of the state. The linear layer is very simple as it is a bit permutation on the full 64-bit state. More precisely,

$$\boldsymbol{y} = P(\boldsymbol{x}) \iff \forall i, j, k \in [0, 3],\ y_{16k+4i+j} = x_{16i+4j+k}.$$

Consequently, the round functions are identical and difference between rounds comes from the key schedule (i.e. the key-expansion algorithm).

The key schedule of PRESENT follows the classical design of Figure 1.12 except the round-key sequence is only composed of the 64 bits $(k_0, \ldots, k_{63})$ for each round whereas the update function works on 80 bits. The update functions for the 80-bit key variant are given by the following steps

1. $(k_0, \ldots, k_{79}) \leftarrow (k_{19}, \ldots, k_{79}, k_0, \ldots, k_{18})$;
2. $(k_{76}, \ldots, k_{79}) \leftarrow \mathtt{Sbox}\,(k_{76}, \ldots, k_{79})$;
3. $(k_{15}, \ldots, k_{19}) \leftarrow (k_{15}, \ldots, k_{19}) \oplus i$ where i is the round index written on five bits ($i \in [1, 31]$).

An overall schema of PRESENT is given in Figure 1.15.

1.4.3 The Advanced Encryption Standard

We briefly introduced the AES in Section 1.4.1 as the block cipher standardized by the NIST in 2001 [Nisa]. For all its standardized modes of operation, the NIST recommends the AES as the first choice, which has made the AES the most used block cipher for symmetric-key encryption schemes.

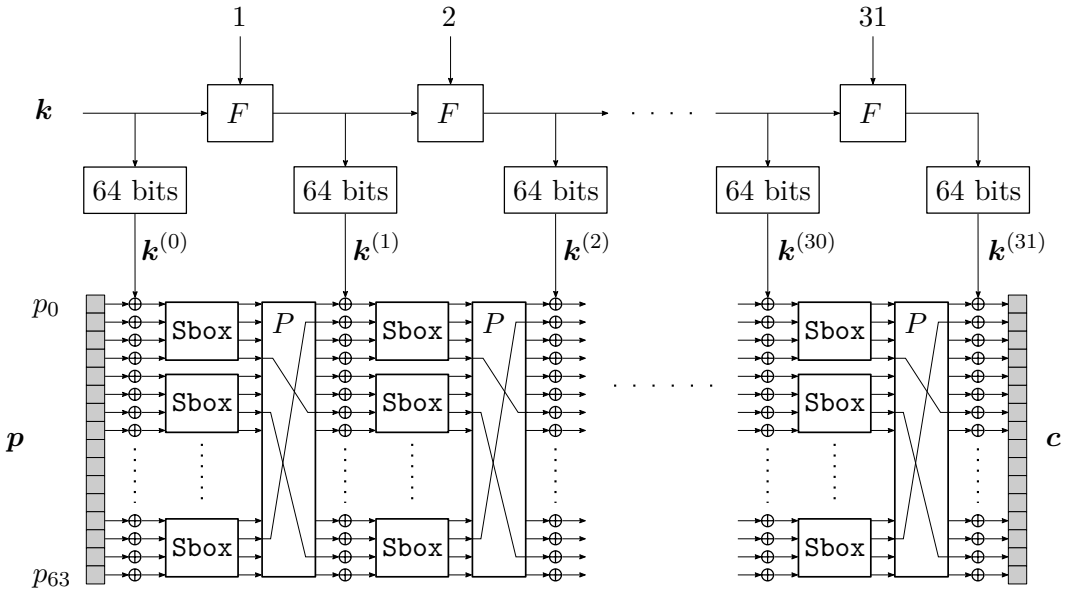

Figure 1.15 – A schematic view of the PRESENT block cipher.

The AES follows the SPN construction described in Section 1.4.1. It has a block size of 128 bits and its number of rounds r varies with the possible key sizes.

Key size	128	192	256
r	10	12	14

The 128-bit state is often represented as a 4×4 matrix of bytes. In other words, we consider depending on the context that the state is an element

$$\boldsymbol{x} = \begin{pmatrix} x_0 & x_4 & x_8 & x_{12} \\ x_1 & x_5 & x_9 & x_{13} \\ x_2 & x_6 & x_{10} & x_{14} \\ x_3 & x_7 & x_{11} & x_{15} \end{pmatrix} = \begin{pmatrix} x_{0,0} & x_{0,1} & x_{0,2} & x_{0,3} \\ x_{1,0} & x_{1,1} & x_{1,2} & x_{1,3} \\ x_{2,0} & x_{2,1} & x_{2,2} & x_{2,3} \\ x_{3,0} & x_{3,1} & x_{3,2} & x_{3,3} \end{pmatrix} \in \mathcal{M}_4(\mathbb{K}) \text{ where } \mathbb{K} \stackrel{\text{def}}{=} \mathbb{F}_{2^8}.$$

To describe the components (Sbox, linear layer, update function) of the AES, we need to fix the field multiplication of $\mathbb{F}_{2^8}$. It is defined by the irreducible polynomial $P(X) \stackrel{\text{def}}{=} X^8 + X^4 + X^3 + X + 1$ over $\mathbb{F}_2$: $\mathbb{F}_{2^8} \stackrel{\text{def}}{=} \mathbb{F}_2[X]/P(X)$. The vector $\boldsymbol{x} \in \mathbb{F}_2^8$ and the field element $x \in \mathbb{F}_{2^8}$ are linked through the isomorphism $\boldsymbol{x} \mapsto \sum_{i=0}^{7} x_i X^i$. Moreover, we define the notation $2 \stackrel{\text{def}}{=} X$ and then $3 = X + 1$.

Confusion. The round function first applies the non-linear part, called SubBytes, which applies in parallel the same 8-bit Sbox.

$$\boldsymbol{y} = \texttt{SubBytes}(\boldsymbol{x}) \iff \forall i \in [0, 15],\ y_i = \texttt{Sbox}(x_i).$$

The AES Sbox is the composition $A \circ \text{Inv}$ of the extended inverse of $\mathbb{F}_{2^8}$ — $\text{Inv}(0) = 0$ and $\text{Inv}(x) = x^{-1}$ for all $x \neq 0$ — and an affine transformation A such that for all $i \in [0, 7]$,

$$A_i(\boldsymbol{x}) = x_i + x_{(i+4) \bmod 8} + x_{(i+5) \bmod 8} + x_{(i+6) \bmod 8} + x_{(i+7) \bmod 8} + c_i,$$

where $\boldsymbol{c} = (1, 1, 0, 0, 0, 1, 1, 0)$.

Algorithm 1 Update function for the key expansion of the AES-128.

function $F^{128}(i, \boldsymbol{x})$
  input
    i is the iteration number of the update function.
    $\boldsymbol{x}$ is the round-key $\boldsymbol{k}^{(i-1)}$.
  output
    $\boldsymbol{y}$ is the round-key $\boldsymbol{k}^{(i)}$.

  $z_\ell \leftarrow \texttt{Sbox}\,(x_{\ell+1 \bmod 4,\ 3}) \in \mathbb{F}_{2^8}$ for all $\ell \in [0,3]$.
  $z_0 \leftarrow z_0 + 2^{i-1}$
  $y_{\ell,0} \leftarrow x_{\ell,0} + z_\ell$ for all $\ell \in [0,3]$.
  for j from 1 to 3 **do**
    $y_{\ell,j} \leftarrow x_{\ell,j} + y_{\ell,j-1}$ for all $\ell \in [0,3]$.
  return $\boldsymbol{y}$

Diffusion. Then the linear part applies the `ShiftRows` operation, which shifts the byte at position (i,j) in the state matrix to the position $(i, j-i \bmod 4)$.

$$\begin{aligned}\boldsymbol{y} = \texttt{ShiftRows}\,(\boldsymbol{x}) &\iff \forall i,j \in [0,3],\ y_{i,(j-i) \bmod 4} = x_{i,j} \\ &\iff \forall i,j \in [0,3],\ y_{i,j} = x_{i,(j+i) \bmod 4}.\end{aligned}$$

Finally, the linear part also applies the `MixColumns` operation (except for the last round) which is a left multiplication of the state matrix by a constant matrix $\boldsymbol{M}$.

$$\boldsymbol{M} \stackrel{\text{def}}{=} \begin{pmatrix} 2 & 3 & 1 & 1 \\ 1 & 2 & 3 & 1 \\ 1 & 1 & 2 & 3 \\ 3 & 1 & 1 & 2 \end{pmatrix}$$

By definition, the matrix multiplication applies in parallel on each column.

$$\boldsymbol{y} = \texttt{MixColumns}\,(\boldsymbol{x}) \iff \boldsymbol{y} = \boldsymbol{M} \cdot \boldsymbol{x} \iff \forall j \in [0,3],\ \boldsymbol{y}_{*,j} = \boldsymbol{M} \cdot \boldsymbol{x}_{*,j}.$$

The full round function R is then the composition

$$R \stackrel{\text{def}}{=} \texttt{MixColumns} \circ \texttt{ShiftRows} \circ \texttt{SubBytes}.$$

Key expansion. The key expansion of the AES follows the general architecture of Figure 1.12 and we give as an example the update function of AES-128 (i.e. for a 128-bit key) in Algorithm 1. For the 196- and 256-bit versions, the update functions are very similar and given in [Nisa]. The addition of a round-key $\boldsymbol{k}^{(i)}$ is denoted by the operation $\texttt{AddRoundKey}_i$.

Finally, AES-128 is for example the composition

$$\begin{aligned}\texttt{AddRoundKey}_{10} \circ \texttt{ShiftRows} \circ \texttt{SubBytes} \circ \texttt{AddRoundKey}_9 \circ R \circ \texttt{AddRoundKey}_8 \circ \cdots \\ \circ \texttt{AddRoundKey}_1 \circ R \circ \texttt{AddRoundKey}_0.\end{aligned}$$

Note that the `MixColumns` operation is omitted in the last round.

1.4.4 Modes of operation

The construction which turns a block cipher into a full encryption scheme is called a *mode of operation*. It is usually proven secure under the assumption that the block cipher is a pseudo-random function.

Pseudo-random functions. Similarly to the definition of a PRP, a pseudo-random function (PRF) is a keyed family of functions $F : \mathcal{K} \times \mathcal{X} \to \mathcal{Y}$ which satisfies the PRF security notion where an adversary has to distinguish $\boldsymbol{x} \mapsto F(\boldsymbol{k}, \boldsymbol{x})$ for a random $\boldsymbol{k}$ from a random function $R : \mathcal{X} \to \mathcal{Y}$. The PRP/PRF Switching Lemma asserts that a PRP is also a PRF up to the *birthday bound*.

Lemma 1.1 (PRP/PRF Switching [BR04])**.** *Let B be a block cipher with block size n and $\mathcal{A}$ be an adversary against* PRP *or* PRF *that makes q queries. Then*

$$\left|\mathrm{Adv}_B^{\mathsf{PRP}}(\mathcal{A}) - \mathrm{Adv}_B^{\mathsf{PRF}}(\mathcal{A})\right| \leqslant \frac{q(q-1)}{2^{n+1}}.$$

The main idea behind the proof of this lemma is that when q elements $y_0, \ldots, y_{q-1}$ of $\mathbb{F}_2^n$ are drawn independently and uniformly at random with replacement, which is the case when querying a random *function*,

$$\mathbb{P}(\exists i \neq j :\ y_i = y_j) \leqslant \sum_{i<j} \mathbb{P}(y_i = y_j) \leqslant \frac{q(q-1)}{2} \cdot \frac{1}{2^n}, \tag{1.2}$$

whereas when $y_0, \ldots, y_{q-1}$ are drawn without replacement, which is the case when querying a random *permutation*, $\mathbb{P}(\exists i \neq j :\ y_i = y_j) = 0$. A formal proof of Lemma 1.1 is given by Bellare and Rogaway in [BR04].

The birthday bound. We see in Lemma 1.1 that when q is near $2^{n/2}$, a PRP-secure block cipher cannot be considered to be PRF-secure. The value $2^{n/2}$ is called the birthday bound, named after the famous birthday paradox.

Proposition 1.2 (Birthday paradox)**.** *In a set of N elements, we draw independently and uniformly at random q elements $x_0, \ldots, x_{q-1}$ with replacement. Then*

$$\mathbb{P}(\exists i \neq j :\ x_i = x_j) \geqslant 1 - \exp\left(-\frac{q(q-1)}{2N}\right).$$

Consequently, for all $p \in [0, 1[$, if $q - 1 \geqslant \sqrt{2N \cdot (-\log(1-p))}$, the probability that we draw at least one element twice is bigger than p.

For example, if we draw $q = 1 + 1.5 \cdot \sqrt{N}$ elements, we draw at least one element twice with probability higher than $\frac{2}{3}$. In particular, with $N = 2^n$ and approximately $2^{n/2}$ queries, the birthday paradox asserts that a block cipher cannot be PRF-secure.

Section 1.4.4.1 and Section 1.4.4.2 give the famous examples of `CBC` and `CTR`, both described in [Dwo01]. There are other modes of operation and we redirect to Chapter 2 in the thesis of Ferdinand Sibleyras [Sib20] for a quick overview over the main modes of operation and to the report of Phillip Rogaway [Rog11] for an in-depth and more exhaustive study.

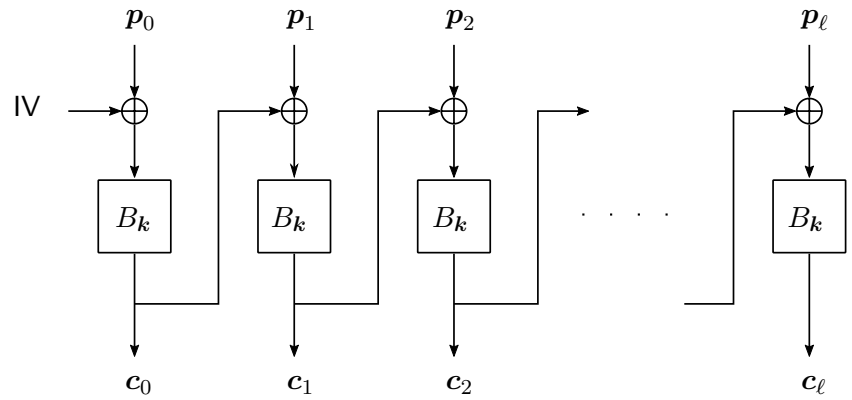

Figure 1.16 – The CBC mode of operation.

1.4.4.1 The CBC mode

The Cipher Block Chaining (CBC) mode is an important historical standard but, according to Rogaway in [Rog11], there are no situations where using CBC rather than CTR makes any sense. However, CBC still gives a typical and instructive example of a secure mode of operation.

Description. Figure 1.16 gives a schematic view of CBC. The key $\boldsymbol{k}$ of the overall encryption scheme is used as a key for the block cipher B. The IV is chosen uniformly at random and independently for each encryption. Since a message $\boldsymbol{p}$ can have a length which is not a multiple of n, an invertible padding like $\boldsymbol{p} \mapsto (\boldsymbol{p} \mid 1 \mid \mathbf{0}_{(-\text{length}(\boldsymbol{p})-1) \bmod n})$ is required. $\boldsymbol{c}_0 = B(\boldsymbol{k}, \mathsf{IV} \oplus \boldsymbol{p}_0)$ and for all $i \geqslant 1$, $\boldsymbol{c}_i = B(\boldsymbol{k}, \boldsymbol{c}_{i-1} \oplus \boldsymbol{p}_i)$. The decryption operation will need an implementation of the inverse B^{-1}: $\boldsymbol{p}_i = \boldsymbol{c}_{i-1} \oplus B^{-1}(\boldsymbol{k}, \boldsymbol{c}_i)$. Moreover, encryption has to perform block cipher evaluations sequentially.

Security proof. Let n be the block size and q be the total number of calls to the block cipher B that the challenger would need to answer queries in the Real mode. More precisely, if the adversary makes Q queries with respective bit lengths $N_1, \ldots, N_Q$, $q \stackrel{\text{def}}{=} \frac{1}{n}\sum_{i=1}^{Q} N_i$. One can show that

$$\text{Adv}^{\mathsf{IND\$\text{-}CPA}}_{\mathtt{CBC}-B}(q \text{ queries to } B) \leqslant 2 \cdot \text{Adv}^{\mathsf{PRP}}_{B}(q) + q^2 \cdot 2^{1-n}. \tag{1.3}$$

Equation (1.3) means that if the block cipher B is PRP-secure, then CBC instantiated with B is IND$-CPA-secure against adversaries who cannot make a number of queries beyond the birthday bound.

Here is the idea of the proof. Informally, if $B_{\boldsymbol{k}}$ behaves like a random function and if the plaintext $\boldsymbol{p} = (\boldsymbol{p}_0 \mid \ldots \mid \boldsymbol{p}_\ell)$ is fixed, then for any i such that $\boldsymbol{c}_{i-1} \oplus \boldsymbol{p}_i$ is a new input to $B_{\boldsymbol{k}}$, the output $\boldsymbol{c}_i = B_{\boldsymbol{k}}(\boldsymbol{c}_{i-1} \oplus \boldsymbol{p}_i)$ and hence $\boldsymbol{c}_i \oplus \boldsymbol{p}_{i+1}$ are uniformly distributed. Therefore, as in Equation (1.2), the probability that there exists i such that $\boldsymbol{c}_i \oplus \boldsymbol{p}_{i+1}$ is not a new input is bounded by $q^2 2^{-n-1}$. Finally, if $B_{\boldsymbol{k}}$ behaves like a random function — which is measured by $\text{Adv}^{\mathsf{PRF}}_{B}(q)$ — and if the inputs to $B_{\boldsymbol{k}}$ are all different — which is the case with probability higher than $1 - q^2 2^{-n-1}$ — then the outputs of CBC cannot be distinguished from random strings. A formal proof would give

$$\text{Adv}^{\mathsf{IND\$\text{-}CPA}}_{\mathtt{CBC}-B}(q \text{ queries to } B) \leqslant 2 \cdot \text{Adv}^{\mathsf{PRF}}_{B}(q) + q^2 \cdot 2^{-n}$$

We can then derive Equation (1.3) from Lemma 1.1. Equation (1.3) was first shown in [Bel+97] for IND-CPA then reaffirmed by Rogaway for IND$-CPA in [Rog11].

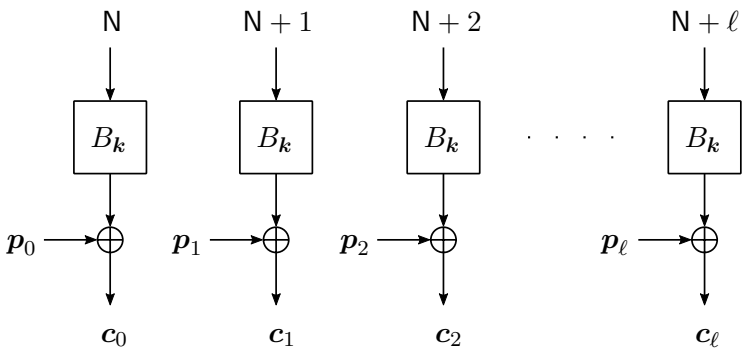

Figure 1.17 – The CTR mode of operation.

Matching distinguisher. In fact, there exists a distinguisher when $q = \Omega\left(2^{n/2}\right)$. As explained in Chapter 6 of [Jou09], we can expect thanks to the birthday paradox that we will get two equal ciphertext blocks $\boldsymbol{c}_i = \boldsymbol{c}_j$ with $i \neq j$ — this is called a *collision* — and since the block cipher is a family of permutations, we consequently have that $\boldsymbol{c}_{i-1} \oplus \boldsymbol{c}_{j-1} = \boldsymbol{p}_i \oplus \boldsymbol{p}_j$. In a IND$-CPA game, the adversary can request the encryption of a plaintext $\boldsymbol{p}$ with length $n2^{n/2}$. The challenger answers with $\boldsymbol{c}$ and upon detection of a collision in the blocks of $\boldsymbol{c}$, the attacker can check whether $\boldsymbol{p}_i \oplus \boldsymbol{p}_j = \boldsymbol{c}_{i-1} \oplus \boldsymbol{c}_{j-1}$. Indeed, this is not expected to occur if $\boldsymbol{c}$ is a random string, independent from $\boldsymbol{p}$ but it is expected for CBC as we just saw. This distinguisher is then called a *matching* distinguisher as it matches the security bound and shows that it is tight.

Interestingly, this distinguisher can be used to recover the plaintext information $\boldsymbol{p}_j$ if $\boldsymbol{p}_i$ is known. This simple fact was used by Bhargavan and Leurent in [BL16] to mount practical attacks in real-world scenarios when the CBC mode is used with a 64-bit block cipher like 3DES, a combination still supported for backwards compatibility by TLS 1.2 [RD08]. Indeed, with such a combination, the birthday bound involves only 32 GigaBytes of data (2^{32} blocks).

1.4.4.2 The CTR mode

CBC is however not secure in the Nonce variant and random IVs are harder to implement securely than just non repeated chosen Nonce. In TLS 1.3, the only available mode of operation is the Counter (CTR) mode inside its authenticated variants CCM and GCM (see Section 1.5.3).

Description. CTR builds a keystream generator with a Nonce and a block cipher as illustrated in (Figure 1.17). There can be different ways of initializing the counter with the Nonce N or incrementing the counter. The most important thing is that the produced keystreams and hence the sequences of the incremented counters should not overlap for two different Nonces. Typically, the Nonce will be $\frac{n}{2}$-bit long and take the first half of the block while the counter will start from 0 in the second half of the block: for all i, $\boldsymbol{c}_i = \boldsymbol{p}_i \oplus B\left(\boldsymbol{k}, (\mathsf{N} \mid i)\right)$ where i is encoded in base 2. CTR has many advantages: decryption is the same as encryption, both operations can be performed with parallel and unordered block cipher evaluations, no padding is required and the inverse of the block cipher does not need to be implemented.

Security proof. An elementary security proof (first given in [Bel+97]) gives that

$$\mathrm{Adv}^{\mathsf{IND\$\text{-}CPA\text{-}N}}_{\mathtt{CTR}-B}\left(q \text{ queries to } B\right) \leqslant 2 \cdot \mathrm{Adv}^{\mathsf{PRF}}_{B}\left(q\right).$$

With Lemma 1.1, we finally have that

$$\mathrm{Adv}^{\mathsf{IND\$\text{-}CPA\text{-}N}}_{\mathtt{CTR}-B}(q \text{ queries to } B) \leqslant 2 \cdot \mathrm{Adv}^{\mathsf{PRP}}_{B}(q) + q^2 \cdot 2^{-n}.$$

`CTR` is then secure in the Nonce variant up to the birthday bound.

Matching distinguisher. Again, when $q = \Omega\left(2^{n/2}\right)$, the birthday paradox applied on the sequence $\boldsymbol{x}_i = \boldsymbol{p}_i \oplus \boldsymbol{c}_i$ gives a chosen plaintext distinguisher on `CTR` when used with a block cipher rather than a PRF. If this particular distinguisher is less devastating than the one against `CBC`, Leurent and Sibleyras exhibited in [LS18] an attack to recover an unknown plaintext block with complexity $\widetilde{O}\left(2^{n/2}\right)$.

Block ciphers with `CTR` vs. dedicated generators. The general opinion (exposed in [Bog+07] for example) is that `CTR` instantiated with a secure block cipher like the AES offers a more convincing security argument than the security analysis of a dedicated keystream generator against IND$-CIV. There are basically two reasons for this opinion.

- The block cipher is a more basic primitive than the keystream generator because the block size n is fixed and the set of permutations of $\mathbb{F}_2^n$ is relatively small.
- Cryptographers have traditionnally put more effort in the study of block ciphers and the trust we have in a security conjecture grows with the amount of cryptanalysis effort. In particular, classical attacks on block ciphers are well understood and benefit from the continuous developement of both theoretical and computational tools.

However, dedicated keystream generators have non-negligeable benefits: they can be simpler to implement, faster to run and more energy efficient. Those characteristics are game changers in constrained environments where speed is important or where energy resources are limited. Statistics on the performance of AES-`CTR` and stream ciphers with dedicated generators like `ChaCha20` can be found in [BL].

1.5 Symmetric-key authentication

In this section, we briefly present how integrity is adressed in the symmetric-key setting with hash functions and message authentication codes (MACs). Hash functions are designed and analyzed similarly to block ciphers or dedicated keystream generators and MACs are necessary to any modern symmetric-key authenticated encryption scheme.

1.5.1 Hash functions

A cryptographic hash function

$$H : \left\{ \begin{array}{ccc} \mathcal{M} & \longrightarrow & \mathbb{F}_2^n \\ \boldsymbol{m} & \longmapsto & \boldsymbol{h}, \end{array} \right. \qquad \mathcal{M} = \bigcup_{i=1}^{N} \mathbb{F}_2^i,\ N > n,$$

is a basic primitive which deterministically outputs a rather small footprint $\boldsymbol{h}$ of a possibly large message $\boldsymbol{m}$. This footprint is called the *hash digest* or *value*. We typically have that $n \in \{128, 256, 512\}$ and $N \in \{2^{64}, 2^{128}\}$. Some hash functions, called universal hash functions (UHF), need a secret key: $\boldsymbol{h} = H(\boldsymbol{k}, \boldsymbol{m})$ with $\boldsymbol{k} \in \mathbb{F}_2^{128}$ for example.

Collision resistance. The main notion which defines the security of a hash function is *collision resistance*.

More precisely, for a UHF, for each message pair $(\boldsymbol{x}, \boldsymbol{y})$, the probability that $H(\boldsymbol{k}, \boldsymbol{x}) = H(\boldsymbol{k}, \boldsymbol{y})$ when $\boldsymbol{k}$ is chosen uniformly at random in $\mathbb{F}_2^{128}$ should be 2^{-128}. This behaviour is captured by this simple game: the game master draws a random key $\boldsymbol{k}$ and keeps it secret. Without further knowledge than the public specification of the UHF, the adversary has to output two *different* messages $\boldsymbol{x}$ and $\boldsymbol{y}$ such that $H(\boldsymbol{k}, \boldsymbol{x}) = H(\boldsymbol{k}, \boldsymbol{y})$.

For a plain unkeyed hash function, the adversary just has to exhibit $\boldsymbol{x}$ and $\boldsymbol{y}$ such that $H(\boldsymbol{x}) = H(\boldsymbol{y})$. In this case, it is obvious that since $N > n$, there always exists such a pair $(\boldsymbol{x}, \boldsymbol{y})$ for any hash function H and an adversary who knows such a pair breaks the collision resistance of H in constant time. Consequently, secure hash functions do not really exist but cryptographers consider a hash function for which no collision has been found so far to be secure. The best generic strategies to compute collisions are based on the birthday paradox and cycle-finding algorithms (see for example Chapters 6 and 7 in [Jou09]). They have a complexity of $2^{\frac{n}{2}}$ queries $\boldsymbol{x} \mapsto H(\boldsymbol{x})$.

Preimage resistance. Collision resistance implies the security notions of preimage resistance. First-preimage resistance captures the fact that the hash function is one-way. For a given $\boldsymbol{h} \in \mathbb{F}_2^n$, it should be hard to find $\boldsymbol{x} \in \mathcal{M}$ such that $H(\boldsymbol{x}) = \boldsymbol{h}$. The corresponding security game has straightforward rules: the master of the game draws $\boldsymbol{h} \in \mathbb{F}_2^n$ uniformly at random and publishes it. The best generic strategy needs 2^n queries $\boldsymbol{x} \mapsto H(\boldsymbol{x})$. Second-preimage resistance is a bit stronger. This time the attacker is given $\boldsymbol{x} \in \mathcal{M}$ and has to find $\boldsymbol{y}$ such that $H(\boldsymbol{x}) = H(\boldsymbol{y})$. Again, the generic bound is 2^n queries.

Random oracles. We just saw that collision resistance is hard to formally define for unkeyed hash functions. In fact, security proofs for systems built with such primitives often need the ideal model of an unkeyed hash function, the random oracle. The random oracle is mathematically defined as a function chosen uniformly at random among the functions from $\mathcal{M}$ to $\mathbb{F}_2^n$. Equivalently, each *new* query $\boldsymbol{x} \mapsto H(\boldsymbol{x})$ is answered by the random oracle with a random element from $\mathbb{F}_2^n$.

In practice. UHF are essentially used as building blocks for MACs and they are much less common than unkeyed hash functions. Unkeyed hash functions are used to build many cryptosystems and for other security-oriented tasks, like storing passwords or doing basic data-integrity checks. However, different hash functions are needed for the different use-cases. For example, storing passwords is better with a hash function for which computing a digest is long — say 100 ms rather than 1 ms: the speed difference is not noticeable for the legitimate user who wants to log in but it makes brute-force or dictionary attacks more difficult. Here are some common hash functions.

- The *Message Digest* family with the very fast `MD5` (1992) [Riv92] which outputs 128-bit digests. Its collision resistance is broken since 2004 [Wan+04]. Therefore, `MD5` cannot be used in cryptographic schemes but it is still in use for computing checksums.
- The *Secure Hash Algorithm* family:
  - `SHA-1` [Nisd] which outputs 160-bit digests was deprecated by the NIST in 2011 [Nisf] because of the theoretical attacks by Wang, Yin and Yu against its collision resistance [WYY05]. Stevens, Bursztein, Karpman, Albertini and Markov managed to implement

this attack to exhibit a collision in 2017 [Ste+17]. The recent implementation of a more powerful attack by Leurent and Peyrin [LP20] now threatens real-world protocols depending on `SHA-1`.

- The `SHA-2` family [Nisd] proposes different sizes for digests from 224 to 512. The most used variants are probably `SHA-256` and `SHA-512`.
- Finally, the `SHA-3` family [Nise] offers an alternative to `SHA-2` with completely different design strategies but the same sizes for digests. Indeed, the design of `SHA-2` is similar to the ones of `SHA-1` and `MD5` and even if `SHA-2` is not expected to be broken any time soon, a sense of caution encouraged the NIST to launch a competition in 2007 to provide a supplement for `SHA-2` [Cha+12]. The `Keccak` proposal by Bertoni, Daemen, Peeters and Van Assche was chosen among 5 finalists and 64 candidates.

- The original `BLAKE` family was a finalist candidate for the `SHA-3` competition designed by Aumasson, Henzen, Meier and Phan [Cha+12]. In 2012, Aumasson, Neves, Wilcox-O'Hearn and Winnerlein proposed the `BLAKE2` family to achieve the best software performance possible [SA15]. Thanks to this good performance, `BLAKE2` is notably used in the Noise protocol framework by Perrin [Per18], used in turn in modern cryptographic protocols like WireGuard [Don20].

1.5.2 Message authentication codes

Message authentication codes (MAC) are cryptosystems which ensure data integrity in the symmetric-key setting. They allow a legitimate user, i.e. an owner of a secret key, to be confident that some data cannot be altered by an attacker who does not known the secret key. As for encryption, this data can be a message $\boldsymbol{m}$ sent over a network or a file on a hard drive, but it does not need to be kept private. For example, it can be interesting to authenticate the header of an IP packet to avoid spoofing or to authenticate an executable file before running it.

Definition. Let $\mathcal{M} \stackrel{\text{def}}{=} \bigcup_{i=1}^{N} \mathbb{F}_2^i$ be the message space, $\mathcal{K} = \mathbb{F}_2^m$ be the key space and $\mathcal{T} = \mathbb{F}_2^\tau$. A MAC is a pair of algorithms (S, V) such that

$$S : \left\{ \begin{array}{rcl} \mathcal{K} \times \mathcal{M} & \longrightarrow & \mathcal{T} \\ (\boldsymbol{k}, \boldsymbol{m}) & \longmapsto & \boldsymbol{t} \end{array} \right. \quad \text{and} \quad V : \left\{ \begin{array}{rcl} \mathcal{K} \times \mathcal{M} \times \mathcal{T} & \longrightarrow & \{\mathsf{accept}, \mathsf{reject}\} \\ (\boldsymbol{k}, \boldsymbol{m}, \boldsymbol{t}) & \longmapsto & \mathsf{accept} \text{ or } \mathsf{reject}. \end{array} \right.$$

S can be probabilistic and is called the *signing* algorithm. Its output $\boldsymbol{t}$ is called a *tag*. V is deterministic and is called the *verification* algorithm. A MAC (S, V) must verify the correctness property

$$\forall \boldsymbol{k}, \boldsymbol{m},\ V(\boldsymbol{k}, \boldsymbol{m}, S(\boldsymbol{k}, \boldsymbol{m})) = \mathsf{accept}.$$

Security notion. The main security notion for MACs is called EUF-CMA for Existential Un-Forgeability against Chosen Message Attacks. In the security game, the challenger first draws a secret key $\boldsymbol{k}$ uniformly at random. The adversary can then query the signature of messages of her choice $\boldsymbol{m}_i \mapsto \boldsymbol{t}_i = S(\boldsymbol{k}, \boldsymbol{m}_i)$, which explains "chosen-message attack". Her goal is to *forge* a valid pair $(\boldsymbol{m}, \boldsymbol{t})$ such that $(\boldsymbol{m}, \boldsymbol{t}) \neq (\boldsymbol{m}_i, \boldsymbol{t}_i)$ for all i. Note that the adversary does not know whether she

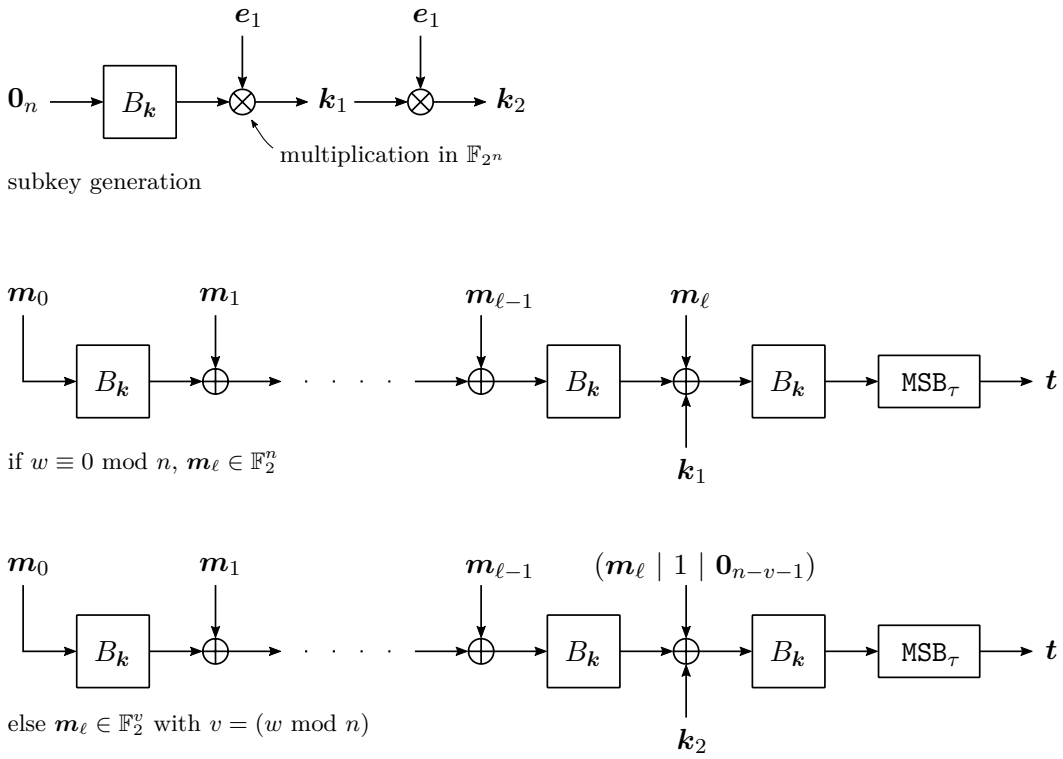

Figure 1.18 – The CMAC mode of operation. $\boldsymbol{m} \in \mathbb{F}_2^w$ for some length w and n is the block size. In the subkey generation, the multiplication by $\boldsymbol{e}_1$ in $\mathbb{F}_{2^n}$ is defined by a constant $\boldsymbol{p}$ and $\forall \boldsymbol{x} \in \mathbb{F}_2^n$, $\boldsymbol{e}_1 \cdot \boldsymbol{x} = (0, x_0, \ldots, x_{n-2}) + x_{n-1} \cdot \boldsymbol{p}$. It is in fact the multiplication by X in $\mathbb{F}_2[X]/P(X)$ where $P = X^n + \sum_{i<n} p_i X^i$ is an irreducible polynomial over $\mathbb{F}_2$ to be specified. $\forall \boldsymbol{x} \in \mathbb{F}_2^n$, $\mathtt{MSB}_\tau(\boldsymbol{x}) = (x_{n-\tau}, \ldots, x_{n-1})$.

succeeded the game. The performance of an adversary $\mathcal{A}$ against EUF-CMA for a MAC $M = (S, V)$ is given by its success probability

$$\mathrm{Succ}_M^{\mathsf{EUF\text{-}CMA}}(\mathcal{A}) = \mathbb{P}\left(\mathcal{A} \text{ outputs a new pair } (\boldsymbol{m}, \boldsymbol{t}) \text{ s.t. } V(\boldsymbol{k}, \boldsymbol{m}, \boldsymbol{t}) = \mathsf{accept}\right).$$

The probability space is given by the uniform distribution of $\boldsymbol{k}$ and the random coins used by S and $\mathcal{A}$.

In practice. As for encryption, there are different strategies for building MACs and the report of Rogaway [Rog11] provides an in-depth review of many of those strategies. We focus here on the example of CMAC, the *Cipher-based* MAC standardized by the NIST in [Dwo05] and described in Figure 1.18. It is a mode of operation for block ciphers and it is proved EUF-CMA-secure if the underlying block cipher is PRF-secure. The signing algorithm of CMAC is deterministic, therefore its verification algorithm is as simple as running the signature and comparing the result with the input tag. A security proof of CBC-MAC is given in [BR04] for example.

CMAC is based on the CBC-MAC family of PRFs illustrated in Figure 1.19. CBC-MAC is a secure PRF for same length messages but it cannot be a secure MAC: if $\boldsymbol{m}_1 \in \mathbb{F}_2^n$ and $\boldsymbol{t} = \mathtt{CBC\text{-}MAC}(\boldsymbol{k}, \boldsymbol{m}_1) = B_{\boldsymbol{k}}(\boldsymbol{m}_1)$ then $\boldsymbol{t} = \mathtt{CBC\text{-}MAC}(\boldsymbol{k}, (\boldsymbol{m}_1, \boldsymbol{m}_2))$ where $\boldsymbol{m}_2 = \boldsymbol{m}_1 \oplus \boldsymbol{t}$. The subkeys of CMAC and their use

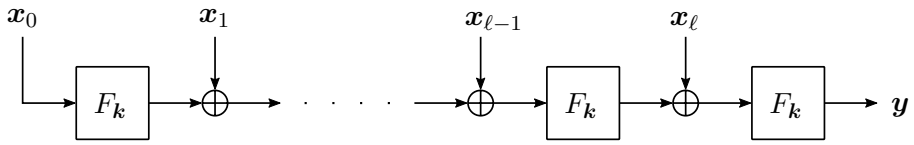

Figure 1.19 – $\texttt{CBC-MAC}_\ell$ expands a PRF $F_{\boldsymbol{k}} : \mathbb{F}_2^n \to \mathbb{F}_2^n$ into a PRF from $\mathbb{F}_2^{n\cdot(\ell+1)}$ to $\mathbb{F}_2^n$.

with the padding are essentially what allows CMAC to be a secure MAC for messages of any length. CMAC was first proposed and proved secure by Iwata and Kurosawa in [IK03] as OMAC1. Finally, the truncation of the tag to τ bits through the function $\texttt{MSB}_\tau$ allows to choose a size/security trade-off.

1.5.3 Authenticated encryption

In Section 1.3.2.2, we presented the IND-CPA game where the adversary can query the encryption of a chosen plaintext, to ensure that a ciphertext could not leak any bit of information about the underlying plaintext. However, we can imagine scenarios where an attacker knows how to modify a ciphertext, even if it looks random, to change the underlying plaintext, even if it remains unknown. This is easy to do with a stream cipher for example: any bit flipped in the ciphertext $\boldsymbol{c} = S(\boldsymbol{k}, \mathsf{IV}) \oplus \boldsymbol{p}$ will result in the same bit being flipped in the plaintext and the recipient will not detect such a malicious tampering. Consequently, the SKE schemes presented in Section 1.3 are not secure in such scenarios and IND-CPA-secure encryption schemes do not provide enough resistance against that kind of adversary. In this section, we define AE schemes, a family of SKE schemes which address this issue and the security notions they should satisfy.

AE schemes. An authenticated encryption (AE) scheme is a SKE scheme such that its decryption algorithm can output reject for some input ciphertexts. This behaviour models the fact that an AE scheme can detect whether a ciphertext has been tampered with. Of course, an AE scheme should be IND-CPA-secure but another security notion is needed if we want to consider scenarios where the adversary can benefit from modified ciphertexts. The different security notions for AE schemes were exhaustively investigated by Bellare and Namprempre in [BN00].

Ciphertext unforgeability. One of these security notions, called *ciphertext integrity* in [BN00], is EUF-CPA. In its security game, the challenger first draws a random key $\boldsymbol{k}$, then the adversary can query the encryption of chosen plaintexts and she must finally output a string $\boldsymbol{m}$ different from the received ciphertexts such that $D(\boldsymbol{k}, \boldsymbol{m}) \neq \mathsf{reject}$. In other words, the adversary must forge a *new valid ciphertext*. As chosen in [Rog+01] for example, an AE scheme can be defined secure when it is satisfies both IND-CPA and EUF-CPA.

Generic constructions. Given the similarity between EUF-CMA for MACs and EUF-CPA for AE schemes, it is natural to use an IND-CPA-secure cipher together with a secure MAC through a generic construction. We only present the encrypt-then-mac (EtM) construction as it is generically secure but a survey of the different possibilities is given in [BN00]. The EtM construction builds a secure AE scheme $(E_{\mathrm{EtM}}, D_{\mathrm{EtM}})$ from an IND-CPA-secure cipher (E, D) and an EUF-CMA-secure

MAC (S, V).

$$E_{\mathrm{EtM}} : \begin{cases} (\mathcal{K}_E \times \mathcal{K}_S) \times \mathcal{M} & \longrightarrow & \mathcal{M} \times \mathcal{T} \\ (\boldsymbol{k}_1, \boldsymbol{k}_2), \boldsymbol{p} & \longmapsto & (\boldsymbol{c}, \boldsymbol{t}) \text{ where } \boldsymbol{c} = E(\boldsymbol{k}_1, \boldsymbol{p}) \text{ and } \boldsymbol{t} = S(\boldsymbol{k}_2, \boldsymbol{c}), \end{cases}$$

$$D_{\mathrm{EtM}} : \begin{cases} (\mathcal{K}_E \times \mathcal{K}_S) \times \mathcal{M} \times \mathcal{T} & \longrightarrow & \mathcal{M} \cup \{\mathsf{reject}\} \\ (\boldsymbol{k}_1, \boldsymbol{k}_2), (\boldsymbol{c}, \boldsymbol{t}) & \longmapsto & \begin{cases} D(\boldsymbol{k}_1, \boldsymbol{c}) & \text{if } V(\boldsymbol{k}_2, \boldsymbol{c}) = \mathsf{accept}, \\ \mathsf{reject} & \text{otherwise.} \end{cases} \end{cases}$$

The security proof of EtM, quite elementary and given in [BN00], needs the MAC key $\boldsymbol{k}_2$ to be independent from the encryption key $\boldsymbol{k}_1$. It gives that

$$\mathrm{Adv}^{\mathsf{IND\text{-}CPA}}_{\mathrm{EtM}}(t) \leqslant \mathrm{Adv}^{\mathsf{IND\text{-}CPA}}_{(E,D)}(t) \quad \text{and} \quad \mathrm{Succ}^{\mathsf{EUF\text{-}CPA}}_{\mathrm{EtM}}(t) \leqslant \mathrm{Succ}^{\mathsf{EUF\text{-}CMA}}_{(S,V)}(t).$$

Having different keys for E and S is a pity when both cryptosystems are built with the same block cipher B since two key expansions have to be computed. Designers thus decided to blend ciphers and MACs together in specific and more compact constructions like `CCM` and `GCM` to use only one block-cipher key.

Nonce and associated data. AE schemes can be built in the Nonce variant. Indeed, as explained for SKE schemes in Section 1.3.1, this variant is often easier to implement securely and may provide better security. In the Nonce variant, the encryption algorithm is deterministic and it has a third input after the key and the plaintext, the Nonce, which is also an input of the decryption algorithm. With EtM for example, this implies that in the Nonce variant, the construction should authenticate the Nonce together with the ciphertext but it should not encrypt the Nonce. The Nonce is then a piece of public data which is neither plaintext, neither ciphertext but the decryption will answer reject if the Nonce has been altered. This property can be very interesting to securely link any kind of public data to a ciphertext, like IP headers or any necessarily public metadata. An AE scheme which supports the addition of authenticated public data is said to perform authenticated encryption with associated data (AEAD). Since this functionality adds very little overhead compared to plain AE schemes, AE schemes are often considered to support associated data by default. Definitions, generic constructions and properties of AEAD schemes were first given by Rogaway in [Rog02].

In practice. Standardized by the NIST, `CCM` [Dwo04] and `GCM` [Dwo07] are block-cipher modes of operation which perform AEAD. They both encapsulate the Nonce-based `CTR` mode but they have different MAC constructions. In particular, `CCM`'s MAC is built with the `CBC-MAC` PRF. Rogaway gives a detailed review for both modes in his report [Rog11]. They are notably used in TLS 1.3 [Res18] with the AES as underlying block cipher.

1.6 Public-key encryption

In the public-key setting, the assumption that users already share a secret key does not hold anymore. Instead, the recipient keeps the decryption key *private* (or secret) and publishes the *public* encryption key. This setting is also called *asymmetric* since the keys for encryption and decryption are different. It is not hard to imagine how interesting public-key encryption (PKE) schemes can be: for example, once we know someone's public key, we can send him or her encrypted information without previous

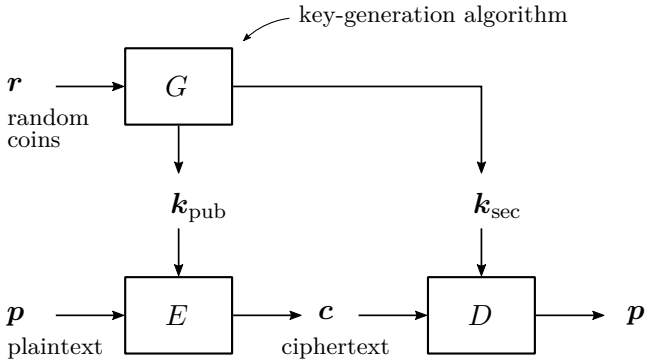

Figure 1.20 – In the public-key setting, a key-generation algorithm has to build a pair of encryption/decryption keys.

nor further interaction[2]. Moreover, each user needs only one key pair to share information with other users. It might be harder to imagine that public-key encryption schemes can satisfy the requirements of a secure encryption and still be practical because of the following observations.

1. Decryption still has to deterministically undo encryption, which implies that the public and the secret key are related somehow. In fact, besides encryption and decryption algorithms, a PKE scheme is composed of a *key-generation* algorithm to forge public and secret keys. This is illustrated in Figure 1.20.

2. The public key (and the public specification of the cryptosystem) should not leak any information that could threaten one-wayness or indistinguishability security notions. This means in particular that the link between the two keys should be *hard* to compute or *hidden*.

In order to answer this second point, PKE schemes are built around a computational assumption which can take two forms: the *one-wayness* of a family of *trapdoor* functions or the security of a key-exchange protocol. We only present the former.

1.6.1 One-way trapdoor functions

These special objects can be understood as the counterparts of block ciphers in the asymmetric world. Indeed, as block ciphers perform the most basic form of symmetric key encryption, one-way trapdoor functions (OWTF) perform the most basic form of public-key encryption.

Let $\mathcal{X}$ be the plaintext space, $\mathcal{Y}$ be the ciphertext space, $\mathcal{K}_{\text{pub}}$ be the public-key space and $\mathcal{K}_{\text{sec}}$ be the secret-key space. A OWTF is a trio of *efficient* algorithms (G, F, F^{-1}).

$$G : \begin{cases} \mathbb{F}_2^t & \longrightarrow & \mathcal{K}_{\text{pub}} \times \mathcal{K}_{\text{sec}} \\ \boldsymbol{r} & \longmapsto & (\boldsymbol{k}_{\text{pub}}, \boldsymbol{k}_{\text{sec}}) \end{cases}$$

[2] The link between a public key and an identity has to be trusted as well. Establishing this link is in fact a keystone in many protocols like TLS. It is often taken care of through a Public Key Infrastructure (PKI) which heavily relies on digital signatures.

is the key generator where $\boldsymbol{r}$ models *random coins* — key generation has to be probabilistic — and $\boldsymbol{k}_{\text{pub}}$ and $\boldsymbol{k}_{\text{sec}}$ are encryption/decryption public and secret keys.

$$F : \left\{ \begin{array}{ccc} \mathcal{K} \times \mathcal{X} \times \mathbb{F}_2^{t'} & \longrightarrow & \mathcal{Y} \\ (\boldsymbol{k}_{\text{pub}}, \boldsymbol{p}, \boldsymbol{r}') & \longmapsto & \boldsymbol{c} \end{array} \right. \quad \text{and} \quad F^{-1} : \left\{ \begin{array}{ccc} \mathcal{K}_{\text{sec}} \times \mathcal{Y} & \longrightarrow & \mathcal{X} \\ (\boldsymbol{k}_{\text{sec}}, \boldsymbol{c}) & \longmapsto & \boldsymbol{p} \end{array} \right.$$

$$\text{are such that} \quad \left(\exists \boldsymbol{r} :\ G(\boldsymbol{r}) = (\boldsymbol{k}_{\text{pub}}, \boldsymbol{k}_{\text{sec}})\right) \ \Rightarrow\ \forall \boldsymbol{p}, \boldsymbol{r}',\ \ F^{-1}(\boldsymbol{k}_{\text{sec}}, F(\boldsymbol{k}_{\text{pub}}, \boldsymbol{p}, \boldsymbol{r}')) = \boldsymbol{p}.$$

F performs a possibly probabilistic encryption — hence the random coins $\boldsymbol{r}'$ — and F^{-1} performs the deterministic decryption for a matching secret key.

The necessary requirement for (G, F, F^{-1}) to be an OWTF is that the map $\boldsymbol{p} \mapsto F(\boldsymbol{k}_{\text{pub}}, \boldsymbol{p}, \boldsymbol{r}')$ should be hard to invert when we do not know the secret key $\boldsymbol{k}_{\text{sec}}$ — hence called the *trapdoor* of this map. This one-wayness requirement is captured by a classical computational security game. The challenger uniformly draws random coins $\boldsymbol{r}$ and runs the key generation $G(\boldsymbol{r})$ to get a key pair $(\boldsymbol{k}_{\text{pub}}, \boldsymbol{k}_{\text{sec}})$. Then he uniformly draws a plaintext $\boldsymbol{p} \in \mathcal{X}$ and random coins $\boldsymbol{r}'$, computes $\boldsymbol{c} = F(\boldsymbol{k}_{\text{pub}}, \boldsymbol{p}, \boldsymbol{r}')$ and sends $(\boldsymbol{k}_{\text{pub}}, \boldsymbol{c})$ to the adversary. The adversary has to guess the plaintext $\boldsymbol{p}$.

1.6.2 RSA one-way trapdoor functions

RSA-based cryptosystems originate from the seminal work of Rivest, Shamir and Adleman in 1978 [RSA78]. Their underlying assumption is that the factorization of $N = p \cdot q$ where p and q are two ℓ-bit primes is hard to compute when ℓ is big enough. As an example, Boudot, Gaudry, Guillevic, Heninger and Thomé reported in 2020 the factorization of a 829-bit number from the RSA challenge [Bou+20] while RSA moduli are typically 2048- or 4096-bit long. We explicit in Algorithm 2 the RSA trapdoor functions. The RSA encryption is a deterministic permutation for which the key size is roughly 3ℓ bits and the plaintext and ciphertext size is 2ℓ bits.

Explanation for Algorithm 2. $\phi(N)$ is the cardinality of the multiplicative group $(\mathbb{Z}/N\mathbb{Z})^\star$. ϕ is called the totient Euler function. For a prime p, $\phi(p) = p - 1$ and since $(\mathbb{Z}/N\mathbb{Z})^\star$ is isomorphic to $(\mathbb{Z}/p\mathbb{Z})^\star \times (\mathbb{Z}/q\mathbb{Z})^\star$, we have $\phi(N) = (p-1) \cdot (q-1)$. Since e is coprime with $p-1$ and $q-1$, the inverse d of e in $(\mathbb{Z}/\phi(N)\mathbb{Z})^\star$ is well defined and can be efficiently computed with the extended Euclidian algorithm. Finally, the probability distributions which define how e, p and q are drawn raise subtle questions which are out of our scope. Encryption and decryption are identical and given by F in Algorithm 2. The domain of F varies with the key pair, which is not consistent with our definition of a OWTF. This is however not a problem in practice for building PKEs.

Trapdoor. The definition of $\phi(N)$ and basic group theory ensure that $\forall x \in (\mathbb{Z}/N\mathbb{Z})^\star$, $x^{\phi(N)} = 1$. If $y = x^e \bmod N$ then $y^d = x^{ed} = x \bmod N$. This means that the exponent d allows to inverse the function $x \mapsto x^e$: it is a trapdoor.

One-way. In [RSA78], the authors define the RSA problem as attacking the security game where the challenger runs G on random coins $\boldsymbol{r}$, publishes $\boldsymbol{k}_{\text{pub}} = (e, N)$ and a ciphertext y and challenges the adversary to compute x such that $x^e = y \bmod N$. Obviously, if the adversary finds the factorization $N = p \cdot q$ then she can compute d as if she had run G herself. Factorization is then harder than the RSA problem. If the converse has never been proved, the best way to solve the RSA problem for now is indeed to factor N.

Algorithm 2 Pseudocode for the RSA one-way trapdoor functions.

Key generation G
  input
    ℓ: integer security parameter.
    $\boldsymbol{r}$: random coins.
  output
    $(\boldsymbol{k}_{\text{pub}}, \boldsymbol{k}_{\text{sec}})$: matching key pair.

  Draw $e \geqslant 3$ at random.
  Draw an ℓ-bit prime p coprime with e.
  Draw an ℓ-bit prime $q \neq p$ coprime with e.

  $N \leftarrow p \cdot q$.
  $d \leftarrow e^{-1} \bmod \phi(N)$ with extended Euclidian algorithm.

  $\boldsymbol{k}_{\text{pub}} \leftarrow (e, N)$.
  $\boldsymbol{k}_{\text{sec}} \leftarrow (d, N)$.
  return $(\boldsymbol{k}_{\text{pub}}, \boldsymbol{k}_{\text{sec}})$.

Encryption F
  input
    $\boldsymbol{k} = (e, N)$: public key.
    $x \in (\mathbb{Z}/N\mathbb{Z})^\star$: plaintext.
  output
    $y \in (\mathbb{Z}/N\mathbb{Z})^\star$: ciphertext.

  $y \leftarrow x^e \bmod N$ with fast modular exponentiation.
  return y.

1.6.3 McEliece one-way trapdoor functions

Let q be a prime power, $n \in \mathbb{N}^\star$, $k < n$. In 1978, Berlekamp, McEliece and van Tilborg showed that for a full-rank matrix $\boldsymbol{G} \in \mathbb{F}_q^{k \times n}$, a weight $w \in [1, n]$ and a vector $\boldsymbol{y} \in \mathbb{F}_q^n$, deciding whether there exists $\boldsymbol{x} \in \mathbb{F}_q^k$ and $\boldsymbol{e} \in \mathbb{F}_q^n$ such that $\mathrm{wt}\,(\boldsymbol{e}) \leqslant w$ and $\boldsymbol{y} = \boldsymbol{x} \cdot \boldsymbol{G} + \boldsymbol{e}$ is an NP-complete problem [BMT78]. The associated computational problem is known as the generic decoding problem: given $\boldsymbol{G}$, w and $\boldsymbol{y}$, the goal is to *compute* $\boldsymbol{e}$ and $\boldsymbol{x}$ such that $\boldsymbol{y} = \boldsymbol{x} \cdot \boldsymbol{G} + \boldsymbol{e}$ and $\mathrm{wt}\,(\boldsymbol{e}) \leqslant w$.

The same year, McEliece proposed a cryptosystem based on the hardness of generic decoding [McE78]. Actually, despite the fact that cryptographers mostly care about computational concrete security, an NP-complete problem may be considered a good place to start to build a reasonable computational assumption. Algorithm 3 gives the algorithms for the generic McEliece OWTF (G, F, F^{-1}) with $\mathcal{X} = \mathbb{F}_q^k$, $\mathcal{Y} = \mathbb{F}_q^n$ and $\mathcal{K}_{\text{pub}} = \mathbb{F}_q^{k \times n} \times [1, n]$. The McEliece encryption is a probabilistic injection from $\mathbb{F}_2^k$ to $\mathbb{F}_2^n$ and the generic key size is roughly $k \cdot n$ bits. This big key size is a major drawback compared to RSA, which partly explains why RSA is much more deployed than McEliece in practice.

Explanation for Algorithm 3. Although generic decoding is a hard problem, some families of structured matrices have efficient decoding algorithms. In particular, McEliece used binary Goppa codes in his original proposal [McE78]. Let $(\boldsymbol{G}_i)_{i\in\mathcal{I}}$ be a family of matrices and t be an integer for which decoding is efficient and well defined up to t: for all $\boldsymbol{y}\in\mathbb{F}_2^n$, there exists at most one $\boldsymbol{x}\in\mathbb{F}_2^k$ such that $\mathrm{wt}\,(\boldsymbol{y}-\boldsymbol{x}\cdot\boldsymbol{G}_i)\leqslant t$ and this $\boldsymbol{x}$ can be efficiently computed from $\boldsymbol{G}_i$ and $\boldsymbol{y}$ by a decoding algorithm D.

$$D:\left\{\begin{array}{rcl}\mathcal{I}\times\mathbb{F}_q^n & \longrightarrow & \mathbb{F}_q^k\cup\{\mathsf{reject}\}\\ (i,\boldsymbol{y}) & \longmapsto & \left\{\begin{array}{ll}\boldsymbol{x} & \text{such that } \mathrm{wt}\,(\boldsymbol{y}-\boldsymbol{x}\cdot\boldsymbol{G}_i)\leqslant t,\\ \mathsf{reject} & \text{if } \boldsymbol{x} \text{ does not exist.}\end{array}\right.\end{array}\right.$$

The idea of McEliece in [McE78] is to allow the use of the decoding algorithm D when the secret key is known and to deny it otherwise by *hiding* the structure which allows this efficient decoding.

Trapdoor. $\boldsymbol{P}$ and $\boldsymbol{P}^{-1}$ are both permutation matrices, i.e. matrices with only one non-zero element per line and per column. The action of $\boldsymbol{P}^{-1}$ on a vector $\boldsymbol{e}$ is just a shuffling of its coordinates and keeps the Hamming weight constant: $\mathrm{wt}\left(\boldsymbol{e}\boldsymbol{P}^{-1}\right)=\mathrm{wt}\,(\boldsymbol{e})$. Since $\boldsymbol{y}=\boldsymbol{c}\boldsymbol{P}^{-1}=\boldsymbol{p}\boldsymbol{S}\boldsymbol{G}_i+\boldsymbol{e}\boldsymbol{P}^{-1}$, the decoding algorithm D outputs $\boldsymbol{x}=\boldsymbol{p}\boldsymbol{S}$. Decryption is correct and the decomposition $(\boldsymbol{S},i,\boldsymbol{P})$ is indeed a trapdoor.

One-way. The security of McEliece highly depends on the family of matrices being used. Obviously, the security also depends on how *hiding the structure* is performed. Even if Algorithm 3 shows the original proposition of McEliece to hide the structure of the secret $\boldsymbol{G}_i$ into the public $\boldsymbol{G}_{\mathrm{pub}}$ with matrices $\boldsymbol{S}$ and $\boldsymbol{P}$, there are other possibilities. In a nutshell, the one-wayness of McEliece can be considered to hold as long as there is no better way to perform decoding for a public key $(\boldsymbol{G}_{\mathrm{pub}},t)$ than the generic decoding and as long as the lenght n and the dimension k make the generic decoding infeasible in practice.

1.6.4 PKE security notions and designs

Like symmetric-key encryption schemes, the main security notions for PKE schemes are based on distinguishing games. There is one big difference though. In the symmetric-key IND-CPA for example, the challenger draws a secret key at random at the very beginning of the game and waits for the queries of the adversary. In the asymmetric-key setting, he will rather run the key-generation algorithm and publish the public key at the beginning of the game. Moreover, for the asymmetric IND-CPA game, the chosen plaintext queries are not really granted to the adversary as she can encrypt as much as she likes with the public key without querying the challenger. Cryptographers say that semantic security implies CPA security in the public-key setting.

As we said at the beginning of Section 1.6, PKE schemes are built around a one-way trapdoor function or a basic key-exchange protocol. More precisely, a scheme which aims at encrypting possibly large messages will follow a hybrid construction involving a key-encapsulation mechanism and a symmetric-key cipher. For instance, it will draw some random nonce, encrypt the nonce with the OWTF (or with some algorithm derived from a key-exchange), derive a symmetric session key from the nonce with a hash function (see Section 1.5.1) and finally encrypt the message with a secure symmetric-key cipher under the computed key. The security proof will be made under the assumptions that the OWTF is one-way (which assumes in turn that some computational problem is hard), that the symmetric-key cipher is secure and that the hash function is secure as well.

Algorithm 3 Pseudocode for McEliece one-way trapdoor functions.

Key generation G
  input
    n, k: integer security parameters.
    $\boldsymbol{r}$: random coins.
  output
    $(\boldsymbol{k}_{\text{pub}}, \boldsymbol{k}_{\text{sec}})$: matching key pair.

  Draw i at random in $\mathcal{I}$.
  Draw an invertible matrix $\boldsymbol{S} \in \mathrm{GL}_k(\mathbb{F}_q)$.
  Draw a permutation matrix $\boldsymbol{P} \in \mathbb{F}_q^{n \times n}$.

  $\boldsymbol{G}_{\text{pub}} \leftarrow \boldsymbol{S} \cdot \boldsymbol{G}_i \cdot \boldsymbol{P}$. ▷ Hide the structure of $\boldsymbol{G}_i$ with $\boldsymbol{S}$ and $\boldsymbol{P}$.

  $\boldsymbol{k}_{\text{pub}} \leftarrow (\boldsymbol{G}_{\text{pub}}, t)$.
  $\boldsymbol{k}_{\text{sec}} \leftarrow (\boldsymbol{S}, i, \boldsymbol{P})$.
  return $(\boldsymbol{k}_{\text{pub}}, \boldsymbol{k}_{\text{sec}})$.

Encryption F
  input
    $\boldsymbol{k}_{\text{pub}} = (\boldsymbol{G}_{\text{pub}}, t)$: public key.
    $\boldsymbol{p} \in \mathbb{F}_2^k$: plaintext.
    $\boldsymbol{r}$: random coins.
  output
    $\boldsymbol{c} \in \mathbb{F}_2^n$: ciphertext.
  Draw $\boldsymbol{e}$ at random such that $\mathrm{wt}(\boldsymbol{e}) = t$.
  $\boldsymbol{c} \leftarrow \boldsymbol{p} \cdot \boldsymbol{G}_{\text{pub}} + \boldsymbol{e}$.
  return $\boldsymbol{c}$.

Decryption F^{-1}
  input
    $\boldsymbol{k}_{\text{sec}} = (\boldsymbol{S}, i, \boldsymbol{P})$: private key.
    $\boldsymbol{c} \in \mathbb{F}_2^n$: ciphertext. ▷ $\boldsymbol{c} = \boldsymbol{pSG}_i\boldsymbol{P} + \boldsymbol{e}$.
  output
    $\boldsymbol{p} \in \mathbb{F}_2^n$: plaintext.
  $\boldsymbol{y} \leftarrow \boldsymbol{c} \cdot \boldsymbol{P}^{-1}$. ▷ $\boldsymbol{y} = \boldsymbol{pSG}_i + \boldsymbol{eP}^{-1}$.
  $\boldsymbol{x} \leftarrow D(i, \boldsymbol{y})$. ▷ Use the decoding algorithm for the secret $\boldsymbol{G}_i$.
  $\boldsymbol{p} \leftarrow \boldsymbol{x} \cdot \boldsymbol{S}^{-1}$.
  return $\boldsymbol{p}$.

In short, a hybrid PKE scheme is like a small protocol which starts with a key exchange and encrypts messages with a symmetric-key cipher. However, for bigger protocols with interaction between parties, it would be a computational shame to perform a key exchange for each message. As we saw in Section 1.1.2 for TLS, protocols rather perform just one key exchange at the beginning of the conversation. This is why there is another flavour of PKE schemes, called *padding schemes*, whose role is to turn a OWTF into a key exchange. In other words, a padding scheme is a PKE scheme for encrypting short messages — typically nonces and symmetric secret keys — and does not need a symmetric-key cipher. Well known padding schemes for RSA include PKCS 1 by RSA Laboratories and OAEP by Bellare and Rogaway [Mor+16; BR95].

1.7 Conclusion

In this chapter, we presented a famous use-case of cryptography with the example of the TLS protocol, the formalism of security in the computational model and several encryption-scheme designs with a clear emphasis on the symmetric-key setting. However, this chapter is not enough to give a full explanation of how TLS achieves its cryptographic goals. Indeed, there are other important cryptosystems like key exchanges and digital signatures, other important security notions for PKE schemes like non-malleability or CCA security but their study is quite far from the scope of this thesis. The goal of this introduction to modern cryptography was mainly to explain how trusting primitives allows to trust complex cryptosystems and how these primitives are built in practice.

This thesis mainly focuses on studying security conjectures of symmetric-key primitives like block ciphers. In Chapter 3, we explore a family of distinguishers on SPN ciphers. In Chapter 4, we provide techniques to turn a MILP solver into an automated cryptanalysis tool for SPN ciphers. In Chapter 5, we try to enhance a family of algebraic attacks mostly suited to keystream generators. Finally in Chapter 6, we study the security of a rank-metric-based McEliece one-way trapdoor function.

The next chapter is dedicated to a presentation of differential cryptanalysis, which will be useful for Chapters 3 and 4.

Chapter 2

Differential cryptanalysis

As explained in Section 1.4.1, block ciphers are primitives in the sense that their PRP security is conjectured. Moreover, we saw in Sections 1.4.4, 1.5.2 and 1.5.3 that they are the central building-blocks for most symmetric-key cryptosystems and that these cryptosystems' security actually relies on the conjectured PRP security of the underlying block cipher. In other words, to trust these cryptosystems used daily by millions of people worldwide, we have to keep confidence in the PRP conjecture of the block cipher. It is therefore of crucial importance to steadily enlarge our knowledge on the possible attacks against classical block-cipher architectures, like the SPN construction introduced in Section 1.4.1 and illustrated in Section 1.4.2 on the example of the `PRESENT` block cipher. In this chapter, we present a famous family of attacks called differential cryptanalysis and how it applies to SPN constructions with an emphasis on the Advanced Encryption Standard (AES).

2.1 Distinguishing with differentials

Let $B : \mathbb{F}_2^m \times \mathbb{F}_2^n \to \mathbb{F}_2^n$ be a block cipher. Statistical attacks aim at winning the PRP game by exhibiting a predicate which holds with good probability when the secret key is uniformly distributed. This predicate will be central in the strategy of the adversary and it has to be found *before* playing the game. This kind of computation prior to the game is often called an *offline* phase while the strategy of the adversary during the game is called the *online* phase. In the particular case of differential cryptanalysis, introduced by Biham and Shamir in [BS91], the predicate is of the form

$$B_{\boldsymbol{k}}(\boldsymbol{x} + \boldsymbol{a}) + B_{\boldsymbol{k}}(\boldsymbol{x}) = \boldsymbol{b}$$

for a specific pair of input and output differences $(\boldsymbol{a}, \boldsymbol{b})$ and a random $\boldsymbol{x}$.

In Section 2.1.1, we give a basic analysis of how differentials appear in random functions and we sketch in Section 2.1.2 the idea of how differential cryptanalysis can give a strategy to win a PRF game.

2.1.1 Differentials in random functions

We first recall some well-known results on the links between the Kullback-Leibler divergence and the tails of binomial distributions. We then properly define differentials and we study the probability that a random function verifies a given differential.

Definition 2.1 (Kullback-Leibler divergence)**.** Let $(p,q) \in]0,1[^2$. The Kullback-Leibler divergence between p and q is defined by

$$D(p \,\|\, q) \stackrel{\text{def}}{=} p \log \frac{p}{q} + (1-p) \log \frac{1-p}{1-q}.$$

Remark 2.1. $D(p \,\|\, p) = 0$ and since log is strictly concave, $D(p \,\|\, q) > 0$ if $p \neq q$.

Proposition 2.1 (Chernoff-Hoeffding bound)**.** *Let X be a random variable with binomial distribution (N,p). Then for all $k \geqslant N \cdot p$,*

$$\mathbb{P}(X \geqslant k) \leqslant \exp\left(-N \cdot D\left(\frac{k}{N} \,\middle\|\, p\right)\right).$$

Proof. This result has been proved by Hoeffding in [Hoe62] as Theorem 1 for sums of independent variables. We give here a succint proof for our specific case.

We can write $X = \sum_{i=1}^{N} X_i$ where the $(X_i)_i$ are independent Bernoulli variables with probability p. For all $y > 0$,

$$\begin{aligned}
\mathbb{P}(X \geqslant k) &= \mathbb{P}(\exp(yX) \geqslant \exp(yk)) \\
&\leqslant \mathbb{E}(\exp(yX)) \cdot \exp(-yk) \ \text{(Markov inequality)} \\
&\leqslant \left(\prod_{i=1}^{N} \mathbb{E}(\exp(yX_i))\right) \cdot \exp(-yk) \ \text{by independence.}
\end{aligned}$$

For all i, $\mathbb{E}(\exp(yX_i)) = (1-p) + p\exp y$. Therefore, $\mathbb{P}(X \geqslant k) \leqslant f(y)^N$ where

$$\begin{aligned}
f(y) &= ((1-p) + p\exp y) \cdot \exp\left(-y\frac{k}{N}\right) \\
\text{then} \quad f'(y) &= \left(-\frac{k}{N}(1-p) + p\left(1-\frac{k}{N}\right)\exp y\right) \cdot \exp\left(-y\frac{k}{N}\right).
\end{aligned}$$

$f'(y_0) = 0 \iff y_0 = \log \frac{k/N}{p} \cdot \frac{1-p}{1-k/N}$. With such a y_0, we have that $\log f(y_0) = -D\left(\frac{k}{N} \,\middle\|\, p\right)$, which ends the proof. □

Proposition 2.2. *Let X be a random variable with binomial distribution (N,p). Then for all $k \geqslant N \cdot p$,*

$$\mathbb{P}(X \geqslant k) \geqslant \mathbb{P}(X = k) \geqslant \frac{\sqrt{2\pi}}{e^2} \cdot \left(k\left(1-\frac{k}{N}\right)\right)^{-1/2} \exp\left(-N \cdot D\left(\frac{k}{N} \,\middle\|\, p\right)\right).$$

Proof. Robbins gives in [Rob55] useful bounds for the Stirling formula. In particular for all $n \geqslant 2$,

$$\sqrt{2\pi} < \frac{n!}{n^{n+1/2}e^{-n}} < e \text{ and } \binom{N}{k} \geqslant \frac{\sqrt{2\pi}}{e^2} \cdot \frac{N^k}{\left(1-\frac{k}{N}\right)^{N-k}} \cdot \left(k\left(1-\frac{k}{N}\right)\right)^{-1/2}.$$

From there, we can deduce the result with $\mathbb{P}(X = k) = \binom{N}{k} p^k (1-p)^{N-k}$. □

Table 2.1 – Lower and upper bounds for $\mathbb{P}\left(p\left(F, \boldsymbol{a} \to \boldsymbol{b}\right) \geqslant 2^{t-n}\right)$ when F is a random function and $n \geqslant 96$.

t	2	3	4	5	6	7
Lower bound	$2^{-3.90}$	$2^{-9.52}$	$2^{-24.24}$	$2^{-61.20}$	$2^{-150.62}$	$2^{-360.95}$
Upper bound	$2^{-1.83}$	$2^{-6.95}$	$2^{-21.17}$	$2^{-57.63}$	$2^{-146.55}$	$2^{-356.38}$

Definition 2.2 (Differential)**.** Let $F : \mathbb{F}_2^n \to \mathbb{F}_2^n$ and $\boldsymbol{a}, \boldsymbol{b} \in \mathbb{F}_2^n$. We define

$$p\left(F, \boldsymbol{a} \to \boldsymbol{b}\right) \stackrel{\text{def}}{=} 2^{-n} \cdot \sum_{\boldsymbol{x} \in \mathbb{F}_2^n} \delta\left(F\left(\boldsymbol{x} + \boldsymbol{a}\right) + F\left(\boldsymbol{x}\right), \boldsymbol{b}\right)$$

where δ is the Kronecker symbol: $\delta(x, y) = 1$ if $x = y$, 0 otherwise. $p\left(F, \boldsymbol{a} \to \boldsymbol{b}\right)$ is essentially the probability that $F\left(\boldsymbol{x} + \boldsymbol{a}\right) + F\left(\boldsymbol{x}\right) = \boldsymbol{b}$ for a uniformly random $\boldsymbol{x}$ but the equation above gives a more rigorous definition when F itself is a random variable. The pair of differences $(\boldsymbol{a}, \boldsymbol{b})$ is called a *differential*.

Remark 2.2. We have that $p\left(F, \boldsymbol{0} \to \boldsymbol{0}\right) = 1$, $p\left(F, \boldsymbol{a} \to \boldsymbol{0}\right) = 0$ if $\boldsymbol{a} \neq \boldsymbol{0}$ and $p\left(F, \boldsymbol{0} \to \boldsymbol{b}\right) = 0$ if $\boldsymbol{b} \neq \boldsymbol{0}$. Let $\boldsymbol{a} \neq \boldsymbol{0}$ and $\boldsymbol{b} \neq \boldsymbol{0}$. Since we can partition $\mathbb{F}_2^n$ in $\mathcal{S} \cup (\mathcal{S} + \boldsymbol{a})$ for some set $\mathcal{S}$ such that $\boldsymbol{x} \in \mathcal{S} \Rightarrow \boldsymbol{x} + \boldsymbol{a} \notin \mathcal{S}$,

$$2^{n-1} \cdot p\left(F, \boldsymbol{a} \to \boldsymbol{b}\right) = \sum_{\boldsymbol{x} \in \mathcal{S}} \delta\left(F\left(\boldsymbol{x} + \boldsymbol{a}\right) + F\left(\boldsymbol{x}\right), \boldsymbol{b}\right) \in \mathbb{N}.$$

Let $(\boldsymbol{a}, \boldsymbol{b})$ be a fixed non-zero differential, $\mathcal{S} \cup (\mathcal{S} + \boldsymbol{a})$ be a partition of $\mathbb{F}_2^n$ and F be a uniformly random function. The δ-terms $\delta\left(F(\boldsymbol{x} + \boldsymbol{a}) + F(\boldsymbol{x}), \boldsymbol{b}\right)$ for $\boldsymbol{x} \in \mathcal{S}$ are thus independent Bernoulli variables with probability 2^{-n}. Moreover, we have with Remark 2.2 that $2^{n-1} p\left(F, \boldsymbol{a} \to \boldsymbol{b}\right)$ is a sum of independent and identically distributed Bernoulli variables and has thus a binomial distribution $(N, p) = (2^{n-1}, 2^{-n})$.

Now, let $t \in [1, n]$ and $f(n, t) \stackrel{\text{def}}{=} \exp\left(-2^{n-1} \cdot D\left(2^{t-n} \,\|\, 2^{-n}\right)\right)$. We have with Proposition 2.1 and Proposition 2.2 that

$$\frac{\sqrt{2\pi}}{e^2} \cdot \left(2^{t-1}\left(1 - 2^{t-n}\right)\right)^{-1/2} \cdot f(n, t) \leqslant \mathbb{P}\left(p\left(F, \boldsymbol{a} \to \boldsymbol{b}\right) \geqslant 2^{t-n}\right) \leqslant f(n, t).$$

When t is fixed,

$$\lim_{n \to +\infty} \log_2 f(n, t) = -t \cdot 2^{t-1} + \frac{1}{\log 2} 2^{t-1} - \frac{1}{2 \log 2}.$$

We give numerical evaluations of both sides for various values of t in Table 2.1. It seems through simple numerical experiments that $f(n, t)$ and the lower bound are very close to their respective limits for $t \leqslant 7$ and $n \geqslant 96$. In fact, for bigger values of n, it is more precise to use the limit than the computations on big `floats`.

2.1.2 Differentials for PRF adversaries.

In this section, we study how the knowledge of a differential in a block cipher can be used to build an adversary against its PRF-security. We chose the PRF- rather than PRP-security because we

studied the behaviour of differentials in random *functions* in Section 2.1.1. Indeed, with random functions, the different outputs are probabilistically independent, which makes the analysis with binomials accurate.

Let B be a block cipher with block size n. We assume that we know a differential $(\boldsymbol{a}, \boldsymbol{b})$ for B such that $p\left(B_{\boldsymbol{k}}, \boldsymbol{a} \to \boldsymbol{b}\right) > 2^{t_B - n}$ for a non-negligeable fraction of the keys and for some $t_B \geqslant 2$. Then we can build an adversary able to distinguish between $B_{\boldsymbol{k}}$ for a random key $\boldsymbol{k}$ and a random function F. The adversary with $2N$ queries will query the encryptions of $\boldsymbol{x}$ and $\boldsymbol{x} + \boldsymbol{a}$ for N elements $\boldsymbol{x} \in \mathcal{S}$ and count the number of times the corresponding output difference is $\boldsymbol{b}$. Then the adversary will answer that the challenger is in Real mode if and only if this number is bigger than 2^{t_B}.

Let $\boldsymbol{k}$ be a key such that $p_B \stackrel{\text{def}}{=} p\left(B_{\boldsymbol{k}}, \boldsymbol{a} \to \boldsymbol{b}\right) > 2^{t_B - n}$ and F be a function such that $p_F \stackrel{\text{def}}{=} p\left(F, \boldsymbol{a} \to \boldsymbol{b}\right) < 2^{t_F - n}$ for some $t_F < t_B$. We fix $\boldsymbol{k}$ and F to avoid mixing probabilities because the adversary is itself probabilistic. In [BGT11], Blondeau, Gérard and Tillich estimate the necessary number of queries N for this adversary to make the correct guess with those fixed $\boldsymbol{k}$ and F. It is easy to see that the non-detection probability, i.e. the probability that the adversary answers Random instead of Real while interacting with $B_{\boldsymbol{k}}$, is less than $\frac{1}{2}$. Let β be a chosen false alarm probability, i.e. the probability with which we allow the adversary to answer Real instead of Random while interacting with F. Easy computations give

$$D\left(p_B \,\|\, p_F\right) \geqslant D\left(2^{t_B - n} \,\middle\|\, 2^{t_F - n}\right) \quad \text{and}$$
$$\lim_{n \to +\infty} 2^n D\left(2^{t_B - n} \,\middle\|\, 2^{t_F - n}\right) = (t_B - t_F) 2^{t_B} \log 2 + 2^{t_F} - 2^{t_B}.$$

Theorem 2 in [BGT11] shows we can estimate an upper bound on N with

$$\frac{-\log(2\sqrt{\pi}\beta)}{D\left(2^{t_B - n} \,\|\, 2^{t_F - n}\right)} \underset{n \to +\infty}{\sim} \frac{-\log(2\sqrt{\pi}\beta)}{(t_B - t_F) 2^{t_B} \log 2 + 2^{t_F} - 2^{t_B}} 2^n.$$

A numerical evaluation with $t_F = 6$, $t_B = 70$ and $\beta = 2^{-32}$ shows that $N = 2^{n-71}$ allows the adversary to distinguish between $B_{\boldsymbol{k}}$ and F. As shown in Table 2.1, $t_F = 6$ ensures that when F is drawn uniformly at random, then $p_F \geqslant 2^{t_F - n}$ with probability less than 2^{-146}. Therefore, if the differential $(\boldsymbol{a}, \boldsymbol{b})$ has probability bigger than 2^{70-n} for all the keys and the block size is $n = 128$, this differential allows to distinguish the block cipher from a random function more efficiently than the birthday paradox. The formulation of the problem in [BGT11] slightly differs from ours since they consider the classical goal of recovering the key rather than distinguishing the block cipher from a random permutation. Actually, since the original work of Biham and Shamir [BS91], differentials have mostly been used to mount key-recoveries.

We have just presented a complexity estimation of the online phase but we still need techniques to find good probability differentials during the offline phase. In Section 2.2, we explain the basic exhaustive search and in Section 2.3, we explain the classical strategy for finding differentials in iterative block ciphers.

2.2 Exhaustive knowledge of differentials

Let $F : \mathbb{F}_2^n \to \mathbb{F}_2^n$ be an arbitrary map. The goal of this section is to introduce the tools and techniques to compute the value $p\left(F, \boldsymbol{a} \to \boldsymbol{b}\right)$ for *all* $\boldsymbol{a}, \boldsymbol{b} \in \mathbb{F}_2^n$.

Algorithm 4 Compute the DDT of a function $F : \mathbb{F}_2^n \to \mathbb{F}_2^n$.

```
Create a table DDT of dimensions 2^n × 2^n for n-bit integers initialized to 0.
for all a ∈ F_2^n do
    for all x ∈ F_2^n do
        Increment DDT(a, F(x + a) + F(x)) by 1.
return DDT
```

The Difference Distribution Table (DDT) of F is the mapping

$$(\boldsymbol{a}, \boldsymbol{b}) \in \mathbb{F}_2^{2n} \mapsto \sum_{\boldsymbol{x} \in \mathbb{F}_2^n} \delta\left(F\left(\boldsymbol{x} + \boldsymbol{a}\right) + F\left(\boldsymbol{x}\right), \boldsymbol{b}\right).$$

Thus, $\mathrm{DDT}\left(\boldsymbol{a}, \boldsymbol{b}\right) = 2^n \cdot p\left(F, \boldsymbol{a} \to \boldsymbol{b}\right) \in 2 \cdot \mathbb{N}$. A basic algorithm to compute the DDT from the value table of F is given as Algorithm 4. It needs 2^{2n} evaluations of F and the memory to store the DDT, 2^{2n} n-bit integers. If we do not need to know the whole DDT but we need to go over all its values, Algorithm 4 can be easily adapted to store just the current line of the DDT. In that case, the memory complexity is just 2^n n-bit integers. In fact, this can be useful if we are only interested in the maximum of the DDT for non-zero differentials, called the differential uniformity.

Definition 2.3 (Differential uniformity [Nyb94])**.** Let $F : \mathbb{F}_2^n \to \mathbb{F}_2^n$. Its differential uniformity $\mathrm{DU}\left(F\right)$ is defined by

$$\mathrm{DU}\left(F\right) \stackrel{\text{def}}{=} \max_{\boldsymbol{a} \neq 0, \boldsymbol{b}} \mathrm{DDT}\left(\boldsymbol{a}, \boldsymbol{b}\right).$$

Remark 2.3. It follows from Remark 2.2 that for any function F, its differential uniformity is even and at least 2. Besides, the differential uniformity is not changed by the composition of additive isomorphisms [Nyb94, Prop. 1].

The DDT is in fact the most basic tool to understand the differential properties of `Sbox`es. Indeed, `Sbox`es are small (up to $n = 8$ bits in general) and the basic Algorithm 4 is enough to compute their DDTs. In particular, the AES `Sbox` has differential uniformity 4. Actually, it has been shown by Nyberg in [Nyb94, Prop. 6] that the extended inverse of any finite field has differential uniformity 4. On the contrary, block ciphers have a block size $n = 128$ and it is impossible in practice to go through the values of the DDT of a block cipher.

Example 2.1. The `Sbox` of `PRESENT`, given in Table 1.1, has the DDT given in Section A.1.

2.3 Differential characteristics

Differential characteristics, also called differential trails, are a powerful tool to find good-probability differentials in an iterative block cipher. In this section, we first define what they are and then explain how the AES is, by design, resistant to the existence of good-probability differential characteristics.

2.3.1 Definition and estimation

For a key-alternating cipher like the AES, with round-function R, let us imagine that we can trace the possible differences in the state from a chosen input difference $\boldsymbol{a} = \boldsymbol{a}^{(0)}$ to a chosen

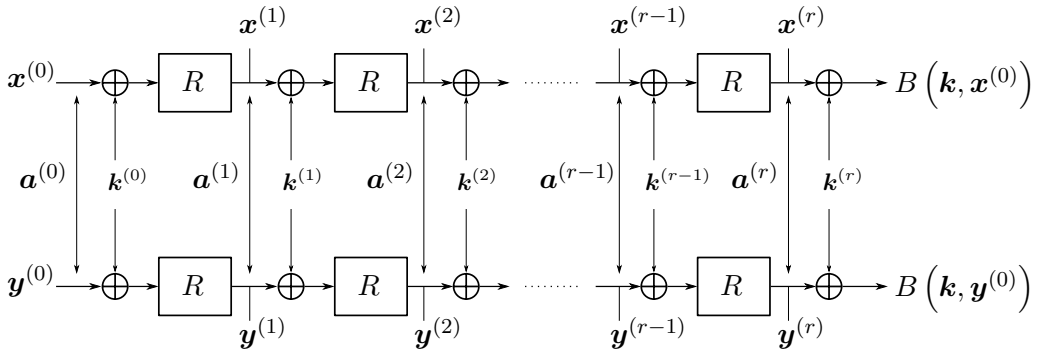

Figure 2.1 – The states during the encryptions of $\boldsymbol{x}^{(0)}$ and $\boldsymbol{y}^{(0)} = \boldsymbol{x}^{(0)} + \boldsymbol{a}$ verify the differential characteristic $\left(\boldsymbol{a}^{(0)}, \ldots, \boldsymbol{a}^{(r)}\right)$.

output difference $\boldsymbol{b} = \boldsymbol{a}^{(r)}$. We denote by ${R_{\boldsymbol{k}}}^{(i)}$ the map $\boldsymbol{x} \mapsto R\left(\boldsymbol{k}^{(i-1)} + \boldsymbol{x}\right)$ and by ${B_{\boldsymbol{k}}}^{(i)}$ the map $\boldsymbol{x} \mapsto {R_{\boldsymbol{k}}}^{(i)} \circ \cdots \circ {R_{\boldsymbol{k}}}^{(1)}(\boldsymbol{x})$.

Definition 2.4 (Differential characteristic)**.** With the above notation, we call an $(r+1)$-uple $\left(\boldsymbol{a}^{(0)}, \ldots, \boldsymbol{a}^{(r)}\right)$ a *differential characteristic* and we define

$$p_{\boldsymbol{k}}\left(\boldsymbol{a}^{(0)}, \ldots, \boldsymbol{a}^{(r)}\right) \stackrel{\text{def}}{=} 2^{-n} \cdot \sum_{\boldsymbol{x} \in \mathbb{F}_2^n} \prod_{i=1}^{r} \delta\left({B_{\boldsymbol{k}}}^{(i)}(\boldsymbol{x} + \boldsymbol{a}) + {B_{\boldsymbol{k}}}^{(i)}(\boldsymbol{x}), \boldsymbol{a}^{(i)}\right).$$

$p_{\boldsymbol{k}}\left(\boldsymbol{a}^{(0)}, \ldots, \boldsymbol{a}^{(r)}\right)$ is essentially the probability that for all $i \in [1, r]$, the states after round i of the encryptions of a random $\boldsymbol{x}$ and $\boldsymbol{x} + \boldsymbol{a}$ have a difference $\boldsymbol{a}^{(i)}$. This is illustrated in Figure 2.1

What makes differential characteristics interesting is the relation

$$p\left(B_{\boldsymbol{k}}, \boldsymbol{a} \to \boldsymbol{b}\right) = \sum_{\left(\boldsymbol{a}^{(1)}, \ldots, \boldsymbol{a}^{(r-1)}\right)} p_{\boldsymbol{k}}\left(\boldsymbol{a}, \boldsymbol{a}^{(1)}, \ldots, \boldsymbol{a}^{(r-1)}, \boldsymbol{b}\right).$$

This relation means that if an attacker can find a differential characteristic such that its $p_{\boldsymbol{k}}$ value is high enough, then the input end $\boldsymbol{a}$ and the output end $\boldsymbol{b}$ of the characteristic make a good pair of differences to win the PRP game with a differential cryptanalysis.

To find a good differential characteristic and estimate its $p_{\boldsymbol{k}}$ value, it is common to assume that

$$\begin{aligned}
&\mathbb{P}\left(\boldsymbol{y}^{(i+1)} + \boldsymbol{x}^{(i+1)} = \boldsymbol{a}^{(i+1)} \,\middle|\, \boldsymbol{y}^{(i)} + \boldsymbol{x}^{(i)} = \boldsymbol{a}^{(i)} \cap \cdots \cap \boldsymbol{y}^{(1)} + \boldsymbol{x}^{(1)} = \boldsymbol{a}^{(1)}\right) \\
&= \mathbb{P}\left(\boldsymbol{y}^{(i+1)} + \boldsymbol{x}^{(i+1)} = \boldsymbol{a}^{(i+1)} \,\middle|\, \boldsymbol{y}^{(i)} + \boldsymbol{x}^{(i)} = \boldsymbol{a}^{(i)}\right) \\
&= p\left({R_{\boldsymbol{k}}}^{(i+1)}, \boldsymbol{a}^{(i)} \to \boldsymbol{a}^{(i+1)}\right)
\end{aligned}$$

where $\boldsymbol{x}$ is random, $\boldsymbol{x}^{(i)} = {B_{\boldsymbol{k}}}^{(i)}(\boldsymbol{x})$ and $\boldsymbol{y}^{(i)} = {B_{\boldsymbol{k}}}^{(i)}(\boldsymbol{x} + \boldsymbol{a})$. In their seminal work [LMM91], Lai, Massey and Murphy call this assumption the *Markov cipher* model because the sequence of random variables $\left(\boldsymbol{x}^{(i)} + \boldsymbol{y}^{(i)}\right)_{i \geqslant 1}$ is a Markov chain. In particular, this assumption holds if the round-keys of the key-alternating cipher are independent and uniformly distributed. It is obviously not the case in practice as round-keys are derived from the master key through the key expansion, but the goal

of making this assumption is mainly to give an estimation of the value $p_{\boldsymbol{k}}\left(\boldsymbol{a}^{(0)},\ldots,\boldsymbol{a}^{(r)}\right)$. Indeed, with independent and uniform round-keys, we have that

$$p_{\boldsymbol{k}}\left(\boldsymbol{a}^{(0)},\ldots,\boldsymbol{a}^{(r)}\right)=\prod_{i=0}^{r-1}p\left(R,\boldsymbol{a}^{(i)}\rightarrow\boldsymbol{a}^{(i+1)}\right). \tag{2.1}$$

A major advantage of this estimation of $p_{\boldsymbol{k}}\left(\boldsymbol{a}^{(0)},\ldots,\boldsymbol{a}^{(r)}\right)$ is that it is independent of the key $\boldsymbol{k}$, making the analysis of differential characteristics a bit simpler. Moreover, Daemen and Rijmen write in Chapter 8 of [DR01a] that such an estimation is quite accurate when the round-function provides good confusion and diffusion and when the value of the estimation is not below 2^{-n+1}. Indeed, an estimation below 2^{-n+1} would not really have much sense as $2^{n-1}p_{\boldsymbol{k}}\left(\boldsymbol{a}^{(0)},\ldots,\boldsymbol{a}^{(r)}\right)$ is an integer. Equation (2.1) finally highlights the importance of studying the propagation of differences through successive applications of the round-function.

2.3.2 The wide-trail strategy

From a designer's point of view, we want our design to resist differential cryptanalysis. Consequently, we want to avoid the existence of good probability differential characteristics. In [DR01b], Daemen and Rijmen explain their *wide-trail* design strategy (first presented in [Dae95]) to bound the quantity $p\left(\boldsymbol{a}^{(0)},\ldots,\boldsymbol{a}^{(4)}\right)$ for 4 rounds of a key-alternating cipher assumed to behave like a Markov cipher. In particular, they used this strategy to design the AES. In this section, we briefly illustrate their ideas with the design of the AES and we first need the definition of the differential branch number.

Definition 2.5 (Hamming weight)**.** Let $\boldsymbol{x}\in\mathbb{K}^N$ for some field $\mathbb{K}$ and a positive integer N. The support and the Hamming weight of $\boldsymbol{x}$ with respect to $\mathbb{K}$ are respectively defined by

$$\mathrm{supp}_{\mathbb{K}}\left(\boldsymbol{x}\right)\stackrel{\text{def}}{=}\left\{\,i\mid x_i\in\mathbb{K}\backslash\{0\}\,\right\}\quad\text{ and }\quad\mathrm{wt}_{\mathbb{K}}\left(\boldsymbol{x}\right)\stackrel{\text{def}}{=}\left|\mathrm{supp}_{\mathbb{K}}\left(\boldsymbol{x}\right)\right|.$$

Definition 2.6 (Differential branch number [Dae95])**.** The differential branch number of the $\mathbb{K}$-linear function $\boldsymbol{x}\mapsto\boldsymbol{M}\cdot\boldsymbol{x}$ with respect to $\mathbb{K}$ is

$$\min_{\boldsymbol{x}\neq 0}\mathrm{wt}_{\mathbb{K}}\left(\boldsymbol{x}\right)+\mathrm{wt}_{\mathbb{K}}\left(\boldsymbol{M}\cdot\boldsymbol{x}\right)$$

We now explicit how to bound the probability of characteristics from the `SubBytes` operation to 4 full rounds. The different bounds will be of the form $\left(\mathrm{DU}\left(\texttt{Sbox}\right)\cdot 2^{-8}\right)^W$ where W is called the number of *active* Sboxes.

Characteristics in `SubBytes`.

Since `SubBytes` is a parallel application of the Sbox, we have for all $\boldsymbol{a},\boldsymbol{b}\in\mathcal{M}_4\left(\mathbb{K}\right)$ that

$$p\left(\texttt{SubBytes},\boldsymbol{a}\rightarrow\boldsymbol{b}\right)=\prod_{i,j=0}^{3}p\left(\texttt{Sbox},a_{i,j}\rightarrow b_{i,j}\right).$$

Moreover, the Sbox is a permutation of $\mathbb{K}$ and then for all $a \in \mathbb{K}^\star$, $p(\texttt{Sbox}, a \to 0) = p(\texttt{Sbox}, 0 \to a) = 0$ and $p(\texttt{Sbox}, 0 \to 0) = 1$. Finally,

$$\begin{aligned} p(\texttt{SubBytes}, \boldsymbol{a} \to \boldsymbol{b}) &= \delta(\mathrm{supp}_{\mathbb{K}}(\boldsymbol{a}), \mathrm{supp}_{\mathbb{K}}(\boldsymbol{b})) \cdot \prod_{(i,j)\in \mathrm{supp}_{\mathbb{K}}(\boldsymbol{a})} p(\texttt{Sbox}, a_{i,j} \to b_{i,j}) \\ &\leqslant \left(\frac{\mathrm{DU}(\texttt{Sbox})}{2^8}\right)^{\mathrm{wt}_{\mathbb{K}}(\boldsymbol{a})}. \end{aligned}$$

To reduce the probability of the differential, the Sbox should have a minimum differential uniformity and since for an 8-bit Sbox, the best known differential uniformity is 4, the AES Sbox has been chosen to verify $\mathrm{DU}(\texttt{Sbox}) = 4$. For a given differential $(\boldsymbol{a}, \boldsymbol{b})$ with $\mathrm{supp}_{\mathbb{K}}(\boldsymbol{a}) = \mathrm{supp}_{\mathbb{K}}(\boldsymbol{b})$, the bytes at position $(i,j) \in \mathrm{supp}_{\mathbb{K}}(\boldsymbol{a})$ are called *active* Sboxes.

Characteristics in R.

Since `ShiftRows` and `MixColumns` are linear operations, they propagate differences deterministically and it is clear that for all $(\boldsymbol{a}, \boldsymbol{b})$,

$$\begin{aligned} p(R, \boldsymbol{a} \to \boldsymbol{b}) &= p\left(\texttt{SubBytes}, \boldsymbol{a} \to \texttt{ShiftRows}^{-1} \circ \texttt{MixColumns}^{-1}(\boldsymbol{b})\right) \\ &\leqslant \left(\mathrm{DU}(\texttt{Sbox}) \cdot 2^{-8}\right)^{\mathrm{wt}_{\mathbb{K}}(\boldsymbol{a})}. \end{aligned}$$

Characteristics in $\mathrm{AES}^{(2)}$.

Let $\left(\boldsymbol{a}^{(0)}, \boldsymbol{a}^{(1)}, \boldsymbol{a}^{(2)}\right) \in \mathcal{M}_4(\mathbb{K})^3$ and $\boldsymbol{b}^{(i)} = \texttt{MixColumns}^{-1}\left(\boldsymbol{a}^{(i)}\right)$. Then

$$\begin{aligned} p\left(\boldsymbol{a}^{(0)}, \boldsymbol{a}^{(1)}, \boldsymbol{a}^{(2)}\right) &= p\left(R, \boldsymbol{a}^{(0)} \to \boldsymbol{a}^{(1)}\right) \cdot p\left(R, \boldsymbol{a}^{(1)} \to \boldsymbol{a}^{(2)}\right) \\ &\leqslant \left(\mathrm{DU}(\texttt{Sbox}) \cdot 2^{-8}\right)^{\mathrm{wt}_{\mathbb{K}}\left(\boldsymbol{b}^{(1)}\right) + \mathrm{wt}_{\mathbb{K}}\left(\boldsymbol{a}^{(1)}\right)}. \end{aligned}$$

The quantity

$$W \stackrel{\text{def}}{=} \mathrm{wt}_{\mathbb{K}}\left(\boldsymbol{b}^{(1)}\right) + \mathrm{wt}_{\mathbb{K}}\left(\boldsymbol{a}^{(1)}\right) = \sum_{j:\ \mathrm{wt}_{\mathbb{K}}\left(\boldsymbol{b}^{(1)}_{*,j}\right) \geqslant 1} \mathrm{wt}_{\mathbb{K}}\left(\boldsymbol{b}^{(1)}_{*,j}\right) + \mathrm{wt}_{\mathbb{K}}\left(\boldsymbol{M} \cdot \boldsymbol{b}^{(1)}_{*,j}\right)$$

is then the number of active Sboxes of this 2-round trail. In fact, since we want to avoid the *existence* of a good probability characteristic, the `MixColumns` matrix $\boldsymbol{M}$ of the AES is specifically chosen to minimize the maximum over $\boldsymbol{x}$ of the quantity

$$\left(\mathrm{DU}(\texttt{Sbox}) \cdot 2^{-8}\right)^{\mathrm{wt}_{\mathbb{K}}(\boldsymbol{x}) + \mathrm{wt}_{\mathbb{K}}(\boldsymbol{M} \cdot \boldsymbol{x})}$$

as much as possible with a maximum differential branch number. As a square matrix of size 4, the AES `MixColumns` matrix $\boldsymbol{M}$ reaches the maximum possible branch number of 5.

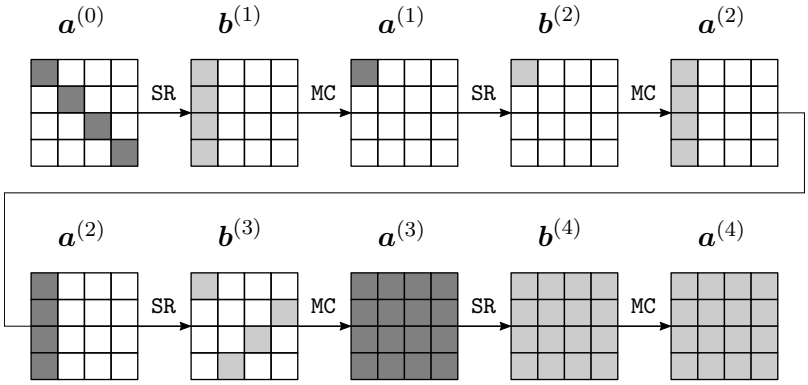

Figure 2.2 – 4-round trail with 25 active Sboxes with $\mathrm{supp}_{\mathbb{K}}\left(\boldsymbol{b}^{(2)}\right) = \{(0,0)\}$.

Characteristics in $\mathrm{AES}^{(4)}$**.**

Let $\left(\boldsymbol{a}^{(0)}, \boldsymbol{a}^{(1)}, \boldsymbol{a}^{(2)}, \boldsymbol{a}^{(3)}, \boldsymbol{a}^{(4)}\right) \in \mathcal{M}_4\left(\mathbb{K}\right)^5$, $\boldsymbol{b}^{(i)} = \texttt{MixColumns}^{-1}\left(\boldsymbol{a}^{(i)}\right)$ and

$$W = \sum_{j:\ \mathrm{wt}_{\mathbb{K}}\left(\boldsymbol{b}^{(1)}_{*,j}\right) \geqslant 1} \mathrm{wt}_{\mathbb{K}}\left(\boldsymbol{b}^{(1)}_{*,j}\right) + \mathrm{wt}_{\mathbb{K}}\left(\boldsymbol{M} \cdot \boldsymbol{b}^{(1)}_{*,j}\right) \quad + \sum_{j:\ \mathrm{wt}_{\mathbb{K}}\left(\boldsymbol{b}^{(3)}_{*,j}\right) \geqslant 1} \mathrm{wt}_{\mathbb{K}}\left(\boldsymbol{b}^{(3)}_{*,j}\right) + \mathrm{wt}_{\mathbb{K}}\left(\boldsymbol{M} \cdot \boldsymbol{b}^{(3)}_{*,j}\right).$$

Then with the case of $\mathrm{AES}^{(2)}$ we have that

$$p\left(\boldsymbol{a}^{(0)}, \boldsymbol{a}^{(1)}, \boldsymbol{a}^{(2)}, \boldsymbol{a}^{(3)}, \boldsymbol{a}^{(4)}\right) = p\left(\boldsymbol{a}^{(0)}, \boldsymbol{a}^{(1)}, \boldsymbol{a}^{(2)}\right) \cdot p\left(\boldsymbol{a}^{(2)}, \boldsymbol{a}^{(3)}, \boldsymbol{a}^{(4)}\right) \leqslant \left(\mathrm{DU}\left(\texttt{Sbox}\right) \cdot 2^{-8}\right)^{W}.$$

W is then the number of active Sboxes for this 4-round trail. Since $\boldsymbol{M}$ has branch number 5, we have that

$$W \geqslant 5 \cdot \left|\left\{ j \;\middle|\; \mathrm{wt}_{\mathbb{K}}\left(\boldsymbol{b}^{(1)}_{*,j}\right) \geqslant 1 \right\}\right| + 5 \cdot \left|\left\{ j \;\middle|\; \mathrm{wt}_{\mathbb{K}}\left(\boldsymbol{b}^{(3)}_{*,j}\right) \geqslant 1 \right\}\right|.$$

Moreover, since the number of active columns is not changed by the `MixColumns` operation, we have as much active columns in $\boldsymbol{b}^{(1)}$ as in $\boldsymbol{a}^{(1)}$. Let j_0 be the index of an active column of $\boldsymbol{a}^{(2)}$. Then with the branch number of $\boldsymbol{M}$, $\mathrm{wt}_{\mathbb{K}}\left(\boldsymbol{a}^{(2)}_{*,j_0}\right) + \mathrm{wt}_{\mathbb{K}}\left(\boldsymbol{b}^{(2)}_{*,j_0}\right) \geqslant 5$. Now the `ShiftRows` operation sends all the active bytes of $\boldsymbol{a}^{(2)}_{*,j_0}$ to different columns of $\boldsymbol{b}^{(3)}$. With $\texttt{ShiftRows}^{-1}$, there is the same phenomenon from $\boldsymbol{b}^{(2)}_{*,j_0}$ to $\boldsymbol{a}^{(1)}$. Finally, $W \geqslant 25$ which is the square of the branch number of $\boldsymbol{M}$. Note that this minimum for the number of active Sboxes can be reached by following the above proof, as shown in Figure 2.2. In this figure, the support of a difference in the state matrix is represented by gray squares and the number of active Sboxes in the trail can be counted on the number of gray squares in $\boldsymbol{a}^{(i)}$ for $i \in [0,3]$.

To summarize, we have the following proposition.

Proposition 2.3. *For all* $\left(\boldsymbol{a}^{(0)}, \ldots, \boldsymbol{a}^{(4)}\right) \in \mathcal{M}_4\left(\mathbb{K}\right)^5$ *with* $\boldsymbol{a}^{(i)} \neq 0$,

$$p\left(\boldsymbol{a}^{(0)}, \ldots, \boldsymbol{a}^{(4)}\right) \leqslant 2^{-150}$$

when $\mathrm{AES}^{(4)}$ *is assumed to behave as a Markov cipher.*

Proof. $\mathrm{DU}\left(\texttt{Sbox}\right) = 4$ and $p\left(\boldsymbol{a}^{(0)}, \ldots, \boldsymbol{a}^{(4)}\right) \leqslant \left(\mathrm{DU}\left(\texttt{Sbox}\right) \cdot 2^{-8}\right)^{-25}$. □

Table 2.2 – [Mou+11, Table 4] Minimum number W of active `Sbox`es for r rounds of AES.

r	1	2	3	4	5	6	7	8	9	10	11	12	13	14
W	1	5	9	25	26	30	34	50	51	55	59	75	76	80

2.3.3 Minimizing the number of active Sboxes

As we saw in the proof of Proposition 2.3, the differential uniformity of the `Sbox` (i.e. the maximum non-trivial value of its DDT) and the minimum number of active `Sbox`es are the two main ingredients for estimating the probability of a characteristic. In fact, the main benefit of the wide-trail design strategy is to build a round function for which the minimum number of active `Sbox`es can be explicitly computed. However, for more than 4 rounds, it is not clear whether the minimum number of active `Sbox`es derived from the minimums over $1, 2, 3$ or 4 rounds is tight or whether they just provide lower bounds. Mouha, Wang, Gu and Preneel proposed in [Mou+11] to explicitly compute this minimum number of active `Sbox`es for any number of rounds by modeling `ShiftRows` and the branch number of `MixColumns` as Mixed-Integer Linear Programming (MILP) constraints. Then, they could use a MILP solver to compute the exact minimum number of active `Sbox`es very easily. We give their results for completeness in Table 2.2. Interestingly, for an even number of rounds $r = 4r_4 + 2r_2 \leqslant 14$, the branch number bounds for 2 and 4 rounds are tight: $W = 25r_4 + 5r_2$.

This success of Mouha et al. in [Mou+11] stimulated the use of MILP solvers to get more accurate bounds on the probability of characteristics but also for many other uses in cryptanalysis. In particular, Sun et al. proposed in [Sun+14a] MILP-based methods to compute accurately the minimum number of active `Sbox`es by taking the internals of DDTs of 4-bit `Sbox`es into account. The designers of the `SKINNY` block cipher [Bei+16] used these approaches to provide lower bounds on the number of active `Sbox`es in their design. Abdelkhalek et al. went further in [Abd+17] by extending the work of Sun et al. in two directions.

- They proposed the first efficient method to model the DDT of an 8-bit `Sbox`.
- By splitting the DDT with respect to its different values, they proposed a method to bound the probability of characteristics directly and without the differential uniformity.

These works (among others) demonstrated how useful modeling cryptographic problems into MILP problems could be in studying symmetric-key primitives.

In Chapter 4, we aim at making the use of MILP solvers even more efficient and practical for cryptographers with an emphasis on differential and impossible differential cryptanalysis.

2.4 Impossible differential cryptanalysis

We saw in Section 2.1 that differential cryptanalysis allowed to mount distinguishers on the basis of a good probability differential holding for a non-negligeable fraction of the keys. Impossible differential cryptanalysis, on the contrary, uses the distinguishing property that a differential $(\boldsymbol{a}, \boldsymbol{b})$ has probability *zero* for all keys. More precisely, if $p\left(B_{\boldsymbol{k}}, \boldsymbol{a} \to \boldsymbol{b}\right) = 0$ for all $\boldsymbol{k}$, then $(\boldsymbol{a}, \boldsymbol{b})$ is called an *impossible* differential. Since for a random function F,

$$\lim_{n \to +\infty} \mathbb{P}\left(p\left(F, \boldsymbol{a} \to \boldsymbol{b}\right) = 0\right) = \exp\left(-1/2\right) \geqslant 0.6,$$

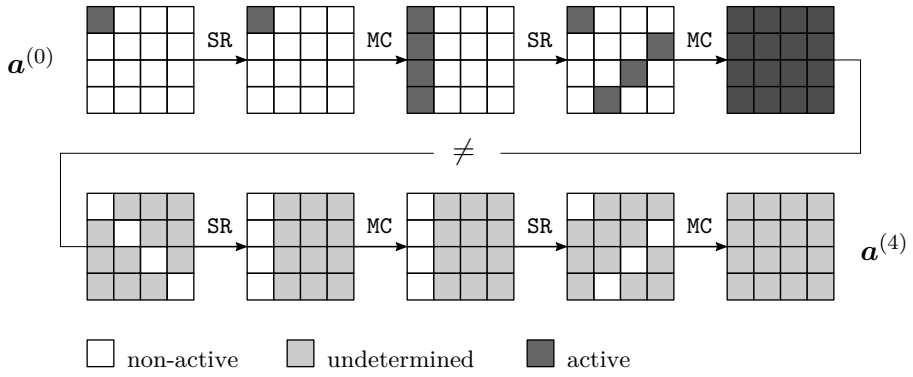

Figure 2.3 – 4-round impossible truncated differential in the AES.

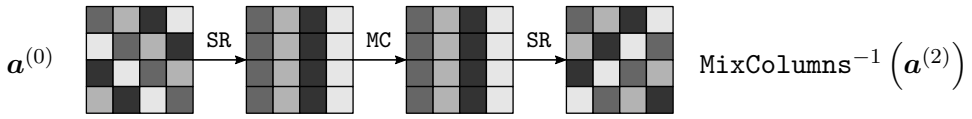

Figure 2.4 – 2-round truncated differential trails. Cells with the same color have to be non-active together.

impossible differentials usually offer a better distinguishing property when they are clustered.

Typically for SPN ciphers, as we saw in Figure 2.2, differential characteristics are naturally clustered around the active `Sboxes`; in the sense that, over a few rounds, two differentials $(\boldsymbol{a}, \boldsymbol{b})$ and $(\boldsymbol{a}', \boldsymbol{b}')$ such that $\mathrm{supp}_{\mathbb{K}}(\boldsymbol{a}) = \mathrm{supp}_{\mathbb{K}}(\boldsymbol{a}')$ and $\mathrm{supp}_{\mathbb{K}}(\boldsymbol{b}) = \mathrm{supp}_{\mathbb{K}}(\boldsymbol{b}')$ will be composed of very similar trails.

Knudsen introduced the study of clustered differentials, called truncated differential cryptanalysis, in [Knu95] and performed the first impossible differential attack on his AES candidate `DEAL` in [Knu98]. Impossible differential cryptanalysis was soon formalized by Biham, Biryukov and Shamir in [BBS99]. Biham et al. notably introduced the *miss-in-the-middle* technique to find impossible differentials. On the example of the AES, the 4-round truncated differentials with exactly one active byte in the input difference and at least one inactive column when applying the last `SubBytes` operation are impossible, as shown in Figure 2.3. This is a miss-in-the-middle: in the middle of the 4 rounds, we have a contradiction between necessarily active bytes because of the input and necessarily non-active bytes because of the output. Bahrak and Aref were the first ones to apply impossible differential cryptanalysis to the AES in [BA08] and improvements came with the works of Lu et al. [Lu+08], Mala et al. [Mal+10] and Boura et al. [Bou+18] for example. All these attacks were based on the same kind of 4-round impossible truncated differential like the one of Figure 2.3.

In 2016, Grassi, Rechberger and Rønjom rephrased the classical 2-round truncated differential trails of Figure 2.4 as *subspace trails* [GRR16]. They also exhibited that using these subspace trails twice gives several 4-round impossible differential distinguishers. The one of Figure 2.3 is one of them. Their new formulation led them to the best chosen plaintext distinguisher on 5-round AES [GRR17] and to the best key-recovery attack on 5-round AES [Gra18]. We present and generalize these attacks to any SPN in Chapter 3.

Chapter 3

Subspace-trail cryptanalysis

Contributions brought forward in this chapter were published in ToSC 2019, Issue 1, and are a joint work with Christina Boura and Anne Canteaut. [BCC19].

3.1 Introduction

Originally designed to withstand against differential and linear statistical attacks and basic algebraic attacks, the AES has inspired the developpement and the improvement of many cryptanalysis techniques like integral attacks [Fer+01; GM00], meet-in-the-middle attacks [DS08] and impossible differential attacks [BA08]. These attacks traditionally aim at performing a key recovery on r_{kr} rounds thanks to a distinguisher on r_{dis} rounds with $r_{\text{dis}} < r_{\text{kr}}$. For all those attacks, the distinguisher affects at most 4 AES rounds — $r_{\text{dis}} \leqslant 4$ — and the key recovery is more efficient than the exhaustive search of the 128, 192 or 256-bit key for respectively 7, 8 and 9 rounds [Jea13].

In the years 2016-2018, distinguishers on 5 AES rounds were found [Sun+16a; GRR17; RBH17; Gra18], revealing unknown properties of this reduced version of the AES. In particular, Grassi, Rechberger and Rønjom demonstrated at Eurocrypt 2017 [GRR17] that there exist two specific linear subspaces V and W of $\mathbb{F}_2^{128}$ such that for any coset of V, the number of pairs of elements $\boldsymbol{x}, \boldsymbol{x}'$, $\boldsymbol{x} \neq \boldsymbol{x}'$ in this coset satisfying

$$\text{AES}^{(5)}(\boldsymbol{x}) + \text{AES}^{(5)}(\boldsymbol{x}') \in W$$

is always a multiple of 8, where $\text{AES}^{(5)}$ is the AES block cipher reduced to 5 rounds, including the whitening key. They named this distinguishing behaviour the *multiple-of-8* property. Unfortunatly, the structure of V and W did not allow to extend this 5-round distinguisher to a key recovery on more rounds, as with classical integral or impossible differential distinguishers. To overcome this issue, Grassi derived new 4-round distinguishers from the proof of the *multiple-of-8* property that he could turn into a key recovery [Gra18]. He called them *mixture-differential* distinguishers. His key recovery was soon improved by Baron, Dunkelman, Keller, Ronen and Shamir to give the best key recovery on $\text{AES}^{(5)}$ [Bar+18].

The proof of the multiple-of-8 property given in [GRR17] suffers from two major drawbacks. First, it is divided in many cases proved separately despite being very close. Second, although Grassi et al. could adapt their proof to other subspaces, this adaptation came at the cost of another yet very similar proof. Third, these redundant proofs were very specific to the AES construction and could not allow to discriminate the properties of its components which are essential to the proofs.

Consequently, the question whether other SPN block ciphers could be vulnerable to the multiple-of-8 property remained open and the same issues could be raised for mixture-differential distinguishers.

The goal of this chapter is to present a general formulation and alternative proofs of both properties for SPN block ciphers. This new formulation allows in particular to study their resistance against multiple-of and mixture-differential distinguishers in a systematic way.

Our contributions. The multiple-of-8 property of [GRR17] and the mixture-differential distinguishers of [Gra18] are the consequences of two facts.

- There exist 2-round *subspace trails* for the AES. This was already shown by Grassi, Rechberger and Rønjom in 2016 [GRR16].
- For at least one subspace of state elements W at the end of a subspace trail and any of its coset, the map from pairs of vectors from this coset to the difference between their images through the AES round function

$$\left\{\boldsymbol{p}^0, \boldsymbol{p}^1\right\} \mapsto R\left(\boldsymbol{p}^0\right) + R\left(\boldsymbol{p}^1\right)$$

  is invariant under an equivalence relation between pairs. This formulation is new and comes from the work presented here.

We provide a simple and compact proof for the second fact and for both the multiple-of-8 property and the mixture differential distinguishers. Our proofs clarify which properties of the AES round function enable these attacks. In order to generalize these attacks to other SPN ciphers, we also explicit the form of linear subspaces W for which the second fact holds. Combining this generalization with an algorithm by Leander, Tezcan and Wiemer dedicated to the search for subspace trails in SPN ciphers [LTW18], we show how to find the same kind of property in any SPN cipher with the examples of `Midori` [Ban+15], `KLEIN` [GNL12], `LED` [Guo+11] and `SKINNY` [Bei+16].

Organization of the chapter. Basic definitions on subspace trails are recalled in Section 3.2. Section 3.3 describes the multiple-of-8 property from [GRR17] and the mixture-differential distinguishers from [Gra18]. In Section 3.4, we exhibit an equivalence relation between pairs of AES states, which leads to a new proof of the above properties and we discuss the influence of the branch number of the `MixColumns` matrix on these results. Then, an adaptation to any SPN and other linear subspaces is presented in Section 3.5. Finally, applications on different ciphers are provided in Section 3.6.

3.2 Subspace trails in the AES

We recall in this section the basic notation on subspace-trail cryptanalysis of AES, first introduced in [GRR16] and we will reuse the notation introduced in Section 1.4.3.

Let d be the degree of the extension over $\mathbb{F}_2$ on which the `Sbox` operates and $\mathbb{K} = \mathbb{F}_{2^d}$. As we saw in Section 1.4.3, $d = 8$ for the AES but since the results do not depend on the `Sbox` or the `MixColumns` matrix, we can choose an arbitrary d. Moreover, simulations are often performed on small-scale variants of the AES with a 4-bit S-box (see e.g. Section 5.2 in [Gra18]). Let $(\boldsymbol{e}_{i,j})_{i,j\in[0,\dots,3]}$ be the canonical basis of $\mathcal{M}_4(\mathbb{K})$. In the following, $\mathrm{vect}_{\mathbb{K}}(\boldsymbol{v}_0, \dots, \boldsymbol{v}_{k-1})$ denotes the linear space formed by all linear combinations with coefficients in $\mathbb{K}$ of the vectors $\boldsymbol{v}_0, \dots, \boldsymbol{v}_{k-1} \in \mathcal{M}_4(\mathbb{K})$. As

in [GRR16; GRR17], we define the following subspaces of $\mathcal{M}_4(\mathbb{K})$ for $i \in [0, \ldots, 3]$, with indices computed modulo 4.

- The column spaces: $\mathcal{C}_i = \mathrm{vect}_{\mathbb{K}}(\boldsymbol{e}_{0,i}, \boldsymbol{e}_{1,i}, \boldsymbol{e}_{2,i}, \boldsymbol{e}_{3,i})$,
- The diagonal spaces: $\mathcal{D}_i = \mathrm{vect}_{\mathbb{K}}(\boldsymbol{e}_{0,i}, \boldsymbol{e}_{1,i+1}, \boldsymbol{e}_{2,i+2}, \boldsymbol{e}_{3,i+3}) = \texttt{ShiftRows}^{-1}(\mathcal{C}_i)$,
- The anti-diagonal spaces: $\mathcal{ID}_i = \mathrm{vect}_{\mathbb{K}}(\boldsymbol{e}_{0,i}, \boldsymbol{e}_{1,i-1}, \boldsymbol{e}_{2,i-2}, \boldsymbol{e}_{3,i-3}) = \texttt{ShiftRows}(\mathcal{C}_i)$,
- The mixed spaces: $\mathcal{M}_i = \texttt{MixColumns}(\mathcal{ID}_i)$.

For example, if $x_0, x_1, x_2, x_3 \in \mathbb{K}$,

$$\begin{pmatrix} x_0 & 0 & 0 & 0 \\ x_1 & 0 & 0 & 0 \\ x_2 & 0 & 0 & 0 \\ x_3 & 0 & 0 & 0 \end{pmatrix} \in \mathcal{C}_0, \qquad \begin{pmatrix} x_0 & 0 & 0 & 0 \\ 0 & x_1 & 0 & 0 \\ 0 & 0 & x_2 & 0 \\ 0 & 0 & 0 & x_3 \end{pmatrix} \in \mathcal{D}_0,$$

$$\begin{pmatrix} 2 \cdot x_0 & x_1 & x_2 & 3 \cdot x_3 \\ x_0 & x_1 & 3 \cdot x_2 & 2 \cdot x_3 \\ x_0 & 3 \cdot x_1 & 2 \cdot x_2 & x_3 \\ 3 \cdot x_0 & 2 \cdot x_1 & x_2 & x_3 \end{pmatrix} \in \mathcal{M}_0, \qquad \begin{pmatrix} x_0 & 0 & 0 & 0 \\ 0 & 0 & 0 & x_1 \\ 0 & 0 & x_2 & 0 \\ 0 & x_3 & 0 & 0 \end{pmatrix} \in \mathcal{ID}_0.$$

For $I \subseteq \{0, 1, 2, 3\}$, we also define:

$$\mathcal{C}_I = \sum_{i \in I} \mathcal{C}_i, \quad \mathcal{D}_I = \sum_{i \in I} \mathcal{D}_i, \quad \mathcal{ID}_I = \sum_{i \in I} \mathcal{ID}_i, \quad \mathcal{M}_I = \sum_{i \in I} \mathcal{M}_i.$$

Now that we have linear subspaces of $\mathcal{M}_4(\mathbb{K})$, we define their *cosets* as affine subspaces of $\mathcal{M}_4(\mathbb{K})$. More precisely, a coset of the linear subspace $V \subseteq \mathcal{M}_4(\mathbb{K})$ is a set of the form $V + \boldsymbol{a} = \{ \boldsymbol{v} + \boldsymbol{a} \mid \boldsymbol{v} \in V \}$ where $\boldsymbol{a} \in \mathcal{M}_4(\mathbb{K})$. Moreover, we are going to pay attention to subspaces that satisfy a specific property defined below.

Definition 3.1 (Subspace trail [GRR16])**.** Let $F : \mathbb{K}^N \to \mathbb{K}^N$ be any map. Two linear subspaces $U, V \subseteq \mathbb{K}^N$ form an F-subspace trail if

$$\forall \boldsymbol{a} \in \mathbb{K}^N, \exists \boldsymbol{b} \in \mathbb{K}^N : F(U + \boldsymbol{a}) \subseteq V + \boldsymbol{b}, \tag{3.1}$$

which is denoted by $U \overset{F}{\rightrightarrows} V$. The negation is denoted by $U \overset{F}{\not\rightrightarrows} V$. An $(r+1)$-tuple of subspaces $\left(U^{(0)}, \ldots, U^{(r)}\right)$ is called a subspace trail over r rounds $\left(F^{(1)}, \ldots, F^{(r)}\right)$ if

$$\forall i \in [1, \ldots, r],\ U^{(i-1)} \overset{F^{(i)}}{\rightrightarrows} U^{(i)}.$$

For example, we have the trivial subspace trails $\{0\} \overset{F}{\rightrightarrows} \{0\}$ and $U \overset{F}{\rightrightarrows} \mathbb{K}^N$. In this chapter, we only consider *exact* subspace trails, i.e., for which equality holds in (3.1). If F is a bijection, this means that $\dim U = \dim V$.

3.3 Distinguishers based on subspace trails

We describe in this section three distinguishers based on subspace trail cryptanalysis in order to have a complete understanding of the context in which our contribution lies. The first one, describes the multiple-of-8 property presented by Grassi, Rechberger and Rønjom at Eurocrypt 2017 [GRR17], while the other two are based on the mixture differential property and were published by Grassi in [Gra18].

3.3.1 The multiple-of-8 property

We first describe the distinguisher presented in [GRR17]. We begin with this easy-to-verify lemma describing a two-round subspace trail for the AES and illustrated in Figure 2.4.

Lemma 3.1 ([GRR16]). *Let $I \subseteq \{0,1,2,3\}$, then $\mathcal{D}_I \overset{R}{\rightrightarrows} \mathcal{C}_I \overset{R}{\rightrightarrows} \mathcal{M}_I$.*

Now comes a more subtle lemma which is the keystone of Theorem 3.1. In the whole chapter, as in [GRR17], we always consider *unordered pairs* of elements and denote them as pair sets, i.e. $\{\boldsymbol{a}, \boldsymbol{b}\}$.

Lemma 3.2 ([GRR17]). *Let $\boldsymbol{a} \in \mathcal{M}_4(\mathbb{K})$, $i \in \{0,1,2,3\}$, $J \subseteq \{0,1,2,3\}$. We define*

$$n = \left| \left\{ \{\boldsymbol{p}^0, \boldsymbol{p}^1\} \text{ with } \boldsymbol{p}^0, \boldsymbol{p}^1 \in \mathcal{M}_i + \boldsymbol{a} \;\middle|\; R\left(\boldsymbol{p}^0\right) + R\left(\boldsymbol{p}^1\right) \in \mathcal{D}_J \right\} \right|.$$

Then $n \equiv 0 \bmod 8$.

A proof of Lemma 3.2 is given in Section 6 of [GRR17] but we provide in Section 3.4 a much more compact proof of this same result. A direct consequence of Lemmas 3.1 and 3.2 is the following theorem.

Theorem 3.1 ([GRR17]). *Let $\boldsymbol{a} \in \mathcal{M}_4(\mathbb{K})$, $i \in \{0,1,2,3\}$, $J \subseteq \{0,1,2,3\}$. We define*

$$n = \left| \left\{ \{\boldsymbol{p}^0, \boldsymbol{p}^1\} \text{ with } \boldsymbol{p}^0, \boldsymbol{p}^1 \in \mathcal{M}_i + \boldsymbol{a} \;\middle|\; \mathrm{AES}^{(5)}\left(\boldsymbol{p}^0\right) + \mathrm{AES}^{(5)}\left(\boldsymbol{p}^1\right) \in \mathcal{M}_J \right\} \right|.$$

Then $n \equiv 0 \bmod 8$.

Theorem 3.1 directly provides a distinguisher for five rounds of the AES independent of the secret key. Indeed, given an oracle simulating either five rounds of the AES, either a random permutation, one can compute the number n from Theorem 3.1 with only the 2^{32} plaintexts belonging to the same coset of $\mathcal{D}_i$. This distinguisher is fully described in [GRR17].

Exploiting the above distinguisher for mounting a key-recovery attack on more rounds revealed to be however a difficult task because of the form of the output subspace $\mathcal{M}_J$. Indeed, as $\mathcal{M}_J$ affects the whole AES-state, a key-recovery attack requires the guess of the entire subkey in the last round. For this reason, Grassi presented in [Gra18] new distinguishers that exploit similar properties but have a description that is more adapted to a key-recovery attack. The counterpart is that these distinguishers cover one round less than in [GRR17].

3.3.2 Mixture-differential distinguishers

In the following, we use the notation from [Gra18]: for a given basis $(\boldsymbol{g}_0, \ldots, \boldsymbol{g}_{k-1})$ and a given element $\boldsymbol{a} \in \mathcal{M}_4(\mathbb{K})$, a vector $\boldsymbol{p}$ in the affine subspace $\boldsymbol{a} + \mathrm{vect}_{\mathbb{K}}(\boldsymbol{g}_0, \ldots, \boldsymbol{g}_{k-1})$ defined by $\boldsymbol{p} = \boldsymbol{a} + \sum_{i=0}^{k-1} x_i \boldsymbol{g}_i$ with $x_i \in \mathbb{K}$ is denoted by $\boldsymbol{p} \equiv (x_0, \ldots, x_{k-1})$.

Theorem 3.2 ([Gra18]). *Let $\boldsymbol{a} \in \mathcal{M}_4(\mathbb{K})$, $i \in \{0,1,2,3\}, J \subseteq \{0,1,2,3\}$ and let $\boldsymbol{p}^0, \boldsymbol{p}^1, \boldsymbol{q}^0, \boldsymbol{q}^1 \in \mathrm{vect}_{\mathbb{K}}(\boldsymbol{e}_{0,i}, \boldsymbol{e}_{1,i}) + \boldsymbol{a}$. Suppose that*

$$\boldsymbol{p}^0 \equiv (x_0, x_1), \qquad \boldsymbol{p}^1 \equiv (y_0, y_1), \qquad \boldsymbol{q}^0 \equiv (x_0, y_1), \qquad \boldsymbol{q}^1 \equiv (y_0, x_1).$$

Then

$$\mathrm{AES}^{(4)}\left(\boldsymbol{p}^0\right) + \mathrm{AES}^{(4)}\left(\boldsymbol{p}^1\right) \in \mathcal{M}_J \iff \mathrm{AES}^{(4)}\left(\boldsymbol{q}^0\right) + \mathrm{AES}^{(4)}\left(\boldsymbol{q}^1\right) \in \mathcal{M}_J.$$

We will provide an alternative proof of Theorem 3.2 in Section 3.5. The following variant of the distinguisher, involving another input subspace, is also exhibited in [Gra18].

Theorem 3.3 ([Gra18]). *Let $\boldsymbol{a} \in \mathcal{M}_4(\mathbb{K})$, $i \in \{0,1,2,3\}$, $J \subseteq \{0,1,2,3\}$ and let $\boldsymbol{p}^0, \boldsymbol{p}^1 \in \mathcal{C}_i + \boldsymbol{a}$. Suppose that $\boldsymbol{p}^0 \equiv (x_0, x_1, x_2, x_3)$ and that $\boldsymbol{p}^1 \equiv (y_0, y_1, y_2, y_3)$. Then,*

$$\mathrm{AES}^{(4)}\left(\boldsymbol{p}^0\right) + \mathrm{AES}^{(4)}\left(\boldsymbol{p}^1\right) \in \mathcal{M}_J \iff \mathrm{AES}^{(4)}\left(\boldsymbol{q}^0\right) + \mathrm{AES}^{(4)}\left(\boldsymbol{q}^1\right) \in \mathcal{M}_J,$$

for all sets of plaintexts $\{\boldsymbol{q}^0, \boldsymbol{q}^1\}$ with $\boldsymbol{q}^0, \boldsymbol{q}^1 \in \mathcal{C}_i + \boldsymbol{a}$ of the following form:

1. $\boldsymbol{q}^0 \equiv (y_0, x_1, x_2, x_3), \boldsymbol{q}^1 \equiv (x_0, y_1, y_2, y_3)$,
2. $\boldsymbol{q}^0 \equiv (x_0, y_1, x_2, x_3), \boldsymbol{q}^1 \equiv (y_0, x_1, y_2, y_3)$,
3. $\boldsymbol{q}^0 \equiv (x_0, x_1, y_2, x_3), \boldsymbol{q}^1 \equiv (y_0, y_1, x_2, y_3)$,
4. $\boldsymbol{q}^0 \equiv (x_0, x_1, x_2, y_3), \boldsymbol{q}^1 \equiv (y_0, y_1, y_2, x_3)$,
5. $\boldsymbol{q}^0 \equiv (x_0, x_1, y_2, y_3), \boldsymbol{q}^1 \equiv (y_0, y_1, x_2, x_3)$,
6. $\boldsymbol{q}^0 \equiv (x_0, y_1, x_2, y_3), \boldsymbol{q}^1 \equiv (y_0, x_1, y_2, x_3)$,
7. $\boldsymbol{q}^0 \equiv (x_0, y_1, y_2, x_3), \boldsymbol{q}^1 \equiv (y_0, x_1, x_2, y_3)$,

We will provide a proof of Theorem 3.3 in Section 3.4.

3.4 A more concise and general proof

This section is dedicated to our proof of Lemma 2 from [GRR17] (Lemma 3.2 here). This new proof is a much more concise version of the case-by-case proof given in the original paper. To be more precise, instead of proving Lemma 2, we prove directly a more general variant. This generalisation is present in the original paper but its proof is only sketched in Appendix A of [GRR17]. Indeed, the proof framework of Lemma 2 in [GRR17] does not allow a compact proof of this generalisation.

Our approach for proving Lemma 3.2 can be divided into three steps:

- there exists an equivalence relation between pairs of elements in a certain subspace of $\mathcal{M}_4(\mathbb{K})$ (Definition 3.3);
- some function on those pairs derived from the round function is invariant under this equivalence relation (Theorem 3.4);
- the cardinality of the equivalence classes is always a multiple of 8 (Proposition 3.1).

In the following, we fix $\boldsymbol{a} \in \mathcal{M}_4(\mathbb{K})$, $I \subseteq \{0,1,2,3\}$ and $J \subseteq \{0,1,2,3\}$. Here it might help to remind that

$$\mathcal{ID}_I = \mathrm{vect}_{\mathbb{K}}\left(\boldsymbol{e}_{k,i-k} \mid k \in \{0,1,2,3\}, i \in I\right) = \mathrm{vect}_{\mathbb{K}}\left(\boldsymbol{e}_{i-k,k} \mid k \in \{0,1,2,3\}, i \in I\right),$$
$$\mathcal{M}_I = \texttt{MixColumns}(\mathcal{ID}_I) = \mathrm{vect}_{\mathbb{K}}\left(\texttt{MixColumns}(\boldsymbol{e}_{i-k,k}) \mid k \in \{0,1,2,3\}, i \in I\right).$$

3.4.1 An equivalence relation between pairs of states

Definition 3.2 (Information set)**.** Let $\{\boldsymbol{p}^0, \boldsymbol{p}^1\}$ be a set of elements in $\mathcal{M}_I + \boldsymbol{a}$, written as

$$\boldsymbol{p}^0 = \sum_{k=0}^{3}\sum_{i\in I} p^0_{i,k} \cdot \texttt{MixColumns}(\boldsymbol{e}_{i-k,k}) + \boldsymbol{a} \quad \text{and} \quad \boldsymbol{p}^1 = \sum_{k=0}^{3}\sum_{i\in I} p^1_{i,k} \cdot \texttt{MixColumns}(\boldsymbol{e}_{i-k,k}) + \boldsymbol{a}$$

for some (uniquely defined) $\boldsymbol{p}^0_{i,k}, \boldsymbol{p}^1_{i,k} \in \mathbb{K}$, $i \in I, 0 \leqslant k \leqslant 3$. The *information set* K of $\{\boldsymbol{p}^0, \boldsymbol{p}^1\}$ is defined as

$$K = \left\{ k \in \{0,1,2,3\} \;\middle|\; \exists i \in I : p^0_{i,k} \neq p^1_{i,k} \right\}.$$

Definition 3.3 (Equivalence relation)**.** Let $P = \{\boldsymbol{p}^0, \boldsymbol{p}^1\}$ and $Q = \{\boldsymbol{q}^0, \boldsymbol{q}^1\}$ with $\boldsymbol{p}^0$, $\boldsymbol{p}^1$, $\boldsymbol{q}^0$, $\boldsymbol{q}^1$ $\in \mathcal{M}_I + \boldsymbol{a}$. We say that $P \sim Q$ if:

- $\{\boldsymbol{p}^0, \boldsymbol{p}^1\}$ and $\{\boldsymbol{q}^0, \boldsymbol{q}^1\}$ have the same information set K.
- $\forall k \in K, \exists b \in \{0,1\} : \forall i \in I, q^0_{i,k} = p^b_{i,k}$ and $q^1_{i,k} = p^{1-b}_{i,k}$.

Clearly, $\sim$ is an equivalence relation on unordered pairs of $\mathcal{M}_I + \boldsymbol{a}$.

Example 3.1. The following two sets $\{\boldsymbol{p}^0, \boldsymbol{p}^1\}$ and $\{\boldsymbol{q}^0, \boldsymbol{q}^1\}$, with $\boldsymbol{p}^0$, $\boldsymbol{p}^1$, $\boldsymbol{q}^0$, $\boldsymbol{q}^1 \in \mathcal{M}_0$ are equivalent and their information set K has cardinality $|K| = 2$.

$$\left\{\boldsymbol{p}^0, \boldsymbol{p}^1\right\} = \left\{ \begin{pmatrix} 2\cdot x_0 & x_1 & z_2 & 3\cdot z_3 \\ x_0 & x_1 & 3\cdot z_2 & 2\cdot z_3 \\ x_0 & 3\cdot x_1 & 2\cdot z_2 & z_3 \\ 3\cdot x_0 & 2\cdot x_1 & z_2 & z_3 \end{pmatrix}, \begin{pmatrix} 2\cdot y_0 & y_1 & z_2 & 3\cdot z_3 \\ y_0 & y_1 & 3\cdot z_2 & 2\cdot z_3 \\ y_0 & 3\cdot y_1 & 2\cdot z_2 & z_3 \\ 3\cdot y_0 & 2\cdot y_1 & z_2 & z_3 \end{pmatrix} \right\}$$
$$\sim$$
$$\left\{\boldsymbol{q}^0, \boldsymbol{q}^1\right\} = \left\{ \begin{pmatrix} 2\cdot x_0 & y_1 & w_2 & 3\cdot w_3 \\ x_0 & y_1 & 3\cdot w_2 & 2\cdot w_3 \\ x_0 & 3\cdot y_1 & 2\cdot w_2 & w_3 \\ 3\cdot x_0 & 2\cdot y_1 & w_2 & w_3 \end{pmatrix}, \begin{pmatrix} 2\cdot y_0 & x_1 & w_2 & 3\cdot w_3 \\ y_0 & x_1 & 3\cdot w_2 & 2\cdot w_3 \\ y_0 & 3\cdot x_1 & 2\cdot w_2 & w_3 \\ 3\cdot y_0 & 2\cdot x_1 & w_2 & w_3 \end{pmatrix} \right\}$$

Now that we have the right definitions, we can state and prove the following key theorem that has also other applications than just proving Lemma 3.2. It is worth noticing that, in the original proof of Lemma 3.2, the authors split the proof procedure in several cases, each case corresponding to a different size of the information set. In our approach we assemble all cases together by using the above introduced equivalence relation.

Theorem 3.4. *For any $\boldsymbol{a} \in \mathcal{M}_4(\mathbb{K})$, the function Δ operating on unordered pairs of elements in $\mathcal{M}_I + \boldsymbol{a}$ and defined by*

$$\Delta : \quad \{\boldsymbol{p}^0, \boldsymbol{p}^1\} \quad \longmapsto \quad R(\boldsymbol{p}^0) + R(\boldsymbol{p}^1)$$

is constant over the equivalence classes for $\sim$.

Proof. Let $P = \{\boldsymbol{p}^0, \boldsymbol{p}^1\}$ and $Q = \{\boldsymbol{q}^0, \boldsymbol{q}^1\}$ with $\boldsymbol{p}^0, \boldsymbol{p}^1, \boldsymbol{q}^0, \boldsymbol{q}^1 \in \mathcal{M}_I + \boldsymbol{a}$ such that $P \sim Q$. We write as in Definition 3.2

$$\boldsymbol{p}^0 = \sum_{k=0}^{3} \sum_{i \in I} p^0_{i,k} \texttt{MixColumns}(\boldsymbol{e}_{i-k,k}) + \boldsymbol{a} \quad \text{and} \quad \boldsymbol{p}^1 = \sum_{k=0}^{3} \sum_{i \in I} p^1_{i,k} \texttt{MixColumns}(\boldsymbol{e}_{i-k,k}) + \boldsymbol{a}.$$

We also write the `MixColumns` matrix $M_{\texttt{MixColumns}} = (m_{\ell,k})_{0 \leqslant \ell,k \leqslant 3}$. Hence

$$\boldsymbol{p}^0 = \sum_{k,\ell} \sum_{i \in I} p^0_{i,k} m_{\ell,i-k} \boldsymbol{e}_{\ell,k} + \boldsymbol{a} = \sum_{k,\ell} \left(\sum_{i \in I} p^0_{i,k} m_{\ell,i-k} + a_{\ell,k} \right) \boldsymbol{e}_{\ell,k}.$$

Then $\texttt{SubBytes}\left(\boldsymbol{p}^0\right) = \sum_{k,\ell} \texttt{Sbox}\left(\sum_{i \in I} p^0_{i,k} m_{\ell,i-k} + a_{\ell,k}\right) \boldsymbol{e}_{\ell,k}$ and

$$\begin{aligned} &\texttt{SubBytes}\left(\boldsymbol{p}^0\right) + \texttt{SubBytes}\left(\boldsymbol{p}^1\right) \\ &\qquad = \sum_{k,\ell} \left[\texttt{Sbox}\left(\sum_{i \in I} p^0_{i,k} m_{\ell,i-k} + a_{\ell,k} \right) + \texttt{Sbox}\left(\sum_{i \in I} p^1_{i,k} m_{\ell,i-k} + a_{\ell,k} \right) \right] \boldsymbol{e}_{\ell,k} \,. \end{aligned} \tag{3.2}$$

It is now clear with Definition 3.3 and Equation (3.2) that $\texttt{SubBytes}\left(\boldsymbol{p}^0\right) + \texttt{SubBytes}\left(\boldsymbol{p}^1\right)$ and $\texttt{SubBytes}\left(\boldsymbol{q}^0\right) + \texttt{SubBytes}\left(\boldsymbol{q}^1\right)$ are equal in $\mathcal{M}_4(\mathbb{K})$. Indeed, with K the information set of P and Q,

$$\begin{aligned} &\texttt{SubBytes}\left(\boldsymbol{q}^0\right) + \texttt{SubBytes}\left(\boldsymbol{q}^1\right) \\ &= \sum_{k,\ell \in \{0,1,2,3\}} \left[\texttt{Sbox}\left(\sum_{i \in I} q^0_{i,k} m_{\ell,i-k} + a_{\ell,k} \right) + \texttt{Sbox}\left(\sum_{i \in I} q^1_{i,k} m_{\ell,i-k} + a_{\ell,k} \right) \right] \boldsymbol{e}_{\ell,k} \\ &= \sum_{\substack{\ell \in \{0,1,2,3\}, \\ k \in K}} \left[\texttt{Sbox}\left(\sum_{i \in I} q^0_{i,k} m_{\ell,i-k} + a_{\ell,k} \right) + \texttt{Sbox}\left(\sum_{i \in I} q^1_{i,k} m_{\ell,i-k} + a_{\ell,k} \right) \right] \boldsymbol{e}_{\ell,k} \\ &= \sum_{\substack{\ell \in \{0,1,2,3\}, \\ k \in K}} \left[\texttt{Sbox}\left(\sum_{i \in I} q^{b(k)}_{i,k} m_{\ell,i-k} + a_{\ell,k} \right) + \texttt{Sbox}\left(\sum_{i \in I} q^{1-b(k)}_{i,k} m_{\ell,i-k} + a_{\ell,k} \right) \right] \boldsymbol{e}_{\ell,k} \\ &= \sum_{\substack{\ell \in \{0,1,2,3\}, \\ k \in K}} \left[\texttt{Sbox}\left(\sum_{i \in I} p^0_{i,k} m_{\ell,i-k} + a_{\ell,k} \right) + \texttt{Sbox}\left(\sum_{i \in I} p^1_{i,k} m_{\ell,i-k} + a_{\ell,k} \right) \right] \boldsymbol{e}_{\ell,k} \\ &= \texttt{SubBytes}\left(\boldsymbol{p}^0\right) + \texttt{SubBytes}\left(\boldsymbol{p}^1\right) \,. \end{aligned}$$

Therefore,

$$\begin{aligned} \Delta(P) &= R\left(\boldsymbol{p}^0\right) + R\left(\boldsymbol{p}^1\right) \\ &= \texttt{MixColumns} \circ \texttt{ShiftRows}\left(\texttt{SubBytes}\left(\boldsymbol{p}^0\right) + \texttt{SubBytes}\left(\boldsymbol{p}^1\right) \right) \\ &= \texttt{MixColumns} \circ \texttt{ShiftRows}\left(\texttt{SubBytes}\left(\boldsymbol{q}^0\right) + \texttt{SubBytes}\left(\boldsymbol{q}^1\right) \right) \\ &= \Delta(Q). \end{aligned}$$

□

We would like to adapt in Section 3.5 the previous theorem to any SPN structure and any linear space. For this, it is important to identify the key argument that makes the proof work. Looking at the proof carefully, it appears that it relies on the fact that the coordinates $p_{i,k}$ of the elements in $\mathcal{M}_I$ can be decomposed into disjoint sets, in such a way that each individual S-box involves only one coordinate subset. Here, the four subsets correspond to the coordinates $p_{i,k}$ sharing the same index k. Indeed, the `Sbox` at Row ℓ and Column k involves all $p_{i,k}, i \in I$. This particular structure then makes possible to exchange between the two pairs $\{\boldsymbol{p}^0, \boldsymbol{p}^1\}$ and $\{\boldsymbol{q}^0, \boldsymbol{q}^1\}$ the coordinates corresponding to one of the subsets. This is exactly the property that we will consider in the generalisation presented in Section 3.5.

3.4.2 Multiple-of-8 equivalence classes

The multiple-of-8 property presented in Lemma 3.2 is then a direct consequence of the previous theorem. It is derived by combining the theorem with the following proposition, which computes the cardinality of the equivalence classes.

Proposition 3.1. *Let $\mathfrak{C}$ be an equivalence class with information set K. The cardinality of $\mathfrak{C}$ is*

$$|\mathfrak{C}| = 2^{|K|-1+d|I|(4-|K|)}.$$

It is always a multiple of 8.

Proof. Since for a given set $\{\boldsymbol{p}^0, \boldsymbol{p}^1\}$ with information set K, we have that $\forall k \notin K, \forall i \in I, p^0_{i,k} = p^1_{i,k}$, we have $(2^d)^{|I|\times(4-|K|)}$ choices for the shared coordinates in a pair of $\mathfrak{C}$. Those coordinates fixed, we have to make for all $k \in K$ the choice $b = 0$ or $b = 1$, i.e. $2^{|K|}$ choices. Since we are counting unordered pairs, we have $2^{|K|-1+d|I|(4-|K|)}$ elements in $\mathfrak{C}$. Obviously, the exponent $|K| - 1 + d|I|(4 - |K|)$ is minimal for $|K| = 4$. We deduce that

$$|K| - 1 + d|I|(4 - |K|) \geqslant 3 \,,$$

leading to $|\mathfrak{C}| \equiv 0 \bmod 8$. □

By combining Proposition 3.1 and Theorem 3.4, we deduce the following corollary which generalises Lemma 3.2 in the sense that we are not restricted to the case $|I| = 1$ as in Lemma 3.2.

Corollary 3.1. *Let*

$$n = \left| \left\{ \left\{ \boldsymbol{p}^0, \boldsymbol{p}^1 \right\} \text{ with } \boldsymbol{p}^0, \boldsymbol{p}^1 \in \mathcal{M}_I + \boldsymbol{a} \;\middle|\; R\left(\boldsymbol{p}^0\right) + R\left(\boldsymbol{p}^1\right) \in \mathcal{D}_J \right\} \right| .$$

Then, $n \equiv 0 \bmod 8$.

Proof. Let $\mathcal{P}^2(\mathcal{M}_I + \boldsymbol{a})$ denote the set of all unordered pairs of elements in $\mathcal{M}_I + \boldsymbol{a}$, and $\mathcal{P}^2(\mathcal{M}_I + \boldsymbol{a})/\sim$ denote the set of all equivalence classes for $\sim$. Since the equivalence classes form a partition of $\mathcal{P}^2(\mathcal{M}_I + \boldsymbol{a})$, we have that

$$n = \left|\Delta^{-1}(\mathcal{D}_J)\right| = \sum_{\mathfrak{C} \in \mathcal{P}^2(\mathcal{M}_I+\boldsymbol{a})/\sim} \left|\Delta^{-1}(\mathcal{D}_J) \cap \mathfrak{C}\right| \,.$$

We know from Theorem 3.4 that Δ is constant on the equivalence classes, implying that $(\Delta^{-1}(\mathcal{D}_J) \cap \mathfrak{C})$ equals either 0 or $|\mathfrak{C}|$. In other words, there exists a function δ from $\mathcal{P}^2(\mathcal{M}_I + \boldsymbol{a})/\sim$ into $\{0,1\}$ such that

$$\left|\Delta^{-1}(\mathcal{D}_J) \cap \mathfrak{C}\right| = \delta(\mathfrak{C}) \times |\mathfrak{C}| \ .$$

It follows that

$$n = \sum_{\mathfrak{C} \in \mathcal{P}^2(\mathcal{M}_I + \boldsymbol{a})/\sim} \delta(\mathfrak{C}) \times |\mathfrak{C}| \equiv 0 \bmod 8,$$

since, by Proposition 3.1, all equivalence classes have a cardinality divisible by 8. □

3.4.3 Influence of the branch number

In [GRR17], the proof of Lemma 2 uses the fact that the differential branch number of `MixColumns` denoted by b is maximal and equal to 5 but as we have seen, the branch number b does not have any importance for this lemma. It affects the exact value of n, but not the fact that it is a multiple of 8.

Indeed, in the formula given in the proof of Corollary 3.1 for computing n, we can distinguish between the different equivalent classes according to the size of their information set:

$$n = \sum_{\mathfrak{C} \in \mathcal{P}^2(\mathcal{M}_I + \boldsymbol{a})/\sim} \delta(\mathfrak{C}) \times |\mathfrak{C}| = \sum_{h=0}^{4} \sum_{\mathfrak{C}:|K(\mathfrak{C})|=h} \delta(\mathfrak{C}) \times |\mathfrak{C}| \ .$$

Obviously, for any $\{\boldsymbol{p}^0, \boldsymbol{p}^1\}$ with information set K,

$$\begin{aligned} \boldsymbol{p}^0 + \boldsymbol{p}^1 &= \sum_{k=0}^{3} \sum_{i \in I} (p^0_{i,k} + p^1_{i,k}) \texttt{MixColumns}(\boldsymbol{e}_{i-k,k}) \\ &= \sum_{k \in K} \sum_{i \in I} (p^0_{i,k} + p^1_{i,k}) \texttt{MixColumns}(\boldsymbol{e}_{i-k,k}) \in \mathcal{C}_K, \end{aligned}$$

implying, by Lemma 3.1, that $(R(\boldsymbol{p}^0) + R(\boldsymbol{p}^1))$ belongs to $\mathcal{M}_K$.

However, it has been proved in [GRR16, Lemma 5] that, for any two sets $I, J \subseteq \{0, \ldots, 3\}$ such that $|I| + |J| < b$ where b is the branch number of `MixColumns`, we have $\mathcal{D}_I \cap \mathcal{M}_J = \{0\}$. It then follows that, if K has size $h < b - |J|$, then $R(\boldsymbol{p}^0) + R(\boldsymbol{p}^1) \notin \mathcal{D}_J$ unless $\boldsymbol{p}^0 = \boldsymbol{p}^1$. We can then express the influence of the branch number on n with the formula

$$n = \sum_{h=b-|J|}^{4} \sum_{\mathfrak{C}:|K(\mathfrak{C})|=h} \delta(\mathfrak{C}) \times |\mathfrak{C}| \ ,$$

which does not affect the multiple-of-8 property.

3.4.4 An alternative proof of Theorem 3.3

The multiple-of-8 property is a consequence of Theorem 3.4, which states that the function Δ is constant over each equivalence class. However, this invariance can be directly used as a distinguishing property. This is actually what is done by the second mixture-differential distinguisher exhibited in [Gra18] and detailed in Theorem 3.3: Theorem 3.3 is nothing else than the combination of Theorem 3.4 with the subspace trail given by Lemma 3.1. Indeed, consider $\boldsymbol{p}^0$ and $\boldsymbol{p}^1$ in $\mathcal{C}_i + \boldsymbol{a}$ with

$\boldsymbol{p}^0 \equiv (x_0, x_1, x_2, x_3)$ and $\boldsymbol{p}^1 \equiv (y_0, y_1, y_2, y_3)$. Then, from Lemma 3.1, $R(\boldsymbol{p}^0)$ and $R(\boldsymbol{p}^1)$ belong to the same coset of $\mathcal{M}_i$ and their decompositions over the basis $\left(\texttt{MixColumns}(\boldsymbol{e}_{i-j,j})_{j\in[0,3]}\right)$ are given by

$$\begin{aligned} R\left(\boldsymbol{p}^0\right) &\equiv (\texttt{Sbox}(x_0 + a_{0,i}), \ldots, \texttt{Sbox}(x_3 + a_{3,i})) \\ R\left(\boldsymbol{p}^1\right) &\equiv (\texttt{Sbox}(y_0 + a_{0,i}), \ldots, \texttt{Sbox}(y_3 + a_{3,i}))\ . \end{aligned}$$

It follows that, if the two pairs $\{\boldsymbol{p}^0, \boldsymbol{p}^1\}$ and $\{\boldsymbol{q}^0, \boldsymbol{q}^1\}$ satisfy one of the relations given in Theorem 3.3, then $\{R(\boldsymbol{p}^0), R(\boldsymbol{p}^1)\}$ and $\{R(\boldsymbol{q}^0), R(\boldsymbol{q}^1)\}$ belong to the same equivalence class for $\sim$. We then deduce from Theorem 3.4 that Δ takes the same value on these two pairs, i.e.,

$$\mathrm{AES}^{(2)}\left(\boldsymbol{p}^0\right) + \mathrm{AES}^{(2)}\left(\boldsymbol{p}^1\right) = \mathrm{AES}^{(2)}\left(\boldsymbol{q}^0\right) + \mathrm{AES}^{(2)}\left(\boldsymbol{q}^1\right)\ .$$

Consequently,

$$\mathrm{AES}^{(2)}\left(\boldsymbol{p}^0\right) + \mathrm{AES}^{(2)}\left(\boldsymbol{p}^1\right) \in \mathcal{D}_J \iff \mathrm{AES}^{(2)}\left(\boldsymbol{q}^0\right) + \mathrm{AES}^{(2)}\left(\boldsymbol{q}^1\right) \in \mathcal{D}_J.$$

Moreover, we know from Lemma 3.1 that $\mathcal{D}_J \overset{R}{\rightrightarrows} \mathcal{C}_J \overset{R}{\rightrightarrows} \mathcal{M}_J$, implying that

$$\mathrm{AES}^{(4)}\left(\boldsymbol{p}^0\right) + \mathrm{AES}^{(4)}\left(\boldsymbol{p}^1\right) \in \mathcal{M}_J \iff \mathrm{AES}^{(4)}\left(\boldsymbol{q}^0\right) + \mathrm{AES}^{(4)}\left(\boldsymbol{q}^1\right) \in \mathcal{M}_J.$$

3.5 Adaptation to a general SPN construction

We provide in this section an extensive version of the equivalence relation and the multiple-of-8 property for a more general SPN cipher than the AES. A natural question for this extension is to find out what is the particular property of the subspaces $\mathcal{M}_I$ for Theorem 3.4 and Lemma 3.2 to work and whether these spaces could be replaced by others without altering the result. For a general SPN cipher, we analyse the form of such subspaces with respect to the non-linear layer of the cipher and provide the necessary conditions for their successful combination.

3.5.1 A more general setting for Theorem 3.4 and Lemma 3.2

Consider a general SPN cipher working on a state of N words, where the size of each word equals the cipher's S-box size. Suppose that the cipher is iterated for an arbitrary number of rounds and that the round-keys as well as the internal state are represented as word-vectors in $\mathbb{K}^N$ (and not as matrices). An SPN round R is the composition $\texttt{LinearLayer} \circ \texttt{SubCells}$ where:

- `SubCells` is the substitution layer applying an invertible $\texttt{Sbox} : \mathbb{K} \to \mathbb{K}$ to each word of the internal state in a certain basis $(\boldsymbol{f}_0, \ldots, \boldsymbol{f}_{N-1})$ of $\mathbb{K}^N$. It is important to notice that we define `SubCells` and the basis together.
- `LinearLayer` is the linear layer, a bijective $\mathbb{F}_2$-linear map of $\mathbb{K}^N$.

We now want to describe a more general subspace V of $\mathbb{K}^N$ that could play the role of $\mathcal{M}_I$ in an adaptation of Theorem 3.4 to the previously described SPN. As we have already noticed in the previous section, the proof of Theorem 3.4 relies on the fact that the coordinates of the elements in

the input subspace V can be decomposed into several subsets in such a way that each Sbox involves only one coordinate subset. This property is captured by the following definition: it requires the existence of a basis of V which can be decomposed over the original basis $(\boldsymbol{f}_0, \ldots, \boldsymbol{f}_{N-1})$ by a block-diagonal matrix.

Definition 3.4. Let V be a subspace of $\mathbb{K}^N$. We say that V is *compatible* with `SubCells` if there exists a basis of V whose elements written in the basis $(\boldsymbol{f}_0, \ldots, \boldsymbol{f}_{N-1})$ form a block-diagonal matrix (with blocks having potentially different dimensions) for a certain order of the elements in both bases. We call such a basis of V a *compatibility basis*.

Given an arbitrary basis of a subspace V, it is quite easy to check whether V is compatible with `SubCells` by computing the unique reduced-echelon form of the corresponding matrix. If, for a given ordering of the rows, this matrix has a reduced-echelon form which is block-diagonal, then V is compatible with `SubCells` and the reduced-echelon form provides a compatibility basis. Otherwise, V is not compatible with `SubCells`.

We provide now the necessary notation for describing a compatibility basis g of a compatible subspace V. For this notation to be as clear as possible, we first give a representation of $\boldsymbol{g}$ as a collection of column vectors written in the basis $\boldsymbol{f}$.

$$\begin{pmatrix}
* & \cdots & * & & & & & & \\
\vdots & \lambda_{0,\ell,i} & \vdots & & 0 & & & 0 & \\
* & \cdots & * & & & & & & \\
 & & & * & \cdots & * & & & \\
 & 0 & & \vdots & \lambda_{k,\ell,i} & \vdots & & 0 & \\
 & & & * & \cdots & * & & & \\
 & & & & & & * & \cdots & * \\
 & 0 & & & 0 & & \vdots & \lambda_{h-1,\ell,i} & \vdots \\
 & & & & & & * & \cdots & * \\
 & 0 & & & 0 & & & 0 &
\end{pmatrix}
\begin{array}{ll}
 & \\
\leftarrow & \text{coordinate on } \boldsymbol{f}_{j_0+\ell} \\
 & \\
 & \\
\leftarrow & j_k + \ell \\
 & \\
 & \\
\leftarrow & j_{h-1} + \ell \\
 & \\
 &
\end{array}$$

$$\begin{array}{ccc}
\uparrow & \uparrow & \uparrow \\
\boldsymbol{g}_{0,i} & \boldsymbol{g}_{k,i} & \boldsymbol{g}_{h-1,i} \\
i < i_0 & i < i_k & i < i_{h-1}
\end{array}$$

- We denote by h the number of blocks. The number of basis vectors in the k-th block, $0 \leqslant k < h$ will be denoted by i_k. It must obviously hold that $\sum_{k=0}^{h-1} i_k = \dim V$.

- The basis of V will be denoted by $(\boldsymbol{g}_{k,i})_{k<h, i<i_k} \in (\mathbb{K}^N)^{\dim V}$, which means that $V = \mathrm{vect}_{\mathbb{K}} \left\{ \boldsymbol{g}_{k,i} \;\middle|\; k < h, i < i_k \right\}$. The index k of a basis element $\boldsymbol{g}_{k,i}$ stands for the block-number, while i represents the position of the vector inside the block k.

- There exist $(h+1)$ integers $j_0, \ldots, j_h$, with $j_0 = 0$ and $j_h \leqslant N$, such that for all vectors inside the k-th block, all coordinates outside the interval $\{j_k, \ldots, j_{k+1} - 1\}$ are zero.

Then, each basis vector $\boldsymbol{g}_{k,i}$ can be written as

$$\boldsymbol{g}_{k,i} = \sum_{\ell=0}^{j_{k+1}-j_k-1} \lambda_{k,\ell,i} \boldsymbol{f}_{j_k+\ell} \tag{3.3}$$

for some $\lambda_{k,\ell,i} \in \mathbb{K}$ with $0 \leqslant k < h$ and $0 \leqslant i < i_k$.

Example 3.2. For the AES, we have $N = 16$ and the original basis $(\boldsymbol{f}_0, \ldots, \boldsymbol{f}_{15})$ is formed by the vectors $\boldsymbol{e}_{i,j}$ with $i, j \in \{0, \ldots, 3\}$. Let $I = \{t_0, \ldots, t_{r-1}\}$ be a subset of size r of $\{0, \ldots, 3\}$. Then, $\mathcal{M}_I$ is compatible with the AES S-box layer. A compatibility basis of $\mathcal{M}_I$ is given by

$$\boldsymbol{g}_{k,i} = \texttt{MixColumns}(\boldsymbol{e}_{t_i-k,k}) \text{ for } 0 \leqslant k < 4 \text{ and } 0 \leqslant i < r \ .$$

Indeed, when the elements of the original basis are ordered by $\boldsymbol{f}_{4j+i} = \boldsymbol{e}_{i,j}$ for $i, j \in \{0, \ldots, 3\}$, $\left(\boldsymbol{g}_{0,0}, \ldots, \boldsymbol{g}_{3,r-1}\right)$ is obtained by multiplying $(\boldsymbol{f}_0, \ldots, \boldsymbol{f}_{15})$ by a matrix with $h = 4$ blocks, all of size r. This comes from the fact that, for $j_k = 4k$, we have

$$\boldsymbol{g}_{k,i} = \texttt{MixColumns}(\boldsymbol{e}_{t_i-k,k}) = \sum_{\ell=0}^{3} m_{\ell,t_i-k} \boldsymbol{e}_{\ell,k} = \sum_{\ell=0}^{j_{k+1}-j_k-1} m_{\ell,t_i-k} \boldsymbol{f}_{j_k+\ell} \ ,$$

where $m_{i,j}$ are the coefficients of the 4×4 matrix defining `MixColumns`. This is exactly the block-diagonal form described by Equation (3.3).

For example, a basis of $\mathcal{M}_0$ can be written as follows, where . corresponds to 0.

$$\begin{pmatrix} A & . & . & . \\ . & B & . & . \\ . & . & C & . \\ . & . & . & D \end{pmatrix} \quad \text{where} \quad A = \begin{pmatrix} 2 \\ 1 \\ 1 \\ 3 \end{pmatrix}, \ B = \begin{pmatrix} 1 \\ 1 \\ 3 \\ 2 \end{pmatrix}, \ C = \begin{pmatrix} 1 \\ 3 \\ 2 \\ 1 \end{pmatrix}, \ D = \begin{pmatrix} 3 \\ 2 \\ 1 \\ 1 \end{pmatrix}.$$

From now on, we fix $\boldsymbol{a} \in \mathbb{K}^N$ and a subspace V compatible with `SubCells` with compatibility basis $\boldsymbol{g}$. The notion of information set and the equivalence relation between pairs of elements in $(\boldsymbol{a} + V)$ now involve the decomposition of the elements in V over the basis $\left\{\boldsymbol{g}_{k,i}\right\}$, where the coordinates corresponding to the same k are gathered together. The next definition adapts the notion of information set to this context.

Definition 3.5 (Information set)**.** Let $\{\boldsymbol{p}^0, \boldsymbol{p}^1\}$ be an unordered pair of elements from $V + \boldsymbol{a}$, written as

$$\boldsymbol{p}^0 = \sum_{k=0}^{h-1} \sum_{i=0}^{i_k-1} p^0_{i,k} \boldsymbol{g}_{i,k} + \boldsymbol{a} \quad \text{and} \quad \boldsymbol{p}^1 = \sum_{k=0}^{h-1} \sum_{i=0}^{i_k-1} p^1_{i,k} \boldsymbol{g}_{i,k} + \boldsymbol{a}$$

for some (uniquely defined) $p^0_{i,k}, p^1_{i,k} \in \mathbb{K}$. The *information set* K of $\{\boldsymbol{p}^0, \boldsymbol{p}^1\}$ is defined as

$$K = \left\{ k \in \{0, \ldots, h-1\} \;\middle|\; \exists i < i_k : p^0_{i,k} \neq p^1_{i,k} \right\} \ .$$

Similarly, we define an equivalence relation between pairs of inputs by considering all pairs of elements in $(\boldsymbol{a} + V)$ obtained by exchanging the sets of coordinates corresponding to the same k.

Definition 3.6 (Equivalence relation)**.** Let $P = \{\boldsymbol{p}^0, \boldsymbol{p}^1\}$ and $Q = \{\boldsymbol{q}^0, \boldsymbol{q}^1\}$ with $\boldsymbol{p}^0$, $\boldsymbol{p}^1$, $\boldsymbol{q}^0$, $\boldsymbol{q}^1$ $\in (V + \boldsymbol{a})$. We say that $P \sim Q$ if:

- $\{\boldsymbol{p}^0, \boldsymbol{p}^1\}$ and $\{\boldsymbol{q}^0, \boldsymbol{q}^1\}$ have the same information set K.
- $\forall k \in K, \exists b \in \{0,1\} : \forall i < i_k, q^0_{i,k} = p^b_{i,k}$ and $q^1_{i,k} = p^{1-b}_{i,k}$.

$\sim$ is an equivalence relation on the set of unordered pairs of elements in $(V + \boldsymbol{a})$.

The next theorem is an adaptation of Theorem 3.4 to any subspace V compatible with `SubCells`.

Theorem 3.5. *For any $\boldsymbol{a} \in \mathbb{K}^N$, the function Δ operating on unordered pairs of elements in $(V + \boldsymbol{a})$ and defined by*

$$\Delta : \quad \{\boldsymbol{p}^0, \boldsymbol{p}^1\} \quad \longmapsto \quad R\left(\boldsymbol{p}^0\right) + R\left(\boldsymbol{p}^1\right)$$

is constant over the equivalence classes for $\sim$.

Proof. Let $P = \{\boldsymbol{p}^0, \boldsymbol{p}^1\}, Q = \{\boldsymbol{q}^0, \boldsymbol{q}^1\}$ such that $P \sim Q$. We have with the previously introduced notation

$$\boldsymbol{p}^0 = \sum_{k=0}^{h-1} \sum_{i=0}^{i_k-1} p^0_{i,k} \boldsymbol{g}_{i,k} + \boldsymbol{a} = \sum_{k=0}^{h-1} \sum_{\ell=0}^{j_{k+1}-j_k-1} \left(\sum_{i=0}^{i_k-1} p^0_{i,k} \lambda_{i,k,\ell} + a_{j_k+\ell} \right) \boldsymbol{f}_{j_k+\ell}$$

where the last equality is obtained by replacing each $\boldsymbol{g}_{i,k}$ by its decomposition on the basis $(\boldsymbol{f}_0, \ldots, \boldsymbol{f}_{N-1})$ given by Equation (3.3). Then

$$\texttt{SubCells}\left(\boldsymbol{p}^0\right) = \sum_{k=0}^{h-1} \sum_{\ell=0}^{j_{k+1}-j_k-1} \texttt{Sbox}\left(\sum_{i=0}^{i_k-1} p^0_{i,k} \lambda_{i,k,\ell} + a_{j_k+\ell} \right) \boldsymbol{f}_{j_k+\ell}$$

and the difference $\texttt{SubCells}\left(\boldsymbol{p}^0\right) + \texttt{SubCells}\left(\boldsymbol{p}^1\right)$ can be written as:

$$\sum_{k,\ell} \left[\texttt{Sbox}\left(\sum_{i=0}^{i_k-1} p^0_{i,k} \lambda_{i,k,\ell} + a_{j_k+\ell} \right) + \texttt{Sbox}\left(\sum_{i=0}^{i_k-1} p^1_{i,k} \lambda_{i,k,\ell} + a_{j_k+\ell} \right) \right] \boldsymbol{f}_{j_k+\ell} \tag{3.4}$$

It is now clear with Definition 3.6 and Equation (3.4) that $\texttt{SubCells}\left(\boldsymbol{p}^0\right) + \texttt{SubCells}\left(\boldsymbol{p}^1\right)$ and $\texttt{SubCells}\left(\boldsymbol{q}^0\right) + \texttt{SubCells}\left(\boldsymbol{q}^1\right)$ are equal in $\mathbb{K}^N$. Therefore,

$$\begin{aligned}
\Delta(P) &= R\left(\boldsymbol{p}^0\right) + R\left(\boldsymbol{p}^1\right) \\
&= \texttt{LinearLayer}\left(\texttt{SubCells}\left(\boldsymbol{p}^0\right) + \texttt{SubCells}\left(\boldsymbol{p}^1\right)\right) \\
&= \texttt{LinearLayer}\left(\texttt{SubCells}\left(\boldsymbol{q}^0\right) + \texttt{SubCells}\left(\boldsymbol{q}^1\right)\right) \\
&= \Delta(Q).
\end{aligned}$$

□

From this invariance theorem, we can derive, as in the case of the AES distinguisher, some information on the number of unordered pairs $\{\boldsymbol{p}^0, \boldsymbol{p}^1\}$ such that $\Delta(\{\boldsymbol{p}^0, \boldsymbol{p}^1\})$ belongs to a given set $\mathcal{E}$. This consequence is similar to the multiple-of-8 property, but the divisibility of this number depends on the structure of the subspace V we consider. This comes from the sizes of the equivalence classes, which are determined in the following proposition.

Proposition 3.2. *Let $\mathfrak{C}$ be an equivalence class with information set K. The cardinality of $\mathfrak{C}$ is*

$$|\mathfrak{C}| = 2^{|K|-1+d\sum_{k\notin K} i_k}.$$

It is always a multiple of 2^{h-1}.

Proof. We have $\prod_{k\notin K}(2^d)^{i_k}$ choices for the shared coordinates in a pair of $\mathfrak{C}$. Those coordinates fixed, we have to make for all $k \in K$ the choice $b = 0$ or $b = 1$, i.e. $2^{|K|}$ choices. Since we are counting unordered pairs, we have $2^{|K|-1+d\sum_{k\notin K} i_k}$ pairs in $\mathfrak{C}$. The exponent $\left(|K| - 1 + d\sum_{k\notin K} i_k\right)$ is minimal for $|K| = h$. Indeed,

$$|K| - 1 + d\sum_{k\notin K} i_k = d \cdot \dim V - 1 - \sum_{k\in K}(d \cdot i_k - 1)$$

which obviously decreases as K gets bigger. Hence $|\mathfrak{C}| \equiv 0 \bmod 2^{h-1}$. □

We then deduce the following generalisation of the multiple-of-8 property.

Corollary 3.2. *Let $\mathcal{E}$ be any subset of $\mathbb{K}^N$ and*

$$n = \left|\left\{ \left\{\boldsymbol{p}^0, \boldsymbol{p}^1\right\} \text{ with } \boldsymbol{p}^0, \boldsymbol{p}^1 \in (V + \boldsymbol{a}) \;\middle|\; R\left(\boldsymbol{p}^0\right) + R\left(\boldsymbol{p}^1\right) \in \mathcal{E} \right\}\right|.$$

Then $n \equiv 0 \bmod 2^{h-1}$.

The proof is the same as the proof of Corollary 3.1, but we detail it for the sake of completeness.

Proof. We denote by $\mathcal{P}^2(V + \boldsymbol{a})$ the set of all unordered pairs of elements in $V + \boldsymbol{a}$ and by $\mathcal{P}^2(\mathcal{M}_I + \boldsymbol{a})/\sim$ the set of all equivalence classes for $\sim$. Since the equivalence classes form a partition of $\mathcal{P}^2(\mathcal{M}_I + \boldsymbol{a})$, we have that

$$n = \left|\Delta^{-1}(\mathcal{E})\right| = \sum_{\mathfrak{C}\in\mathcal{P}^2(\mathcal{M}_I+\boldsymbol{a})/\sim} \left|\Delta^{-1}(\mathcal{E}) \cap \mathfrak{C}\right| .$$

We know from Theorem 3.4 that Δ is constant on the equivalence classes, implying that

$$\left|\Delta^{-1}(\mathcal{E}) \cap \mathfrak{C}\right| = \delta(\mathfrak{C}) \times |\mathfrak{C}|$$

for some function δ from $\mathcal{P}^2(\mathcal{M}_I + \boldsymbol{a})/\sim$ into $\{0, 1\}$. It follows that

$$n = \sum_{\mathfrak{C}\in\mathcal{P}^2(\mathcal{M}_I+\boldsymbol{a})/\sim} \delta(\mathfrak{C}) \times |\mathfrak{C}| \equiv 0 \bmod 2^{h-1},$$

since, by Proposition 3.2, all equivalence classes have a cardinality divisible by 2^{h-1}. □

3.5.2 A new proof of Theorem 3.2

As a first illustration, we now show that the mixture-differential distinguisher described by Theorem 3.2 and originally stated in [Gra18] can be seen as a direct application of Theorem 3.5. In this case, the compatible subspace V is

$$\mathrm{vect}_{\mathbb{K}}\left(\texttt{MixColumns}(\boldsymbol{e}_{0,i}), \texttt{MixColumns}(\boldsymbol{e}_{1,i})\right) = \mathcal{M}_i \cap \mathcal{C}_{0,1} \text{ with } i \in \{0, \ldots, 3\}\,.$$

The proof of Theorem 3.2 is then similar to the one presented in Section 3.4.4, but for a different subspace V. We fix $\boldsymbol{a} \in \mathcal{M}_4(\mathbb{K})$ for the rest of this section.

Proof. Let $V \overset{\text{def}}{=} \mathcal{M}_i \cap \mathcal{C}_{0,1}$. A basis of this subspace is composed of the two column vectors $\boldsymbol{g}_{0,0} = \texttt{MixColumns}(\boldsymbol{e}_{0,i})$ and $\boldsymbol{g}_{1,0} = \texttt{MixColumns}(\boldsymbol{e}_{1,i})$ and its decomposition over the canonical basis is

$$\begin{pmatrix} A & . \\ . & B \\ . & . \\ . & . \end{pmatrix} \quad \text{where} \quad A = \begin{pmatrix} 2 \\ 1 \\ 1 \\ 3 \end{pmatrix} \quad \text{and} \quad B = \begin{pmatrix} 1 \\ 1 \\ 3 \\ 2 \end{pmatrix}.$$

This justifies that V is compatible with the AES substitution layer. Let $\boldsymbol{p}^0,\ \boldsymbol{p}^1,\ \boldsymbol{q}^0,\ \boldsymbol{q}^1 \in \mathrm{vect}_{\mathbb{K}}(\boldsymbol{e}_{0,i}, \boldsymbol{e}_{1,i}) + \boldsymbol{a}$ such that

$$\boldsymbol{p}^0 \equiv (x_0, x_1), \quad \boldsymbol{p}^1 \equiv (y_0, y_1) \quad \text{and} \quad \boldsymbol{q}^0 \equiv (x_0, y_1), \quad \boldsymbol{q}^1 \equiv (y_0, x_1)\,.$$

Then,

$$\begin{aligned} R\left(\boldsymbol{p}^0\right) &\equiv \left(\texttt{Sbox}\left(x_0 + a_{0,i}\right), \texttt{Sbox}\left(x_1 + a_{1,i}\right)\right) \\ \text{and} \quad R\left(\boldsymbol{p}^1\right) &\equiv \left(\texttt{Sbox}\left(y_0 + a_{0,i}\right), \texttt{Sbox}\left(y_1 + a_{1,i}\right)\right)\,. \end{aligned}$$

This exactly means that $\{R(\boldsymbol{p}^0), R(\boldsymbol{p}^1)\}$ and $\{R(\boldsymbol{q}^0), R(\boldsymbol{q}^1)\}$ are equivalent pairs of a coset of V. Theorem 3.5 then gives that $\mathrm{AES}^{(2)}(\boldsymbol{p}^0) + \mathrm{AES}^{(2)}(\boldsymbol{p}^1) = \mathrm{AES}^{(2)}(\boldsymbol{q}^0) + \mathrm{AES}^{(2)}(\boldsymbol{q}^1)$. Consequently,

$$\mathrm{AES}^{(2)}\left(\boldsymbol{p}^0\right) + \mathrm{AES}^{(2)}\left(\boldsymbol{p}^1\right) \in \mathcal{D}_J \iff \mathrm{AES}^{(2)}\left(\boldsymbol{q}^0\right) + \mathrm{AES}^{(2)}\left(\boldsymbol{q}^1\right) \in \mathcal{D}_J.$$

Since $\mathcal{D}_J \overset{2R}{\rightrightarrows} \mathcal{M}_J$, this equivalently means that

$$\mathrm{AES}^{(4)}\left(\boldsymbol{p}^0\right) + \mathrm{AES}^{(4)}\left(\boldsymbol{p}^1\right) \in \mathcal{M}_J \iff \mathrm{AES}^{(4)}\left(\boldsymbol{q}^0\right) + \mathrm{AES}^{(4)}\left(\boldsymbol{q}^1\right) \in \mathcal{M}_J.$$

□

Obviously the same result also holds for any subspace V formed by the intersection between $\mathcal{M}_i$ and another subset (unless the matrix defining the corresponding basis has a single block).

3.6 Applications

In this section we provide some applications of Theorem 3.5 to SPN ciphers other than the AES. The goal is to show in practice that the mixture-differential distinguishers and the *multiple-of* property are not proper to the AES but that they hold for many other SPN constructions. Furthermore, as a result of the adaptation provided in the previous section, the application to other ciphers is almost straightforward. Therefore, we adapt Theorem 3.1 to the SPN ciphers `LED` [Guo+11], `Midori` [Ban+15], `KLEIN` [GNL12] and `SKINNY` [Bei+16] and discuss why the result does not adapt to `CRYPTON` [Lim99] or `PRINCE` [Bor+12].

In practice, for finding a *multiple-of* property for some cipher, two conditions must be met. The first one is the existence of a subspace V compatible with the substitution layer for the *multiple-of* property through one round to hold. This condition is described through Corollary 3.2, which can be seen as a general replacement of Lemma 3.2. The second condition is the existence of exact subspace trails covering some rounds before and after the central round on which the *multiple-of* property is found. For this, we need a result equivalent to Lemma 3.1 for each analysed cipher. For searching for subspace trails that can replace the subspaces $\mathcal{D}_I, \mathcal{C}_I$ and $\mathcal{M}_I$ in Lemma 3.1, we use an algorithm proposed by Leander, Tezcan and Wiemer in [LTW18] as Algorithm 2.

3.6.1 Finding the longest exact subspace trails

In this section, we briefly recall the method of Leander et al. in [LTW18, Section 4] to find subspace trails in an SPN cipher. We reuse the notation of Section 3.5 describing a general SPN cipher. In particular, $(\boldsymbol{f}_0, \ldots, \boldsymbol{f}_{N-1})$ is the basis of $\mathbb{K}^N$ defined by `SubCells`.

Notation 3.1. Let $I \subseteq [0, N-1]$, we denote the vector space $\mathrm{vect}_{\mathbb{K}}(\boldsymbol{f}_i \mid i \in I)$ by $\mathcal{E}_I$.

We have for all I that $\mathcal{E}_I \overset{\texttt{SubCells}}{\rightrightarrows} \mathcal{E}_I$. In fact, Proposition 3.3 says that this is optimal for some Sboxes like the AES Sbox.

Definition 3.7 (Linear structures)**.** Let $F : \mathbb{K} \to \mathbb{K}$. The set of linear structures of F is

$$\mathrm{LS}(F) \overset{\text{def}}{=} \{\, (\alpha, u) \in \mathbb{K} \mid \exists c_{\alpha,u} \in \mathbb{F}_2 :\ \forall x,\ \langle\, \alpha \mid F(x+u) + F(x) \,\rangle = c_{\alpha,u} \,\}\,.$$

The set of *trivial* linear structures is $(\{0\} \times \mathbb{K}) \cup (\mathbb{K} \times \{0\})$.

In particular, the Sboxes of the AES, `KLEIN` and `PRINCE` *only* have trivial linear structures.

Proposition 3.3 (Proposition 2 in [LTW18])**.** *Let $U, V \subseteq \mathbb{K}^N$ such that $U \overset{\texttt{SubCells}}{\rightrightarrows} V$. Then if the Sbox has no non-trivial linear structures, $\exists I \subseteq [0, N-1] :\ U = \mathcal{E}_I$ and $\mathcal{E}_I \subseteq V$.*

Algorithm 2 in [LTW18] then exhaustively computes all subspace trails of the form

$$\mathcal{E}_I \overset{R}{\rightrightarrows} \mathcal{E}_J \tag{3.5}$$

for which we necessarily have that $\texttt{LinearLayer}(\mathcal{E}_I) \subseteq \mathcal{E}_J$. We rewrite it here as Algorithm 5. From the 1-round trails given by Algorithm 5, we can exhaustively compute the exact subspace trails of any length ℓ of the form

$$\mathcal{E}_{I_0} \overset{R}{\rightrightarrows} \cdots \overset{R}{\rightrightarrows} \mathcal{E}_{I_{\ell-1}} \overset{R}{\rightrightarrows} \texttt{LinearLayer}\left(\mathcal{E}_{I_{\ell-1}}\right).$$

Algorithm 5 [LTW18, Algorithm 2]. Exhaustive computation of the trails of the form of Equation (3.5).

input
    `LinearLayer`
output
    Table `Tab` such that $\forall I \subset [0, N-1]$, `Tab`$[I]$ is the smallest set J such that `LinearLayer` $(\mathcal{E}_I) \subseteq \mathcal{E}_J$.
Create a table `Tab` indexed by subsets of $[0, N-1]$.
for all $I \subset [0, N-1]$ **do**
    $J \leftarrow \emptyset$
    for all $i \in I$ **do**
        $\boldsymbol{x} \leftarrow$ `LinearLayer` $(\boldsymbol{f}_i)$
        $J \leftarrow J \cup \mathrm{supp}_{\mathbb{K}}(\boldsymbol{x})$.
    `Tab`$[I] \leftarrow J$.
return `Tab`

Algorithm 5 also works when the Sbox has linear structures but it is not necessarily optimal as Proposition 3.3 does not apply anymore. Leander et al. study this case in [LTW18, Section 5] where they give an algorithm to upper bound the length of subspace trails which do not end with the full space $\mathbb{K}^N$. Applying these methods did not give more interesting results than the ones of Algorithm 5 in our setting where we are interested in exact subspace trails. In particular, the Sboxes of `LED` (which is the same as `PRESENT`), `Midori` and `SKINNY` have non-trivial linear structures but we equally used Algorithm 5 to find subspace trails.

3.6.2 The cases of AES, `LED`, `Midori`, `KLEIN` and `SKINNY`

AES. First of all, a natural idea would be to use our adaptation with a longer subspace trail than the ones of Lemma 3.1 for the AES. However, as shown in [LTW18], the exact trails

$$\mathcal{D}_I \overset{R}{\rightrightarrows} \mathcal{C}_I \overset{R}{\rightrightarrows} M_I \tag{3.6}$$

are the longest possible trails for this cipher, which means that Corollary 3.2 cannot give any improvement for the AES.

`LED`. `LED` is a lightweight block cipher proposed by Guo et al. [Guo+11]. Its round function has the same structure as the AES and exhibits the same two-round subspace trails of Equation (3.6), leading to a 5-round distinguisher.

`Midori`. `Midori` is a lightweight block cipher designed by Banik et al. [Ban+15], optimized with respect to the energy consumption. The round function $R_{\texttt{Midori}}$ is the composition $\texttt{LinearLayer}_{\texttt{Midori}} \circ \texttt{SubCells}_{\texttt{Midori}}$, where $\texttt{LinearLayer}_{\texttt{Midori}} = \texttt{MixColumns}_{\texttt{Midori}} \circ \texttt{ShuffleCells}$. $\texttt{MixColumns}_{\texttt{Midori}}$

applies the binary involutive matrix

$$\begin{pmatrix} 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 0 \end{pmatrix}$$

to the columns of the state. The branch number of the above matrix is 4. `ShuffleCells` permutes the words of the state as follows:

$$(s_0, s_1, \ldots, s_{15}) \leftarrow (s_0, s_7, s_{14}, s_9, s_5, s_2, s_{11}, s_{12}, s_{15}, s_8, s_1, s_6, s_{10}, s_{13}, s_4, s_3)$$

where the words are numbered column-wise. As in Section 3.2, we define the following subspaces depending on the linear components of `Midori`'s round-function. If $i \in \{0, 1, 2, 3\}$,

$$\begin{aligned} \mathcal{C}_i &= \mathrm{vect}_{\mathbb{K}}\left(\boldsymbol{e}_{0,i}, \boldsymbol{e}_{1,i}, \boldsymbol{e}_{2,i}, \boldsymbol{e}_{3,i}\right), \\ \mathcal{D}_i^{\texttt{Midori}} &= \texttt{ShuffleCells}^{-1}(\mathcal{C}_i), \\ \mathcal{M}_i^{\texttt{Midori}} &= \texttt{LinearLayer}_{\texttt{Midori}}(\mathcal{C}_i). \end{aligned}$$

Applying Algorithm 5 gives that the longest exact subspace trails are the two-round trails of the form

$$\mathcal{D}_I^{\texttt{Midori}} \overset{R_{\texttt{Midori}}}{\rightrightarrows} \mathcal{C}_I \overset{R_{\texttt{Midori}}}{\rightrightarrows} \mathcal{M}_I^{\texttt{Midori}}.$$

Besides, $\mathcal{M}_I^{\texttt{Midori}}$ has a basis whose matrix is block-diagonal with four blocks. For example, $\mathcal{M}_0^{\texttt{Midori}}$ has a basis whose representation as a collection of column vectors is given in Figure 3.1a. We then have by Corollary 3.2 that for all $I, J \subseteq \{0, 1, 2, 3\}$, $\boldsymbol{a} \in \mathbb{F}_{2^d}^{16}$,

$$\left|\left\{ \left\{\boldsymbol{p}^0, \boldsymbol{p}^1\right\} \text{ with } \boldsymbol{p}^0, \boldsymbol{p}^1 \in \mathcal{D}_I^{\texttt{Midori}} + \boldsymbol{a} \;\middle|\; \texttt{Midori}^{(5)}\left(\boldsymbol{p}^0\right) + \texttt{Midori}^{(5)}\left(\boldsymbol{p}^1\right) \in \mathcal{M}_J^{\texttt{Midori}} \right\}\right| \equiv 0 \bmod 8,$$

which gives a 5-round distinguisher similar to the AES-one. It is worth noticing that the property holds even if the branch number of the `MixColumns` operation in `Midori` is only 4.

`KLEIN`. `KLEIN` is a lightweight block cipher proposed in 2011 by Gong et al. [GNL12]. The round function of `KLEIN` $R_{\texttt{KLEIN}} = \texttt{LinearLayer}_{\texttt{KLEIN}} \circ \texttt{SubCells}_{\texttt{KLEIN}}$ can be seen as the application of a non-linear layer $\texttt{SubCells}_{\texttt{KLEIN}}$ followed by a linear layer $\texttt{LinearLayer}_{\texttt{KLEIN}}$. The linear layer $\texttt{LinearLayer}_{\texttt{KLEIN}} = \texttt{MixNibbles} \circ \texttt{RotateNibbles}$ is the composition of `RotateNibbles`, which rotates the state two bytes to the left and `MixNibbles`. This last operation applies the AES `MixColumns` transformation to each half of the state. We denote the canonical basis of $\mathbb{F}_{2^4}^{16}$ by $(\boldsymbol{f}_i)_{0 \leqslant i < 16}$ and then define the following subspaces for $i \in \{0, 1\}$:

$$\begin{aligned} \mathcal{C}_i &= \mathrm{vect}_{\mathbb{K}}\left(\boldsymbol{f}_k \mid 0 \leqslant k < 8\right), \\ \mathcal{D}_i^{\texttt{KLEIN}} &= \texttt{RotateNibbles}^{-1}(\mathcal{C}_i), \\ \mathcal{M}_i^{\texttt{KLEIN}} &= \texttt{LinearLayer}_{\texttt{KLEIN}}(\mathcal{C}_i). \end{aligned}$$

Algorithm 5 gives that the longest exact subspace trails are two-round trails of the form

$$\mathcal{D}_i^{\texttt{KLEIN}} \overset{R_{\texttt{KLEIN}}}{\rightrightarrows} \mathcal{C}_i \overset{R_{\texttt{KLEIN}}}{\rightrightarrows} \mathcal{M}_i^{\texttt{KLEIN}}.$$

$$\begin{pmatrix}
0 & . & . & . \\
1 & . & . & . \\
1 & . & . & . \\
1 & . & . & . \\
. & 1 & . & . \\
. & 1 & . & . \\
. & 1 & . & . \\
. & 0 & . & . \\
. & . & 1 & . \\
. & . & 0 & . \\
. & . & 1 & . \\
. & . & 1 & . \\
. & . & . & 1 \\
. & . & . & 1 \\
. & . & . & 1 \\
. & . & . & 0
\end{pmatrix}$$

(a) $\mathcal{M}_0^{\texttt{Midori}}$ basis

$$\begin{pmatrix}
2 & . & 3 & . & . & . & . & . \\
. & 2 & . & 3 & . & . & . & . \\
1 & . & 2 & . & . & . & . & . \\
. & 1 & . & 2 & . & . & . & . \\
1 & . & 1 & . & . & . & . & . \\
. & 1 & . & 1 & . & . & . & . \\
3 & . & 1 & . & . & . & . & . \\
. & 3 & . & 1 & . & . & . & . \\
. & . & . & . & 1 & . & 1 & . \\
. & . & . & . & . & 1 & . & 1 \\
. & . & . & . & 3 & . & 1 & . \\
. & . & . & . & . & 3 & . & 1 \\
. & . & . & . & 2 & . & 3 & . \\
. & . & . & . & . & 2 & . & 3 \\
. & . & . & . & 1 & . & 2 & . \\
. & . & . & . & . & 1 & . & 2
\end{pmatrix}$$

(b) $\mathcal{M}_0^{\texttt{KLEIN}}$ basis

$$\begin{pmatrix}
1 & . & . & . & . \\
. & 1 & . & . & . \\
1 & 1 & . & . & . \\
1 & . & . & . & . \\
. & . & 1 & . & . \\
. & . & 1 & . & . \\
. & . & . & . & . \\
. & . & 1 & . & . \\
. & . & . & 1 & . \\
. & . & . & . & 1 \\
. & . & . & 1 & 1 \\
. & . & . & 1 & . \\
. & . & . & . & . \\
. & . & . & . & . \\
. & . & . & . & . \\
. & . & . & . & .
\end{pmatrix}$$

(c) W_0 basis

Figure 3.1 – Compatibility basis representations for `Midori` (left) and `KLEIN` (right).

Besides, $\mathcal{M}_i^{\texttt{KLEIN}}$ has a basis whose matrix is block-diagonal with two blocks. For example, $\mathcal{M}_0^{\texttt{KLEIN}}$ has a basis whose representation as a collection of column vectors is given in Figure 3.1b. We then have by Corollary 3.2 for all $i, j \in \{0,1\}$, $\boldsymbol{a} \in \mathbb{F}_4^{16}$,

$$\left|\left\{ \left\{\boldsymbol{p}^0, \boldsymbol{p}^1\right\} \text{ with } \boldsymbol{p}^0, \boldsymbol{p}^1 \in \mathcal{D}_i^{\texttt{KLEIN}} + \boldsymbol{a} \;\middle|\; \texttt{KLEIN}^{(5)}\left(\boldsymbol{p}^0\right) + \texttt{KLEIN}^{(5)}\left(\boldsymbol{p}^1\right) \in \mathcal{M}_j^{\texttt{KLEIN}} \right\}\right| \equiv 0 \bmod 2 \,,$$

meaning that `KLEIN` has a *multiple-of-2* property for 5 rounds. Note that even if we get only a multiple of 2 in the case of `KLEIN`, as the pairs are not ordered, this can still be considered as a distinguishing property.

`SKINNY`. `SKINNY` is a family of tweakable lightweight block ciphers, designed in 2016 by Beierle et al. [Bei+16]. The round function follows a classical SPN construction $R_{\texttt{SKINNY}} = \texttt{LinearLayer}_{\texttt{SKINNY}} \circ \texttt{SubCells}_{\texttt{SKINNY}}$ where $\texttt{LinearLayer}_{\texttt{SKINNY}} = \texttt{MixColumns}_{\texttt{SKINNY}} \circ \texttt{ShiftRows}_{\texttt{SKINNY}}$. The operation $\texttt{ShiftRows}_{\texttt{SKINNY}}$ is similar to the AES `ShiftRows` operation, with the only difference that the shift is performed to the right. In $\texttt{MixColumns}_{\texttt{SKINNY}}$, each column of the state is multiplied by the following binary matrix of branch number 2:

$$\begin{pmatrix}
1 & 0 & 1 & 1 \\
1 & 0 & 0 & 0 \\
0 & 1 & 1 & 0 \\
1 & 0 & 1 & 0
\end{pmatrix}$$

Applying here again Algorithm 5 gives as result that the longest exact subspace trails are two-round-long. There are 1294 such trails. A Gaussian elimination on the last subspace of each

trail gives that among these trails, 1282 end with a subspace compatible with the substitution layer. This allows to adapt Theorem 3.1 for 5-round SKINNY, concluding that 5-round SKINNY always has the *multiple-of-*2^{h-1} property. However, it is interesting to note here that depending on the trail, the value of h varies. More precisely, $h \in \{2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 14\}$.

We now give an example of such a distinguisher for $h = 3$. For SKINNY, the internal state is classically represented as an element of $\mathcal{M}_4(\mathbb{K})$. We first exhibit two 2-round subspace trails,

$$U_i \overset{R_{\texttt{SKINNY}}}{\rightrightarrows} V_i \overset{R_{\texttt{SKINNY}}}{\rightrightarrows} W_i$$

for $i \in \{0, 1\}$ where

$$\begin{aligned} U_0 &= \mathrm{vect}_{\mathbb{K}}\left(\boldsymbol{e}_{1,1}, \boldsymbol{e}_{1,2}, \boldsymbol{e}_{1,3}, \boldsymbol{e}_{3,1}, \boldsymbol{e}_{3,3}\right), \\ U_1 &= \mathrm{vect}_{\mathbb{K}}\left(\boldsymbol{e}_{0,3}, \boldsymbol{e}_{1,0}, \boldsymbol{e}_{1,2}, \boldsymbol{e}_{1,3}, \boldsymbol{e}_{2,1}, \boldsymbol{e}_{2,3}, \boldsymbol{e}_{3,0}, \boldsymbol{e}_{3,1}, \boldsymbol{e}_{3,2}, \boldsymbol{e}_{3,3}\right), \\ V_i &= \texttt{LinearLayer}_{\texttt{SKINNY}}(U_i), \\ W_i &= \texttt{LinearLayer}_{\texttt{SKINNY}}(V_i) \end{aligned}$$

Moreover, W_0 is compatible with $h = 3$. One of its compatibility basis is represented in Figure 3.1c. We then have that

$$\left|\left\{ \left\{\boldsymbol{p}^0, \boldsymbol{p}^1\right\} \text{ with } \boldsymbol{p}^0, \boldsymbol{p}^1 \in U_0 + \boldsymbol{a} \;\middle|\; \texttt{SKINNY}^{(5)}\left(\boldsymbol{p}^0\right) + \texttt{SKINNY}^{(5)}\left(\boldsymbol{p}^1\right) \in W_1 \right\}\right| \equiv 0 \bmod 4 .$$

Finally, we recall that this section only aims at giving examples of how Theorem 3.1 can be adapted to other SPN ciphers and does not claim new cryptanalytic results. In particular, this property on 5 rounds does not threaten the overall security of the cipher that is composed of 32 rounds.

3.6.3 The cases of CRYPTON and PRINCE

CRYPTON. The block cipher CRYPTON [Lim98], designed by Lim in 1998 was among the candidates to the NIST AES competition. It has a structure very similar to the one of AES and this is why we considered it as a natural candidate for the *multiple-of* property. Indeed, the round function of CRYPTON is naturally decomposed as $R_{\texttt{CRYPTON}} = \texttt{LinearLayer}_{\texttt{CRYPTON}} \circ \texttt{SubCells}_{\texttt{CRYPTON}}$, where $\texttt{LinearLayer}_{\texttt{CRYPTON}}$ is the composition of a byte transposition of columns into rows with respect to the anti-diagonal of the internal state and a permutation at the bit level applied column-wise. Algorithm 5 gives two-round exact subspace trails. However, the problem here is that $\texttt{LinearLayer}_{\texttt{CRYPTON}}$ is only $\mathbb{F}_2$-linear and not $\mathbb{F}_{2^8}$-linear and this implies that the last subspaces in the subspace trails are not $\mathbb{F}_{2^8}$-vector subspaces and cannot verify the compatibility hypothesis of Corollary 3.2. As a consequence, Theorem 3.1 cannot be adapted to 5-round CRYPTON.

However, CRYPTON has 1-round exact subspace trails that end with subspaces of the form $\mathrm{vect}_{\mathbb{K}}\left(\boldsymbol{e}_{i,j} | i \in I, j \in \{0, 1, 2, 3\}\right)$. Those subspaces are obviously compatible with the substitution layer, and we have a 4-round version of Theorem 3.1 for CRYPTON: the first round is a 1-round exact subspace trail ending with a compatible subspace, the second round exploits Corollary 3.2 and the last two rounds are a 2-round exact subspace trail. Indeed, the trail used after Corollary 3.2 does not need to end with a compatible subspace.

PRINCE. PRINCE [Bor+12] is a block cipher proposed by Borghoff et al. in 2012 with a specific structure named α-reflection. The main parts of this structure follow the SPN construction. As for CRYPTON, the round function is defined as $R_{\texttt{PRINCE}} = \texttt{LinearLayer}_{\texttt{PRINCE}} \circ \texttt{SubCells}_{\texttt{PRINCE}}$ with $\texttt{LinearLayer}_{\texttt{PRINCE}}$ a $\mathbb{F}_2$-linear map which is not $\mathbb{F}_{2^4}$-linear. Again, as for CRYPTON, PRINCE exhibits two-round exact subspace trails but the last subspaces of those trails are not $\mathbb{F}_{2^4}$-linear subspaces. We then cannot mount a 5-round distinguisher by adapting Theorem 3.1. However, one-round exact subspace trails ending with compatible diagonal $\mathbb{F}_{2^4}$-linear subspaces allow to have a 4-round version of Theorem 3.1 for PRINCE.

3.7 Conclusion

We have presented a general result which allows cryptanalysts to search for mixture-differential distinguishers, or *multiple-of* properties, in a systematic way, for any SPN. This result then avoids the redundant proofs which were previously needed for each new occurrence of these distinguishing properties. Also, it highlights the properties of the ciphers which have to be taken into account for establishing the existence of such distinguishers and it shows that mixture-differential distinguishers directly apply to a more general class of SPNs than what was previously believed. As shown in the previous examples, all these distinguishing properties can be exhibited by combining our framework with the search for subspace trails, which is investigated in [LTW18]. Since our result, exploiting an appropriate equivalence relation, applies in many situations, it appears that the main limitation for finding efficient distinguishers is the existence of long subspace trails for the ciphers.

Chapter 4

Efficient MILP models for symmetric primitives

Contributions brought forward in this chapter were published in ToSC 2020, Issue 3, and are a joint work with Christina Boura [BC20].

4.1 Introduction

In symmetric-key cryptography, a popular technique for proving resistance against classical attacks is to model the behaviour of the cipher as a Mixed Integer Linear Programming (MILP) problem and solve it by some MILP solver. This method was applied for the first time by Mouha et al. [Mou+11] and by Wu and Wang [WW11] for finding the minimum number of differentially and linearly active Sboxes and provides in such a way a proof of resistance against these two classical attacks. Since then, the use of MILP not only by designers but also by cryptanalysts has increased, the advantage being that it is relatively easy to translate the cryptanalytic problem into linear constraints and use the available solvers to solve it.

4.1.1 MILP problems

MILP problems are linear optimization problems — also called linear programming (LP) problems — for which some variables are constrained to integer values. As for all mathematical optimization problems, a linear optimization problem is defined by a set of *variables*, a set of *constraints* and eventually an *objective* function. In the case of linear optimization,

- the variables can take values in $\mathbb{R}$,
- the constraints are linear equations or linear inequalities between variables,
- the objective function is a linear function of the variables.

A *solution* to the problem will be an assignment of the variables to some values such that the constraints are satisfied. When the problem has an objective function, the goal is to compute a solution which optimizes (i.e. maximizes or minimizes) the output of the objective function.

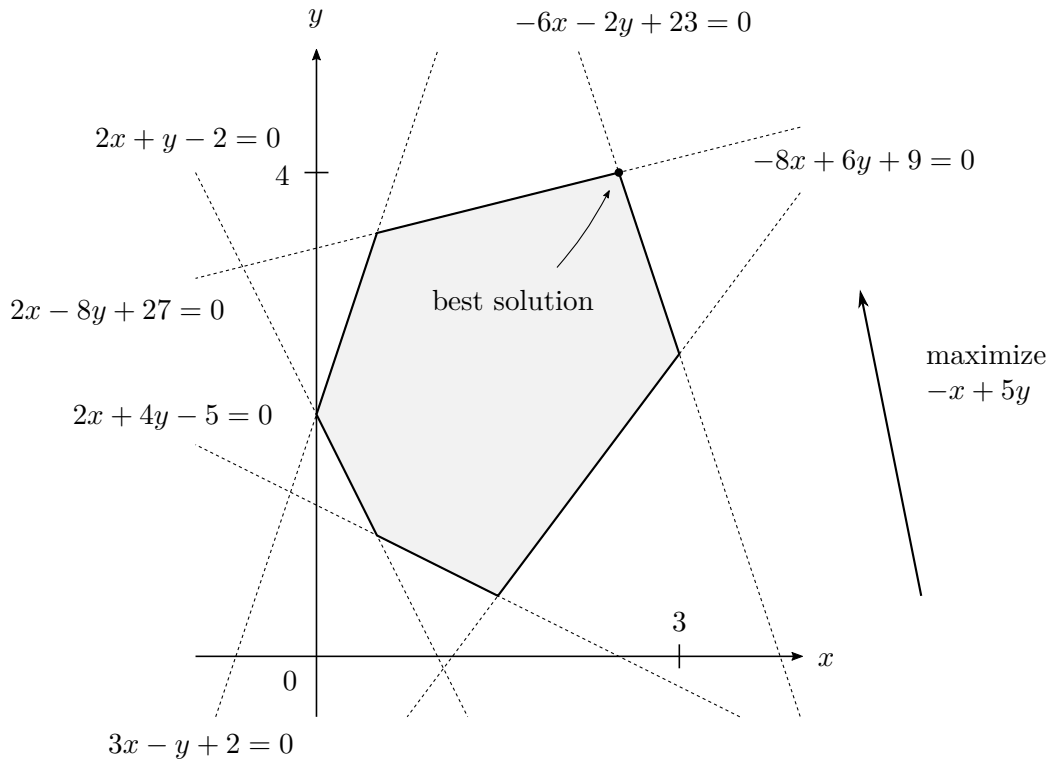

Figure 4.1 – A linear optimization problem.

Otherwise, the problem is just to exhibit a solution or to list some solutions. The general form of a linear optimization problem of *dimension* n (i.e. with n variables) is

$$\text{maximize } \langle \boldsymbol{c} \mid \boldsymbol{x} \rangle \text{ subject to } \boldsymbol{A} \cdot \boldsymbol{x} \leqslant \boldsymbol{b}$$

where $\boldsymbol{x}$ is the variable vector, $\boldsymbol{c} \in \mathbb{R}^n$ defines the linear objective function, $\langle \, . \mid . \, \rangle$ denotes the canonical scalar product in $\mathbb{R}^n$ and $\boldsymbol{A} \in \mathcal{M}_{m,n}(\mathbb{R})$ and $\boldsymbol{b} \in \mathbb{R}^m$ define m linear constraints $\langle \boldsymbol{A}_{i,*} \mid \boldsymbol{x} \rangle \leqslant b_i$. The set $\{ \boldsymbol{x} \in \mathbb{R}^n \mid \boldsymbol{A} \cdot \boldsymbol{x} \leqslant \boldsymbol{b} \}$ is called the feasible region and it is by definition a convex polyhedron of $\mathbb{R}^n$, possibly unbounded.

Example 4.1. The linear optimization problem in Figure 4.1 can be written as

$$\begin{array}{lrrr} \text{maximize} & -x + 5y & & \\ \text{subject to} & 2x + 4y - 5 \geqslant 0, & 2x + y - 2 \geqslant 0, & 3x - y + 2 \geqslant 0, \\ & 2x - 8y + 27 \geqslant 0, & -6x - 2y + 23 \geqslant 0, & -8x + 6y + 9 \geqslant 0. \end{array}$$

The best solution is given by the edge point $(2.5, 4)$ for which the objective function has the maximum value 17.5.

Linear optimization problems are rather easy to solve. Dantzig invented in 1947 the first efficient algorithm, called the simplex method, but with worst-case exponential time. Khachiyan proved in 1979 that LP problems were solvable in polynomial-time and Karmarkar proposed in 1984 a

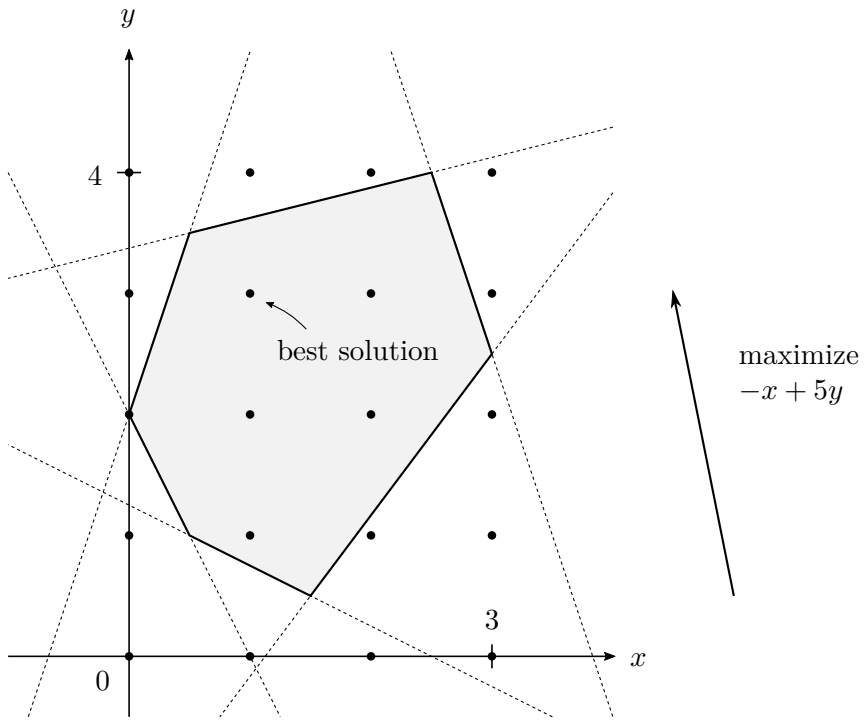

Figure 4.2 – An integer linear optimization problem.

practically efficient *and* polynomial time algorithm [Kar84]. However, many problems cannot be reduced to LP problems with real variables and a common need is to constrain some variables to *integer* values. We then get Mixed-Integer LP (MILP) problems.

Example 4.2. We can turn the LP problem in Example 4.1 into a MILP problem by simply adding the constraints that x and y are in $\mathbb{Z}$. As shown in Figure 4.2, the best solution in this case is given by the interior point with coordinates $(1, 3)$. Interestingly, the point $(2, 3)$ is closer to the best LP solution $(2.5, 4)$ for the Euclidian distance but it does not maximize the objective function.

For cryptographic applications, we will mostly be interested in the binary variants of MILP problems for which all variables are constrained to take values in $\{0, 1\} \subseteq \mathbb{Z}$. Contrarily to LP problems, the decisional version of MILP problems is NP-complete but the resolution of MILP problems has so many industrial applications that academics and companies developped solvers with very good performances. In particular, all of our MILP experiments were done with the `Gurobi` optimizer [GO20].

Reducing cryptographic problems to MILP problems then allows cryptographers to leverage the high performance of MILP solvers. Naturally, the reduction (or *modeling*) method has an important influence on the running time of the MILP solver or even on its ability to solve a given cryptographic problem. Exploring modeling methods for differential cryptanalysis is the goal of the present chapter.

4.1.2 Word-oriented models

For SPN block ciphers, it is possible to get easy and relatively small models when searching for properties at the word level. For example, we saw in Section 2.3.3 that the first MILP models used

in cryptanalysis were created by Mouha, Wang, Gu and Preneel [Mou+11] to minimize the number of active Sboxes for a given number of rounds of an SPN. In such models, the MILP variables represent together a *truncated* differential trail with respect to the Sbox layer (see Section 2.4) and the constraints represent the action of the linear layer on truncated differentials.

More precisely, the state (active or not) of each word (or cell) in a truncated differential trail is encoded in a binary MILP variable $x_i^{(r)}$ where i is the cell index and r is the round index.

Example 4.3. In this example, we want to use a MILP solver to compute the minimum number of active Sboxes in 4 AES rounds, which is known to be 25 thanks to the wide-trail strategy. We also want to compute a truncated differential trail which atteins this minimum, like the one in Figure 2.2.

- We have 4 rounds and 16 bytes per round, which makes 64 MILP variables $\left(x_{i,j}^{(r)}\right)_{r,i,j\in[0,3]}$.
- The branch number of each `MixColumns` operation adds constraints on the possible values of $\left(x_{i,j}^{(r-1)}\right)_{i\in[0,3]}$ and $\left(x_{i,j}^{(r)}\right)_{i\in[0,3]}$ for $r \in [1,3]$ and $j \in [0,3]$. It is modeled by Mouha et al. by the inequality
$$\sum_{i=0}^{3} x_{i,\ i+j \bmod 4}^{(r-1)} + \sum_{i=0}^{3} x_{i,j}^{(r)} \geqslant 5 \cdot t_{r,j}$$
where 5 is the branch number of `MixColumns` and $t_{r,j}$ is a dummy binary MILP variable. Its meaning would be that the column j during round r is active. This inequality also takes the action of `ShiftRows` into acount thanks to the indexing $(i, i + j \bmod 4)$ in the terms of the left sum.
- Since we want to minimize the number of active Sboxes, the objective function is simply the sum of all the variables.

We finally have the following MILP problem.

$$\begin{aligned} &\text{minimize} \sum_{r,i,j\in[0,3]} x_{i,j}^{(r)} \\ &\text{subject to } \sum_{i=0}^{3} x_{i,\ i+j \bmod 4}^{(r-1)} + \sum_{i=0}^{3} x_{i,j}^{(r)} \geqslant 5 \cdot t_{r,j} \text{ for all } r \in [1,3],\ j \in [0,3]. \end{aligned}$$

Such models are simple and efficient as there is a reasonable amount of MILP integer variables. They have nonetheless a few drawbacks. First, they are restricted to SPN ciphers with a word-oriented linear layer. We saw for example in Section 1.4.2 that the linear layer of the `PRESENT` block cipher is a bit permutation. Although it is a very elementary linear layer, the framework of [Mou+11] does not apply to it. Second, they are much less accurate than models at bit level. For example, modeling the propagation of the differences bitwise and looking inside the Sboxes could yield longer impossible differentials than those discovered when only truncated differences are analyzed [ST17b].

4.1.3 Bit-oriented models

Sun et al. [Sun+14b; Sun+14a] were the first to propose bit-oriented modelings for the cryptanalysis of SPN ciphers. A non-trivial problem in doing so is to find an efficient representation of the valid differential propagations through an Sbox.

Sbox modeling. Indeed, Sboxes are non-linear Boolean vectorial functions, therefore modeling their differential properties with $\mathbb{R}$-linear inequalities is not natural. Several approaches have been suggested to solve this problem. In the original works of Sun et al. [Sun+14a; Sun+14b] two different methods were notably proposed for modeling an Sbox. The first of them is a geometrical approach that consists in representing all possible transitions $\boldsymbol{a} \to \boldsymbol{b}$ through an n-bit Sbox as points $(\boldsymbol{a}, \boldsymbol{b}) \in \mathbb{R}^{2n}$ and then computing the H-representation of the convex hull of this set, that is all the geometric faces of the smallest convex containing it. This permits to remove all impossible differential transitions through the Sbox. However, as this method yields a high number of (possibly redundant) inequalities, the authors further developed a greedy algorithm for selecting a representative number of inequalities among them. The second method, based on a logical condition approach, consists in representing by linear inequalities some conditional differential properties of the Sbox. Unfortunately, the problem of these two methods is that they are not efficient for large (e.g. 8-bit) Sboxes.

To solve this problem for large Sboxes, Abdelkhalek et al. [Abd+17] observed that generating a minimal number of constraints in logical condition modeling can be converted into the problem of minimizing the product-of-sum representation of Boolean functions. This last problem is well-studied and algorithms for solving it exist, for example the *Quine-McCluskey* (QM) [Qui52; Qui55; McC56] or the *Espresso* [Bra+84] algorithms. In this way, Abdelkhalek et al. managed for the first time to generate linear constraints for 8-bit Sboxes, notably for the Sboxes of AES and `SKINNY-128` [Bei+16]. While the number of linear constraints for the Sbox of `SKINNY-128` provided in [Abd+17] is as low as 372, the same method yields 8302 linear inequalities for the Sbox of AES, a modeling that is often too heavy to be used in practice.

Linear layer modeling. Efficiently representing the Sboxes is a crucial part of the modeling process. But a bad modeling of the diffusion layer can render the optimization process very slow or even impractical. Indeed, with the exception of some ciphers, e.g. `PRESENT`, where the linear layer is just a bit-permutation, the diffusion is usually ensured by `xor` gates. Yet, the `xor` operation, while linear in $\mathbb{F}_2$ models very badly in $\mathbb{R}$. So, bitwise modeling of heavy linear layers, that need many `xor`s to be represented, can lead to impractical systems with many linear inequalities or with many dummy variables.

Dummy variables. We saw that Mouha et al. used a dummy variable to model the branch number of the `MixColumns` operation in a word-oriented model. In bit-oriented models, the use of dummy variables is also possible and actually quite popular. Indeed, one can model $x_1 \oplus \cdots \oplus x_n = 0$ with a dummy integer variable t as $x_1 + \cdots + x_n = 2 \cdot t$ or use a dummy binary variable a for recording an intermediate state in the computation. For example, the variable a could *record* the value of $x_1 \oplus \cdots \oplus x_k$ and we could model the above computation with $x_1 \oplus \cdots \oplus x_k \oplus a = 0$ and $a \oplus x_{k+1} \oplus \cdots \oplus x_n = 0$. For Sbox modelings, when the construction of the Sbox is known and well-suited, only the latter can be helpful. Typically, a complex Sbox with a known circuit to compute it could be modeled with variables recording intermediate computations. However in general, in order to minimize the running time of the MILP solver, it is important to minimize the number of integer variables. In this paper we study bitwise modelings which need by definition many MILP integer variables. We hence restricted ourselves to not introducing dummy variables and leave the use of dummy integer variables for improving performance as an open problem.

4.1.4 Our Contributions

In this chapter, we propose several new bitwise MILP modelings for the propagation of differential properties through both Sboxes and linear layers. Our methods permit to efficiently model the exact differential propagation through an SPN cipher and can be applied to prove resistance against differential cryptanalysis and its variants or to search for new differential-type attacks. Besides, the methods used for modeling the DDT of an Sbox are general enough for modeling an LAT or any Boolean function.

Modelings for large Boolean functions and Sboxes. We introduce three different modeling methods for Boolean functions, based on algebraic or geometrical approaches. With our techniques we manage to decrease significantly the number of inequalities needed to model large Sboxes. While Sasaki and Todo showed in [ST17a] that reaching the minimal number of inequalities is not necessarily the best approach for decreasing the solving time, yet a big difference in the number of inequalities leads to a real difference for the optimization process, as we demonstrate with experiments. We managed notably to represent the AES Sbox with 2882 inequalities, dividing by three the number of inequalities needed in the best previous approach.

Modeling matrices with entries in $\mathbb{F}_2$. We provide efficient modelings for linear layers without the use of dummy variables. We first explain why modeling the `xor` of several binary variables, which is the central operation in most of the matrix/vector products over $\mathbb{F}_2$, needs many inequalities. Then, we introduce new algorithms inspired from coding theory that change the modeled matrix to significantly decrease this number.

Applications to impossible differential cryptanalysis from the designer's point of view. We complete the work of Sasaki and Todo in [ST17b] and we use our modeling techniques for proving partial resistance against impossible differential cryptanalysis for the AES and `SKINNY-128`. More precisely, we show for both ciphers that, even when the Sbox details are taken into account, there are no impossible differentials with one active input/output byte for 5-round AES and 13-round `SKINNY-128`.

Organization. The rest of the chapter is organized as follows. Section 4.2 is dedicated to our different methods for modeling Boolean functions and Sboxes. Then, in Section 4.3 we present an algorithm to efficiently model linear layers. In Section 4.4, we run an experiment for illustrating the effectiveness of the previously-introduced methods. Finally, we apply our new models to impossible differential cryptanalysis in Section 4.5.

4.2 MILP Modeling for Boolean functions and Sboxes

The methods to be introduced in this section can be applied to characterize any set $\mathcal{P} \subset \{0,1\}^m$ with $\mathbb{R}$-linear inequalities. As any such subset $\mathcal{P} \subset \{0,1\}^m$ can be seen as the support of some Boolean function $F_{\mathcal{P}} : \mathbb{F}_2^m \to \mathbb{F}_2$ operating on m bits, the inequalities representing $\mathcal{P}$ model the constraint $F_{\mathcal{P}}(x_0, \ldots, x_{m-1}) = 1$. However, as our main target is the modeling of differential, linear or other behaviours through an $\texttt{Sbox} : \mathbb{F}_2^n \to \mathbb{F}_2^n$, and such behaviours can be represented (as will be

explained below) by some Boolean function, we will describe some of our techniques, without loss of generality, through the spectrum of the differential behaviour of an Sbox.

In what follows, a (differential) transition $\boldsymbol{x} \to \boldsymbol{y}$ through the Sbox will be seen as a vector of $\mathbb{F}_2^m$, involving $m = 2n$ binary variables and will be represented, depending on the context, either as $(x_0, \ldots, x_{n-1}, y_0, \ldots, y_{n-1})$ or as $(x_0, \ldots, x_{m-1})$.

4.2.1 Modeling Boolean functions and DDTs

For many problems, as for example the search for good differential characteristics, bitwise modelings are better than wordwise ones, as they are more precise and permit to follow information propagation at the bit level. In such models, a binary variable is assigned to each bit of a differential characteristic. A variable has the value 1 if the corresponding bit has a difference in the characteristic — in which case it is called active — and 0 otherwise. In a MILP model, in order to follow how the information is propagated through the different components of the cipher, each different layer has to be efficiently modeled. For Sboxes, this is typically done by looking at their DDT (see Section 2.2). When one only cares whether a differential propagation is possible or not, it is enough to model the Boolean function

$$\begin{cases} \mathbb{F}_2^{2n} & \longrightarrow & \mathbb{F}_2 \\ (\boldsymbol{x}, \boldsymbol{y}) & \longmapsto & 1 \text{ if and only if } \mathrm{DDT}(\boldsymbol{x}, \boldsymbol{y}) \neq 0. \end{cases}$$

However, if one wants to take into account the different integer values of the DDT, the authors of [Abd+17] propose to encode this information by modeling instead the Boolean functions

$$\begin{cases} \mathbb{F}_2^{2n} & \longrightarrow & \mathbb{F}_2 \\ (\boldsymbol{x}, \boldsymbol{y}) & \longmapsto & 1 \text{ if and only if } \mathrm{DDT}(\boldsymbol{x}, \boldsymbol{y}) = p, \end{cases}$$

for each value $p > 0$ such that $\exists (\boldsymbol{a}, \boldsymbol{b}) : \mathrm{DDT}(\boldsymbol{a}, \boldsymbol{b}) = p$.

As one can see, modeling a DDT is equivalent to modeling some specific Boolean function. From now on, we will focus on modeling general Boolean functions but all our examples and applications will be focused on DDTs. In all these examples we will be only interested in whether differential propagations are possible or not, without caring about their concrete probability. However, one has to remember, that all these methods can be applied to any other Boolean function and in particular to other cryptanalysis techniques whose properties can be described through some table, like the linear and the boomerang cryptanalysis [Wag99].

Example 4.4. Let us take the example of the 3-bit Sbox of Table 4.1a used inside the block cipher `PrintCipher` [Knu+10]. Then, by denoting by x_0, x_1, x_2 the three bits of the input difference $\boldsymbol{x}$ and by y_0, y_1, y_2 the three bits of the output difference $\boldsymbol{y}$, where x_0 and y_0 are the least significant bits of $\boldsymbol{x}$ and $\boldsymbol{y}$ respectively, the system of 7 inequalities in Table 4.1b is satisfied by all the valid transitions while removing the 35 impossible transitions $\boldsymbol{x} \to \boldsymbol{y}$ for which $\mathrm{DDT}(\boldsymbol{x}, \boldsymbol{y}) = 0$.

4.2.2 State of the art

Given the truth table of a Boolean function F, the question is how to efficiently model the constraint $F(\boldsymbol{x}) = 1$ by a system of $\mathbb{R}$-linear inequalities. This problem can then be divided into two sub-problems:

Table 4.1 – Value table, DDT and MILP modeling of the possible transitions of the Sbox of `PrintCipher`.

(a) Value table

$\boldsymbol{x}$	0	1	2	3	4	5	6	7
$\texttt{Sbox}(\boldsymbol{x})$	0	1	3	6	7	4	5	2

(b) MILP modeling

$$
\begin{aligned}
-2x_0 - 2x_1 + x_2 - 2y_0 - 2y_1 + y_2 + 6 &\geqslant 0\\
-2x_0 + x_1 - 2x_2 - 2y_0 + y_1 - 2y_2 + 6 &\geqslant 0\\
x_0 - 2x_1 - 2x_2 + y_0 - 2y_1 - 2y_2 + 6 &\geqslant 0\\
x_0 + 2x_1 + 4x_2 + 3y_0 + 2y_1 - 4y_2 &\geqslant 0\\
-3x_0 + 2x_1 - x_2 + 4y_0 + 2y_1 + 4y_2 &\geqslant 0\\
4x_0 - 2x_1 + x_2 - 2y_0 + 4y_1 + 3y_2 &\geqslant 0
\end{aligned}
$$

(c) DDT

	$\boldsymbol{b}$							
$\boldsymbol{a}$	0	1	2	3	4	5	6	7
0	8	0	0	0	0	0	0	0
1	0	2	0	2	0	2	0	2
2	0	0	2	2	0	0	2	2
3	0	2	2	0	0	2	2	0
4	0	0	0	0	2	2	2	2
5	0	2	0	2	2	0	2	0
6	0	0	2	2	2	2	0	0
7	0	2	2	0	2	0	0	2

Inequality generation. How to generate a (possibly large) set of inequalities on variables $x_0, \ldots, x_{m-1}$ that correctly models $F(\boldsymbol{x}) = 1$?

Efficient subset. How to choose a (typically much smaller) subset of this set of inequalities that still correctly models $F(\boldsymbol{x}) = 1$ but leads to more efficient MILP models?

For differential cryptanalysis, the above general problem corresponds to modeling the fact that $(x_0, \ldots, x_{n-1}) \to (x_n, \ldots, x_{2n-1})$ is a possible transition in a DDT. To solve the *Inequality generation* problem, two different approaches were proposed in 2014 by Sun et al. [Sun+14b; Sun+14a]. The first is a geometrical one and consists in computing the H-representation of the convex hull of the set of possible transitions. The second one is based on logical condition modeling. Below, we briefly explain these two methods.

Convex hull inequalities. The method of the H-representation of the convex hull consists, as its name suggests, in computing the H-representation of the convex hull of all possible points $\boldsymbol{a} \in \mathbb{F}_2^m$ such that $F(\boldsymbol{a}) = 1$ seen as vectors of $\mathbb{R}^m$. Taking then the $(m-1)$-dimensional faces of the convex hull yields a correct set of inequalities. The H-representation can be for example computed through an algebra computer system such as `Sage` [The20] and gives a system of linear inequalities excluding all impossible points (e.g. all points $\boldsymbol{a}$ such that $F(\boldsymbol{a}) = 0$).

Logical inequalities. The second method proposed by Sun et al. and called logical condition modeling is based on the idea that each impossible point can be removed by a single inequality in a simple way. Indeed, consider an impossible point $\boldsymbol{a} = (a_0, \ldots, a_{m-1})$. Then, the inequality

$$\sum_{i=0}^{m-1} (1 - a_i)x_i + a_i(1 - x_i) \geqslant 1 \tag{4.1}$$

only discards this point $\boldsymbol{a}$. Lets take as example the DDT of `PrintCipher`. As it can be seen from Table 4.1c, (`0x1,0x6`) is an impossible transition through the DDT. By writing the input and output in a bitwise manner we have that $(100) \nrightarrow (011)$. The above formula gives the inequality

$$-x_0 + x_1 + x_2 + y_0 - y_1 - y_2 \geqslant -2$$

that is satisfied by all points in $\mathbb{F}_2^6$ but (`0x1`, `0x6`). This method can then be applied to all impossible transitions $\boldsymbol{x} \to \boldsymbol{y}$ and yields easily a system of inequalities containing as many constraints as the number of impossible transitions through the DDT, or as the number of zeros of the Boolean function in the general case.

Towards modeling 8-bit Sboxes. However, as mentioned in [Abd+17], both these methods that provide a solution for the *Inequality generation* problem have the disadvantage of not being efficient for modeling 8-bit Sboxes. For the first one, computing the H-representation of the convex hull for such big Sboxes is nearly impossible, while the second method yields a very large number of initial inequalities which, by construction, cannot have a strict subset as a solution to the *Efficient subset* problem. For example, the `SKINNY-128` Sbox has 54067 impossible transitions and this number of inequalities is too high to represent the Sbox for any related MILP problem.

In 2017, Abdelkhalek et al. [Abd+17] made an important step forwards in the logical-condition-modeling direction, by translating the problem of searching for good inequalities for 8-bit Sboxes into the classical problem of minimization of the product-of-sum representation of the related Boolean function and by using the Quine-McCluskey (QM) algorithm to solve it. This method permits to solve at once the two steps of the Sbox modeling: Find many good inequalities (the prime implicants in the QM vocabulary) and keep among them a good representative set. In the case of QM this representative set corresponds to the minimal number of equations. The problem however of QM is that in practice it needs high memory ressources and it can be slow. For this reason, Abdelkhalek et al. used another algorithm, called the *Espresso* algorithm, a heuristic method for minimizing the number of terms in a product-of-sum representation. Espresso is not guaranteed to find the minimum, and it usually doesn't in the case of 8-bit Sboxes, but its solutions are good enough to be used in practice.

The *Efficient subset* problem. No matter the method used for solving the *Inequality generation* problem, one must choose among the initial set of inequalities a good representative set for representing the support of the Boolean function. This is what we called the *Efficient subset* problem. As mentioned in [ST17a], determining how many and which inequalities to keep is not an evident decision. This step is however necessary, as in both methods from [Sun+14b; Sun+14a] and all the new methods that we are going to present, the number of generated inequalities is high and has an important impact on the optimization time. For example, in the convex-hull method, the number of linear inequalities that `Sage` returns is typically quite high, containing notably many redundant ones. The authors of [Sun+14a] applied then a greedy algorithm for solving the *Efficient subset* problem. At each step, this algorithm adds to the solution set the best possible inequality, that is the inequality removing the highest number of points among those that have not been removed yet. A nice approach for solving this step was later given by Sasaki and Todo in [ST17a]. They proposed to model the problem of minimizing the set of inequalities that remove all the impossible propagation points as a MILP problem itself and solve it by some solver. More precisely, their method consists in assigning a binary variable z_i to each inequality found by solving the *Inequality*

generation problem. Then for each impossible point $\boldsymbol{a}$, it is required that at least one inequality removing $\boldsymbol{a}$ is chosen with

$$\sum_{i \text{ s.t. ineq. } i \text{ removes } \boldsymbol{a}} z_i \geqslant 1.$$

Finally the MILP solver is used for minimizing $\sum_i z_i$ and $\{\, i \mid z_i = 1 \,\}$ gives a solution of the *Efficient subset* problem.

***Efficient subset*: greedy or minimum?** It appears however in [ST17a] that the smallest subset of inequalities found by this approach that correctly models a DDT, will not necessarily provide the overall best performance when running a complete cipher modeling. Moreover, this auxiliary MILP problem can be too heavy when the initial set of inequalities is large. In our experiments, we have found the greedy approach from [Sun+14a] to provide better subsets for performance even if they are a bit larger. We hence used it in the applications. However, we consider the minimization approach from [ST17a] in solving the *Efficient subset* problem to be a good benchmarking indicator for methods solving the *Inequality generation* problem.

Focusing on *Inequality generation* with minimum. In the remaining part of the section we analyze in depth the problem of efficiently modeling a Boolean function with MILP inequalities and we concentrate on the *Inequality generation* problem. Indeed, our goal is to generate efficient algorithms for modeling a Boolean function by using the smallest number of inequalities as an indicator for the quality of the method.

We propose different methods for doing so. The first method, based on generating better inequalities from the H-representation is applicable for up to 12-bit Boolean functions and gives better results than all previous methods for up to 6-bit Sboxes. Unfortunately, this method is not applicable for 8-bit Sboxes. For this reason, we develop other methods for larger Boolean functions and Sboxes. With these methods, described in Sections 4.2.4 and 4.2.5 we manage to model 8-bit Sboxes with a much smaller number of inequalities than what was done before.

In the remaining part of this section, and to facilitate comprehension, all methods will be described through the DDT modeling application. Nevertheless, none of these techniques is DDT-specific and they can be applied directly for modeling any Boolean function.

4.2.3 Inequalities derived from the convex hull

When the computation of the H-representation of the convex hull of all possible transitions in a DDT is computationally feasible, this H-representation provides by definition a set of inequalities modeling the possible differential transitions through the `Sbox`. However, as we will show, it is possible to compute many other linear inequalities from this initial set by simply adding up some of them. The reason for doing so is to generate potentially better inequalities than the ones directly given by the convex hull, where better means that the new inequalities remove more impossible transitions than the initial ones do.

Indeed, if a possible differential transition $\boldsymbol{z} = (\boldsymbol{x} \mid \boldsymbol{y}) \in \{0,1\}^m$, with $m = 2n$ satisfies the k inequalities $C_1, \ldots, C_k$, e.g.

$$c_0^\ell z_0 + \cdots + c_{m-1}^\ell z_{m-1} + b_\ell \geqslant 0 \quad \text{with} \quad \ell \in [1, k],$$

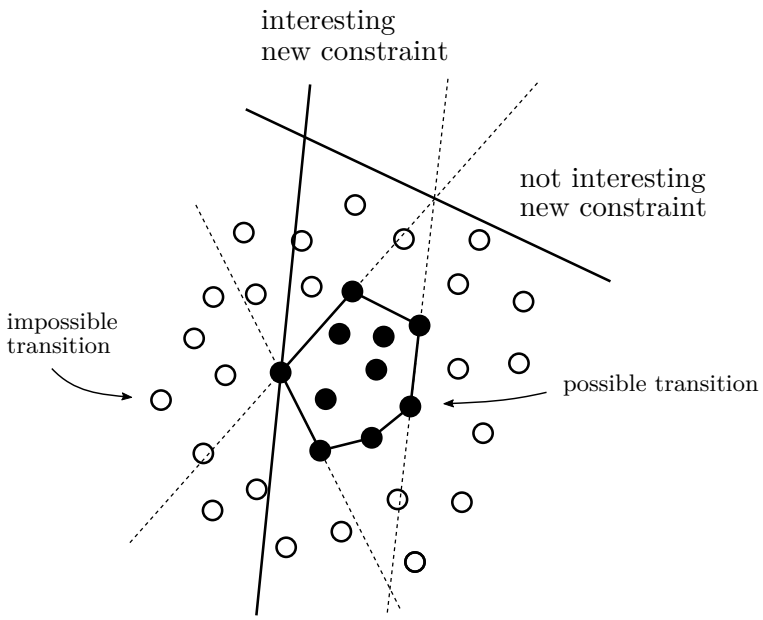

Figure 4.3 – Example of the convex hull of a set of possible transitions. Since in practice transitions only lie on the hypercube $\{0,1\}^m$, this figure is just a sketch to give intuition about our method.

then it obviously also satisfies the inequality

$$\left(\sum_{i=1}^{k} c_0^i\right) z_0 + \cdots + \left(\sum_{i=1}^{k} c_{m-1}^i\right) z_{m-1} + \sum_{i=1}^{k} b_i \geqslant 0$$

produced by simply summing up the initial k inequalities and denoted in the sequel as $C_{\text{new}} = C_1 + \ldots + C_k$.

Of course, most of the inequalities produced by randomly summing k inequalities from the H-representation of the convex hull, do not present any interest, as they will very probably be satisfied by the whole space $\{0,1\}^m$. In order to produce meaningful new linear inequalities from the H-representation of the convex hull, we noticed that if k hyperplanes of the H-representation share a vertex on the cube $\{0,1\}^m$, (i.e. a possible transition), then the addition of the k corresponding inequalities will probably yield an interesting new constraint, given that its hyperplane intersects with the cube at least on this particular vertex. By "interesting" we mean here that the new inequality C_{new} will remove a different (potentially larger) set of impossible transitions than the original inequalities. This idea is illustrated in Figure 4.3 and described by Algorithm 6.

Algorithm 6 takes as input a set $\mathcal{S}_{\text{pos}}$ corresponding to the possible transitions through an `Sbox` and a parameter k that indicates the number of inequalities to be added together each time. Then it starts by generating the convex hull corresponding to $\mathcal{S}_{\text{pos}}$ by using for example the `inequality_generator()` function of the `sage.geometry.polyhedron` class of the `Sage` computer algebra system [The20]. This gives us an initial set of inequalities $\mathcal{C}_{\text{set}}$ of the form

$$c_0 x_0 + \cdots + c_{m-1} x_{m-1} + b_\ell \geqslant 0.$$

Then, for every point $\boldsymbol{a}$ in $\mathcal{S}_{\text{pos}}$ we add together any k inequalities $C_1, \ldots, C_k$ that this point satisfies with a zero left-hand side, i.e. p belongs to the hyperplanes defined by those inequalities. We add in

Algorithm 6 Compute a set of inequalities from possible transitions.

input
  $\mathcal{S}_{\text{pos}}$ a set of possible points.
  k the number of constraints to sum to get a new constraint.
output
  $\mathcal{C}_{\text{set}}$, a set of $\mathbb{R}$-linear constraints.

$\mathcal{H} \leftarrow \text{Hull}(\mathcal{S}_{\text{pos}})$
$\mathcal{C}_{\text{set}} \leftarrow \mathcal{H}$
for all $\boldsymbol{p} \in \mathcal{S}_{\text{pos}}$ **do**
  for all $\{C_1, \ldots, C_k\} \in \mathcal{P}(\mathcal{H})$ such that $\boldsymbol{p}$ belongs to the hyperplanes of $C_1, \ldots, C_k$ **do**
    $C_{\text{new}} = C_1 + \ldots + C_k$
    if C_{new} removes a new set of impossible transitions **then**
      $\mathcal{C}_{\text{set}} \leftarrow \mathcal{C}_{\text{set}} \cup \{C_{\text{new}}\}$
return $\mathcal{C}_{\text{set}}$

$\mathcal{C}_{\text{set}}$ the new inequality only if it is meaningful, in the sense that it removes a new set of impossible transitions.

In practice, Algorithm 6 is rather fast for 4-bit `Sbox`es — a few minutes for $k = 2$ and a few hours at most for $k = 3$ — and the set of constraints $\mathcal{C}_{\text{set}}$ obtained that way does not have too many elements, which allows fast minimization in solving the *Efficient subset* problem. For example, `Sage` returns 327 linear inequalities for the `Sbox` of `PRESENT`. By applying Algorithm 6 with $k = 2$ we get a little bit less than 500 inequalities, and for $k = 3$ we get a little bit less than 700 inequalities.

Application to DDTs of 4-bit `Sbox`es

We applied Algorithm 6 to solving the *Inequality generation* problem for different `Sbox`es from the literature and benchmarked it by solving the *Efficient subset* problem with the method of [ST17a] to obtain a minimal modeling. The results are summarized in Table 4.2. We took $k = 2$ to run Algorithm 6, apart for `TWINE`, `PRIDE`, `Serpent S3` and `Serpent S7` where taking $k = 3$ gave slightly better results. The case $k = 1$, corresponds to the method of [ST17a]. Even as already said, the exact minimum is not necessarily what one has to take to minimize the solving time of the global problem, it is however a good indicator of the quality of the method used to solve the *Inequality generation* problem and permits to compare the different methods between them. We nonetheless also give the results with the greedy approach from [Sun+14b] and $k = 1$ for completeness. As can be seen from Table 4.2 but also for all the `Sbox`es we tested for $n \leqslant 6$, running the algorithm with $k > 1$ always gave better results than with $k = 1$.

Unfortunately, for larger `Sbox`es, and notably for $n = 8$ computing the convex hull is computationally hard. For this reason, we describe in the following sections, alternative methods that can be used for modeling 8-bit `Sbox`es.

4.2.4 Inequalities to remove logical sets $\boldsymbol{a} \oplus \text{Prec}(\boldsymbol{u})$

In this first section, we show that one can easily derive simple inequalities to remove spaces of the form $\boldsymbol{a} \oplus \text{Prec}(\boldsymbol{u})$ inside the DDT. Let again $m = 2n$ where n is the bit-size of the Sbox.

Table 4.2 – Number of inequalities to model differential transitions for various 4-bit Sboxes

	# Inequalities		
Sbox	[Sun+14b]	[ST17a]	Alg. 6
PRESENT	22	21	**17**
KLEIN	22	21	**19**
TWINE	23	23	**19**
PRINCE	26	22	**19**
Piccolo	23	21	**16**
MIBS	27	23	**20**
LBlock S0	28	24	**17**
LBlock S1	27	24	**17**
LBlock S2	27	24	**17**
LBlock S3	27	24	**17**
LBlock S4	28	24	**17**
LBlock S5	27	24	**17**
LBlock S6	27	24	**17**
LBlock S7	27	24	**17**
LBlock S8	28	24	**17**
LBlock S9	27	24	**17**

	# Inequalities		
Sbox	[Sun+14b]	[ST17a]	Alg. 6
Serpent S0	23	21	**17**
Serpent S1	24	21	**17**
Serpent S2	25	21	**18**
Serpent S3	31	27	**20**
Serpent S4	26	23	**19**
Serpent S5	25	23	**19**
Serpent S6	22	21	**17**
Serpent S7	30	27	**20**
Lilliput	–	23	**21**
Minalpher	–	22	**19**
Midori S0	–	21	**16**
Midori S1	–	22	**20**
RECTANGLE	–	21	**17**
SKINNY	–	21	**16**
GIFT	–	–	**17**
PRIDE	–	–	**16**

Notation 4.1. $\mathrm{Prec}\,(\boldsymbol{u})$ denotes the linear space

$$\mathrm{Prec}\,(\boldsymbol{u}) \stackrel{\text{def}}{=} \{\, \boldsymbol{x} \in \mathbb{F}_2^m \mid \boldsymbol{x} \preceq \boldsymbol{u} \,\},$$

where $\boldsymbol{x} \preceq \boldsymbol{u}$ means that $x_i \leqslant u_i$ for all $i \in [0, m-1]$.

Proposition 4.1. *Let* $\boldsymbol{a} \in \mathbb{F}_2^m$ *and* $\boldsymbol{u} \in \mathbb{F}_2^m$ *such that* $\mathrm{supp}\,(\boldsymbol{a}) \bigcap \mathrm{supp}\,(\boldsymbol{u}) = \emptyset$ *and let* $I = [0, m-1] \setminus (\mathrm{supp}\,(\boldsymbol{a}) \bigcup \mathrm{supp}\,(\boldsymbol{u}))$. *Then, for all* $x \in \mathbb{F}_2^m$,

$$-\sum_{i \in \mathrm{supp}(\boldsymbol{a})} x_i + \sum_{i \in I} x_i \geqslant 1 - \mathrm{wt}\,(\boldsymbol{a}) \iff \boldsymbol{x} \notin \boldsymbol{a} \oplus \mathrm{Prec}\,(\boldsymbol{u}).$$

Proof. Let $\boldsymbol{x} \in \mathbb{F}_2^m$. If $\boldsymbol{x} \in \boldsymbol{a} \oplus \mathrm{Prec}\,(\boldsymbol{u})$,

$$-\sum_{i \in \mathrm{supp}(\boldsymbol{a})} x_i + \sum_{i \in I} x_i = -\sum_{i \in \mathrm{supp}(\boldsymbol{a})} a_i = -\mathrm{wt}\,(\boldsymbol{a}).$$

Otherwise $\boldsymbol{x} \oplus \boldsymbol{a} \not\preceq \boldsymbol{u}$, so there exists some $\ell \in \mathrm{supp}\,(\boldsymbol{a}) \cup I$ such that $x_\ell = 1 - a_\ell$.

- If $\ell \in \mathrm{supp}\,(\boldsymbol{a})$ then $x_\ell = 0$ and $\sum_{i \in \mathrm{supp}(\boldsymbol{a})} x_i \leqslant \mathrm{wt}\,(\boldsymbol{a}) - 1$.
- If $\ell \in I$, then $x_\ell = 1$ and $\sum_{i \in I} x_i \geqslant 1$.

In both cases,

$$-\sum_{i \in \mathrm{supp}(\boldsymbol{a})} x_i + \sum_{i \in I} x_i \geqslant 1 - \mathrm{wt}\,(\boldsymbol{a}).$$

□

Example 4.5. We show how the above method can be applied to remove some invalid transitions for the Sbox of the block cipher `PRESENT` (see Section 1.4.2). The Sbox used in `PRESENT` is given in Table 1.1 and its DDT is given in Section A.1.

Here $m = 2 \times 4 = 8$. For better visualizing points in the DDT, we will see $\mathbb{F}_2^8$ as $\mathbb{F}_2^4 \times \mathbb{F}_2^4$. Further, we will use the following bit ordering. For a point $(\boldsymbol{x}, \boldsymbol{y}) \in \mathbb{F}_2^4 \times \mathbb{F}_2^4$, the index 0 will correspond to the least significant bit (LSB) of $\boldsymbol{x}$, 3 will correspond to the most significant bit (MSB) of $\boldsymbol{x}$, 4 to the LSB of $\boldsymbol{y}$ and 7 to the MSB of $\boldsymbol{y}$. Let $\boldsymbol{a} = [\texttt{0,1}], \boldsymbol{u} = [\texttt{9,4}] \in \mathbb{F}_2^4 \times \mathbb{F}_2^4$. Then

$$\mathrm{Prec}\,(\boldsymbol{u}) = \{[0,0],[0,4],[1,0],[1,4],[8,0],[8,4],[9,0],[9,4]\}\,.$$

Further, as $\mathrm{supp}\,(\boldsymbol{a}) = \{4\}$ and $\mathrm{supp}\,(\boldsymbol{u}) = \{0,3,6\}$, $I = \{1,2,5,7\}$. Therefore the equation

$$-x_4 + x_1 + x_2 + x_5 + x_7 \geqslant 0$$

removes exactly the 8 points in the space

$$\boldsymbol{a} \oplus \mathrm{Prec}\,(\boldsymbol{u}) = \{[0,1],[0,5],[1,1],[1,5],[8,1],[8,5],[9,1],[9,5]\}\,.$$

We can verify from the DDT in Section A.1 that all these points correspond indeed to invalid transitions through the DDT.

Algorithmic aspects

Given a set of impossible transitions $\mathcal{I}$, Algorithm 7 finds all subsets of the form $\boldsymbol{a} \oplus \mathrm{Prec}\,(\boldsymbol{u})$ excluding those that are subsets of others. For each $\boldsymbol{a} \in \mathcal{I}$, it builds spaces $\boldsymbol{a} \oplus \mathrm{Prec}\,(\boldsymbol{u}) \subseteq \mathcal{I}$ by progressively incrementing the weight of $\boldsymbol{u}$, with $\boldsymbol{u}$ such that $\boldsymbol{a} \oplus \boldsymbol{u} \in \mathcal{I}$ and $\mathrm{supp}\,(\boldsymbol{u}) \cap \mathrm{supp}\,(\boldsymbol{a}) = \emptyset$ and checking for all $\boldsymbol{v} \preceq \boldsymbol{u}, \mathrm{wt}\,(\boldsymbol{v}) = \mathrm{wt}\,(\boldsymbol{u}) - 1$ whether $\boldsymbol{a} \oplus \mathrm{Prec}\,(\boldsymbol{v})$ has already been identified as a subset of $\mathcal{I}$.

We show next that Algorithm 7 and the Quine-McCluskey algorithm are strongly related. The Quine-McCluskey algorithm has two steps. Given a set of points $\mathcal{I}$, the first step finds a set $\mathcal{S}$ of subspaces $\boldsymbol{a} \oplus \mathrm{Prec}\,(\boldsymbol{u}) \subseteq \{0,1\}^m$ such that

$$\mathcal{I} = \bigcup_{\boldsymbol{a} \oplus \mathrm{Prec}(\boldsymbol{u}) \in \mathcal{S}} \boldsymbol{a} \oplus \mathrm{Prec}\,(\boldsymbol{u})\,, \quad \forall\,(\boldsymbol{a} \oplus \mathrm{Prec}\,(\boldsymbol{u})) \subseteq \mathcal{I}, \exists \mathcal{E} \in \mathcal{S} :\ \boldsymbol{a} \oplus \mathrm{Prec}\,(\boldsymbol{u}) \subseteq \mathcal{E},$$
$$\text{and } \forall \mathcal{E}, \mathcal{F} \in \mathcal{S}, \mathcal{E} \not\subseteq \mathcal{F}. \quad (4.2)$$

This is exactly what does Algorithm 7. The second step of QM then searches for a minimal set $\mathcal{S}' \subseteq \mathcal{S}$, in the sense that:

$$\mathcal{I} = \bigcup_{\boldsymbol{a} \oplus \mathrm{Prec}(\boldsymbol{u}) \in \mathcal{S}'} \boldsymbol{a} \oplus \mathrm{Prec}\,(\boldsymbol{u}) \quad \text{and} \quad \forall \mathcal{E}, \mathcal{F}, \mathcal{G} \in \mathcal{S}', \mathcal{E} \not\subseteq \mathcal{F} \cup \mathcal{G}.$$

This formulation of the Quine-McCluskey algorithm is very different from the one encountered in the literature and in the best of our knowledge, it is the first time that this algorithm is presented in this way. We use this presentation as it is well-suited for understanding the link between the two methods.

The important thing for us is that we principally need this first step to find good modelings since it corresponds to solving the *Inequality generation* problem. Besides, the second step of the QM

Algorithm 7 Find $\boldsymbol{a} \oplus \mathrm{Prec}(\boldsymbol{u})$ sets included in the set $\mathcal{I} \subset \{0,1\}^m$.

input
  Set of impossible transitions $\mathcal{I} \subset \{0,1\}^m$.
output
  $\mathcal{S}_{\text{out}}$, set of subsets of $\mathcal{I}$ of the form $\boldsymbol{a} \oplus \mathrm{Prec}(\boldsymbol{u})$ satisfying Equation (4.2).

$\mathcal{S}_{\text{out}} \leftarrow \emptyset$
for all $\boldsymbol{a} \in \mathcal{I}$ **do**
  $\mathcal{S}_{\text{int}} \leftarrow \emptyset$ ▷ Set of interesting new inequalities of the form $\boldsymbol{a} \oplus \mathrm{Prec}(\boldsymbol{u})$
  for all $i \in [0, m]$ **do**
    $\mathcal{S}_i \leftarrow \emptyset$
    $\mathcal{U}_i \leftarrow \emptyset$
  ▷ We start by grouping all $\boldsymbol{u}$ such that $\boldsymbol{a} \oplus \mathrm{Prec}(\boldsymbol{u}) \subseteq \mathcal{I}$
  ▷ and $\mathrm{supp}(\boldsymbol{a}) \cap \mathrm{supp}(\boldsymbol{u}) = \emptyset$ by their weight in sets $\mathcal{U}$.
  for all $\boldsymbol{p} \in \mathcal{I}$ **do**
    $\boldsymbol{u} \leftarrow \boldsymbol{a} \oplus \boldsymbol{p}$
    if $\mathrm{supp}(\boldsymbol{a}) \cap \mathrm{supp}(\boldsymbol{u}) = \emptyset$ **then**
      $\mathcal{U}_{\mathrm{wt}(\boldsymbol{u})} \leftarrow \mathcal{U}_{\mathrm{wt}(\boldsymbol{u})} \cup \{u\}$
  ▷ If $\mathcal{U}_1 \neq \emptyset$, $\boldsymbol{a} \oplus \mathrm{Prec}(0)$ is no longer interesting
  ▷ since $\forall \boldsymbol{u} \in \mathcal{U}_1$, $\boldsymbol{a} \in \boldsymbol{a} \oplus \mathrm{Prec}(\boldsymbol{u})$.
  if $\mathcal{U}_1 = \emptyset$ **then**
    $\mathcal{S}_{\text{int}} \leftarrow \{\, \boldsymbol{a} \oplus \mathrm{Prec}(\boldsymbol{u}) \mid \boldsymbol{u} \in \mathcal{U}_0 \,\}$
  else
    $\mathcal{S}_{\text{int}} \leftarrow \{\, \boldsymbol{a} \oplus \mathrm{Prec}(\boldsymbol{u}) \mid \boldsymbol{u} \in \mathcal{U}_1 \,\}$
  ▷ $\boldsymbol{a} \oplus \mathrm{Prec}(\boldsymbol{u}) \in \mathcal{S}_k \iff \mathrm{wt}(\boldsymbol{u}) = k$ and $\boldsymbol{a} \oplus \mathrm{Prec}(\boldsymbol{u}) \subseteq \mathcal{I}$
  $\mathcal{S}_0 \leftarrow \{\, \boldsymbol{a} \oplus \mathrm{Prec}(\boldsymbol{u}) \mid \boldsymbol{u} \in \mathcal{U}_0 \,\}$
  $\mathcal{S}_1 \leftarrow \{\, \boldsymbol{a} \oplus \mathrm{Prec}(\boldsymbol{u}) \mid \boldsymbol{u} \in \mathcal{U}_1 \,\}$
  for $k \in [2, m]$ **do**
    for $\boldsymbol{u} \in \mathcal{U}_k$ **do**
      if $\forall \boldsymbol{v} \preceq \boldsymbol{u}$ st. $\mathrm{wt}(\boldsymbol{v}) = k-1$, $\boldsymbol{a} \oplus \mathrm{Prec}(\boldsymbol{v}) \in \mathcal{S}_{k-1}$ **then**
        $\mathcal{S}_k \leftarrow \mathcal{S}_k \cup \{\boldsymbol{a} \oplus \mathrm{Prec}(\boldsymbol{u})\}$
        for all $v \preceq u$ st. $\mathrm{wt}(\boldsymbol{v}) = k-1$ **do**
          ▷ Remove $\boldsymbol{a} \oplus \mathrm{Prec}(\boldsymbol{v})$ since $\boldsymbol{a} \oplus \mathrm{Prec}(\boldsymbol{v}) \subset \boldsymbol{a} \oplus \mathrm{Prec}(\boldsymbol{u})$
          $\mathcal{S}_{\text{int}} \leftarrow \mathcal{S}_{\text{int}} \setminus \{\boldsymbol{a} \oplus \mathrm{Prec}(\boldsymbol{v})\}$
    $\mathcal{S}_{\text{int}} \leftarrow \mathcal{S}_{\text{int}} \cup \mathcal{S}_k$
  $\mathcal{S}_{\text{out}} \leftarrow \mathcal{S}_{\text{out}} \cup \mathcal{S}_{\text{int}}$
return $\mathcal{S}_{\text{out}}$

Table 4.3 – Number of inequalities to exactly exclude the set of impossible differential transitions $\mathcal{I}$ for two 8-bit Sboxes with the methods of [Abd+17] and Algorithm 7.

		# Inequalities		
		[Abd+17]		Algorithm 7 with
`Sbox`	$\lvert\mathcal{I}\rvert$	QM	Espresso	MILP minimization
AES	33150	–	8302	**7461**
`SKINNY-128`	54067	**372**	376	**372**

algorithm provides a solution for the *Efficient subset* problem by minimizing the number of terms. However, the objective of this second step is to find a good circuit for a Boolean function, but not necessarily a good MILP modeling. Moreover, this second step is computationally harder than the first one and acts as a bottleneck when using QM as a black box inequality generator for MILP modelings. Indeed, it is much faster to use Algorithm 7 alone for solving the *Inequality generation* problem together with a greedy algorithm or a MILP-based algorithm for solving the *Efficient subset* problem. This is not only faster but can also provide a significantly lower number of inequalities.

Application to the AES and `SKINNY-128`

We notably applied Algorithm 7 for generating an initial set of inequalities for the 8-bit Sboxes of AES and `SKINNY-128`. We then obtained 70336 initial inequalities for AES and 8829 for `SKINNY-128`. The running time was of around 15 minutes for AES and 2 hours for `SKINNY-128` where 90% of the time was spent not for finding new inequalities but for removing not interesting ones (lines 25 – 26 in Algorithm 7). The difference in these running times is explained by the fact that as the DDT of `SKINNY-128` is very sparse, there are many more possible spaces $a \oplus \mathrm{Prec}(\boldsymbol{u})$ than for the DDT of AES. After this, to find a representative set among these initial inequalities (i.e. to solve the *Efficient subset* problem) we set up a minimization problem and solved it with `Gurobi`. While for `SKINNY-128` the problem was solved in just a few seconds providing us with the global minimum (which is thus the same as the QM algorithm), for AES the problem didn't reach the minimum even after 22 days of search on an 8-core laptop. However, the solution provided by `Gurobi`, even if not the minimal one is much better than the one given by Espresso, as it can be seen in Table 4.3. Furthermore, according to the authors of [Abd+17], QM itself cannot be applied to AES because of its memory complexity.

While for Sboxes with sparse DDTs, as the one of `SKINNY-128`, the method of this section provides a quite compact modeling, for Sboxes with low differential uniformity (i.e. highest value in the DDT), as the one of AES the number of obtained inequalities is quite high for practical applications. For this reason, we provide in the following sections new methods for modeling large Sboxes that outperform in most of the cases the methods provided up to now.

4.2.5 Inequalities to remove Hamming balls $\mathcal{B}(d, \boldsymbol{c})$

We saw in the previous section that each set of impossible transitions of the form $\boldsymbol{a} \oplus \mathrm{Prec}(\boldsymbol{u})$ provided a single inequality for removing all points in the set. What we learned in particular from this is that it is interesting to group impossible transitions in sets having a particular algebraic

description and get inequalities out of them. In this section, we investigate a different type of sets and show how to use their mathematical description to get nice inequalities. More precisely, we show that points lying in a ball $\mathcal{B}(d, \boldsymbol{c})$ of radius d centered at a point $\boldsymbol{c} \in \mathbb{F}_2^m$, can be removed together by a simple inequality.

Definition 4.1 (Hamming ball)**.** A *ball* of $\mathbb{F}_2^m$ of radius d centered at $\boldsymbol{c} \in \mathbb{F}_2^m$ is the subset of all points whose Hamming distance from the center $\boldsymbol{c}$ is at most d:

$$\mathcal{B}(d, \boldsymbol{c}) \stackrel{\text{def}}{=} \{\, \boldsymbol{x} \in \mathbb{F}_2^m \mid \mathrm{wt}\,(\boldsymbol{x} \oplus \boldsymbol{c}) \leqslant d \,\}.$$

Furthermore, by $\mathcal{S}(d, \boldsymbol{c})$ we will denote the *sphere* of radius d centered on $\boldsymbol{c}$, that is the set of points $\boldsymbol{x} \in \mathcal{B}(d, \boldsymbol{c})$ for which $\mathrm{wt}\,(\boldsymbol{x} \oplus \boldsymbol{c}) = d$.

To illustrate this idea, we start with an example of the simplest case — balls of radius 1.

Example 4.6. Consider $m = 4$ and let $\mathcal{B}(1, \boldsymbol{c})$ be a ball of radius 1 centered at $\boldsymbol{c} = (1, 0, 0, 0) \in \mathbb{F}_2^4$: $\mathcal{B}(1, \boldsymbol{c}) = \{(1, 0, 0, 0), (0, 0, 0, 0), (1, 1, 0, 0), (1, 0, 1, 0), (1, 0, 0, 1)\}$. It can be checked that all five points of the above ball can be removed by

$$(1 - x_0) + x_1 + x_2 + x_3 \geqslant 2.$$

We can construct similar examples for any ball of dimension $d > 1$. As we will show now, for any dimension m and any point $\boldsymbol{c} \in \mathbb{F}_2^m$, it is possible to construct an inequality that removes all points of the ball $\mathcal{B}(d, \boldsymbol{c})$.

Proposition 4.2. *Let $\boldsymbol{c} \in \mathbb{F}_2^m$. Then*

$$\sum_{i=0}^{m-1} (1 - c_i)x_i + c_i(1 - x_i) \geqslant d + 1 \iff \boldsymbol{x} \notin \mathcal{B}(d, \boldsymbol{c}). \tag{4.3}$$

Proof. Notice here that

$$\sum_{i=0}^{m-1} (1 - c_i)x_i + c_i(1 - x_i) = \sum_{i=0}^{m-1} x_i \oplus \boldsymbol{c}_i = \mathrm{wt}\,(\boldsymbol{x} \oplus \boldsymbol{c}).$$

For any point $\boldsymbol{u} \in \mathcal{B}(d, \boldsymbol{c})$, we thus have

$$\sum_{i=0}^{m-1} (1 - c_i)u_i + c_i(1 - u_i) = \mathrm{wt}\,(\boldsymbol{u} \oplus \boldsymbol{c}) \leqslant d.$$

On the other side, for any point $\boldsymbol{u} \in \mathbb{F}_2^m \setminus \mathcal{B}(d, \boldsymbol{c})$, $\mathrm{wt}\,(\boldsymbol{u} \oplus \boldsymbol{c}) > d$. □

Distorted balls

When searching for inequalities removing impossible transitions for a DDT, we have to be sure that the corresponding ball does not contain any possible transitions that we would mistakenly remove. In Sboxes used in practice, notably those with a low differentially uniformity, as the number of non-zero coefficients is usually large, removing entire balls does not usually work, as the number of balls for which all points correspond to impossible transitions is typically very small. While the

above method works well for sparse DDTs as the one of `SKINNY-128`, for the Sbox of `PRESENT`, no plain ball, even of radius 1, can be removed with this method.

We will show now, that we can still extract an inequality from a ball for which we have removed from its edge all possible transition points. We call such a ball *distorted*.

Example 4.7. Consider again Example 4.6. Inequality

$$(1 - x_0) + x_1 + x_2 + x_3 \geqslant 2$$

removed all five points of the ball $\mathcal{B}(1, (1, 0, 0, 0))$. Suppose now that we want to remove all the above points except from $(0, 0, 0, 0)$ and $(1, 0, 1, 0)$. Then, intuitively it is enough to increase a little bit the coefficient a_0 corresponding to point $(0, 0, 0, 0)$, that is the coefficient of $(1 - x_0)$, to be sure that when $x_0 = 0$ then $a_0(1 - x_0) \geqslant 2$. As for the other points of the ball $x_0 = 1$, this change does not have any impact on them. In the same way, we can increase the coefficient a_2 before x_2, to be sure to keep $(1, 0, 1, 0)$. One can check that

$$2(1 - x_0) + x_1 + 2x_2 + x_3 \geqslant 2$$

removes indeed the three remaining points of the ball.

The next proposition formalizes the previous example to distorted balls of dimension d.

Proposition 4.3. *Let $\mathcal{B}(d, \boldsymbol{c}) \subset \mathbb{F}_2^m$ be a ball of radius d from which we remove the set of points*

$$\mathcal{Q} = (\boldsymbol{c} \oplus \mathrm{Prec}\,(\boldsymbol{q})) \cap \mathcal{S}(d, \boldsymbol{c})$$

for some $\boldsymbol{q} \in \mathbb{F}_2^m$. Here $\boldsymbol{p} \in \mathcal{Q}$ represents a possible transition towards the edge of the ball. We define $\boldsymbol{a} \in \mathbb{Q}^m$ such that

$$a_i = \begin{cases} \frac{d+1}{d} & \text{if } q_i = 1, \\ 1 & \text{otherwise.} \end{cases}$$

Then

$$\sum_{i=0}^{m-1} a_i\big[(1 - c_i)x_i + c_i(1 - x_i)\big] \geqslant d + 1 \iff \boldsymbol{x} \notin \mathcal{B}(d, \boldsymbol{c}) \setminus \mathcal{Q}.$$

Proof. First note that since $\boldsymbol{x}, \boldsymbol{c} \in \mathbb{F}_2^m$,

$$a_i\big[(1 - c_i)x_i + c_i(1 - x_i)\big] = a_i(x_i \oplus c_i) \text{ for all } i \in [0, m - 1].$$

- If $\boldsymbol{x} \notin \mathcal{B}(d, \boldsymbol{c})$, then

$$\sum_{i=0}^{m-1} a_i(x_i \oplus c_i) \geqslant \sum_{i=0}^{m-1} x_i \oplus c_i \geqslant \mathrm{wt}\,(\boldsymbol{x} \oplus \boldsymbol{c}) \geqslant d + 1.$$

- If $\boldsymbol{x} \in \mathcal{B}(d - 1, \boldsymbol{c})$, then

$$\sum_{i=0}^{m-1} a_i(x_i \oplus c_i) \leqslant \frac{d+1}{d}(d - 1) \leqslant d.$$

- Let $\boldsymbol{x} \in \mathcal{S}(d, \boldsymbol{c})$.

– If $\boldsymbol{x} \in \mathcal{Q}$ then $\boldsymbol{x} \oplus \boldsymbol{c} \preceq \boldsymbol{q}$ and

$$\sum_{i=0}^{m-1} a_i(x_i \oplus c_i) = \frac{d+1}{d}\mathrm{wt}\left(\boldsymbol{x} \oplus \boldsymbol{c}\right) = d+1.$$

– If $\boldsymbol{x} \notin \mathcal{Q}$ then $\boldsymbol{x} \oplus \boldsymbol{c} \not\preceq \boldsymbol{q}$ and there exists some j such that $x_j \neq c_j$, $q_j = 0$ and $a_j = 1$. Then

$$\begin{aligned}
&\sum_{i=0}^{m-1} a_i(x_i \oplus c_i) = 1 + \sum_{i \neq j} a_i(x_i \oplus c_i) \\
&\leqslant\ 1 + \frac{d+1}{d}\sum_{i \neq j} x_i \oplus c_i = 1 + \frac{d+1}{d}\left(\mathrm{wt}\left(\boldsymbol{x} \oplus \boldsymbol{c}\right) - 1\right) = 1 + \frac{d+1}{d}(d-1) \\
&< d+1.
\end{aligned}$$

□

Remark 4.1. For $d = 1$, Proposition 4.3 implies that for any ball of radius 1 and for any subset $\mathcal{Q}$ of points on its edge, it is possible to create an inequality that removes the points in the ball minus the set $\mathcal{Q}$. For $d > 1$ the situation is a little bit different, as removing some points from the edge can also remove other points that we would like to keep. Despite this, inequalities created this way are usually interesting as they can permit to remove different points together.

Example 4.8. Consider again the example of `PRESENT` and let $\mathcal{B}(1, \boldsymbol{c})$ be the ball centered at $\boldsymbol{c} = [0, 1]$:

$$\mathcal{B}(1, \boldsymbol{c}) = \{[0,1], [0,0], [0,3], [0,5], [0,9], [1,1], [2,1], [4,1], [8,1]\}.$$

By looking at the corresponding DDT (given in Section A.1), one can see that all the points in $\mathcal{B}(1, \boldsymbol{c})$ correspond to impossible transitions, except $[0, 0]$. Then, if $\boldsymbol{q} = [0, 1]$, we have that $\mathcal{Q} = \{[0, 0]\}$ and

$$x_0 + x_1 + x_2 + x_3 + 2(1 - y_0) + y_1 + y_2 + y_3 \geqslant 1$$

removes $\mathcal{B}(1, \boldsymbol{c}) \setminus \mathcal{Q}$.

The inequalities provided up to now can remove points in a single ball. While this provides a simple and fast algorithm by itself by simply going through all balls into a DDT and writing down the corresponding inequalities, in practice we only used this method in combination with a more powerful method that we detail in the next section. This new method permits us to remove points belonging to the union of three different balls of radius 1. We call this process *merging*.

4.2.6 Inequalities to remove merged distorted Hamming balls

This method has similarities with the adding-inequalities method used in Algorithm 6, except that we combine inequalities obtained with the distorted balls approach. We start by computing distorted balls of radius 1 centred on impossible transitions. However, simply adding inequalities obtained from neighbouring distorted balls often results in "bad" inequalities, in the sense that they do not discard many impossible transitions. To get a more efficient method, we found that when slightly changing one of the distorted balls, adding 2 to the right-hand side of the sum of the three inequalities gives a better inequality. We now present in detail how to get such an inequality.

Let $\mathcal{I}$ be the set of all impossible transitions through the Sbox and let $\boldsymbol{a}$, $\boldsymbol{b}$ and $\boldsymbol{c}$ be three distinct points in $\mathcal{I}$ such that

- $\mathrm{wt}(\boldsymbol{a} \oplus \boldsymbol{b}) = \mathrm{wt}(\boldsymbol{a} \oplus \boldsymbol{c}) = 1$,
- $\boldsymbol{b} \neq \boldsymbol{c}$,
- $\boldsymbol{a} \oplus \boldsymbol{b} \oplus \boldsymbol{c} \in \mathcal{I}$.

Let $\mathcal{B}(1, \boldsymbol{a}), \mathcal{B}(1, \boldsymbol{b})$ and $\mathcal{B}(1, \boldsymbol{c})$ be the corresponding balls of radius 1. We denote by

$$\mathcal{P}_{\boldsymbol{a}} = \mathcal{B}(1, \boldsymbol{a}) \setminus \mathcal{I} \subseteq \mathcal{S}(1, \boldsymbol{a}) \qquad \mathcal{P}_{\boldsymbol{b}} = \mathcal{B}(1, \boldsymbol{b}) \setminus \mathcal{I} \subseteq \mathcal{S}(1, \boldsymbol{b})$$
$$\mathcal{P}_{\boldsymbol{c}} = \mathcal{B}(1, \boldsymbol{c}) \setminus \mathcal{I} \subseteq \mathcal{S}(1, \boldsymbol{c})$$

the possible transition points inside each ball. Finally, consider the sets

$$Q_1 = \mathcal{P}_{\boldsymbol{a}} \oplus \boldsymbol{a} \oplus \boldsymbol{b} \subseteq \mathcal{S}(1, \boldsymbol{b}) \qquad Q_2 = \mathcal{P}_{\boldsymbol{a}} \oplus \boldsymbol{a} \oplus \boldsymbol{c} \subseteq \mathcal{S}(1, \boldsymbol{c})$$
$$Q_3 = \mathcal{P}_{\boldsymbol{b}} \oplus \boldsymbol{b} \oplus \boldsymbol{c} \subseteq \mathcal{S}(1, \boldsymbol{c})$$

and let

$$\mathcal{Q} \stackrel{\text{def}}{=} \mathcal{P}_{\boldsymbol{a}} \cup \mathcal{P}_{\boldsymbol{b}} \cup \mathcal{P}_{\boldsymbol{c}} \cup Q_1 \cup Q_2 \cup Q_3.$$

With Proposition 4.3, one can compute the inequalities $A(\boldsymbol{x}) \geqslant 2, B(\boldsymbol{x}) \geqslant 2$ and $C(\boldsymbol{x}) \geqslant 2$ that remove respectively $\mathcal{B}(1, \boldsymbol{a}) \setminus (\mathcal{Q} \cup \{\boldsymbol{c}\}), \mathcal{B}(1, \boldsymbol{b}) \setminus \mathcal{Q}$ and $\mathcal{B}(1, \boldsymbol{c}) \setminus \mathcal{Q}$. With the above notation, we have the following proposition.

Proposition 4.4. *Let $\boldsymbol{x} \in \{0,1\}^m$, then*

$$A(\boldsymbol{x}) + B(\boldsymbol{x}) + C(\boldsymbol{x}) \geqslant 8 \quad \iff \quad \boldsymbol{x} \notin (\mathcal{B}(1, \boldsymbol{a}) \cup \mathcal{B}(1, \boldsymbol{b}) \cup \mathcal{B}(1, \boldsymbol{c})) \setminus \mathcal{Q}.$$

A proof for Proposition 4.4 is provided on Page 116. The method is summarized in Algorithm 8.

The sets $\mathcal{R}_{\boldsymbol{a}}, \mathcal{R}_{\boldsymbol{b}}$ and $\mathcal{R}_{\boldsymbol{c}}$ in Algorithm 8 correspond to the points inside each ball that have to be kept when writing down the equation for the corresponding distorted ball. More precisely, $\mathcal{R}_{\boldsymbol{a}} = \mathcal{P}_{\boldsymbol{a}} \cup \{c\}, \mathcal{R}_{\boldsymbol{b}} = \mathcal{P}_{\boldsymbol{b}} \cup \mathcal{Q}_1$ and $\mathcal{R}_{\boldsymbol{c}} = \mathcal{P}_{\boldsymbol{c}} \cup \mathcal{Q}_2 \cup \mathcal{Q}_3$. It is interesting to note that there is no symmetry between $\boldsymbol{b}$ and $\boldsymbol{c}$: if one changes the roles of $\boldsymbol{b}$ and $\boldsymbol{c}$, then the new inequality created will remove a different subset of points from the three balls, giving thus a different inequality for our collection.

Example 4.9. Let $\boldsymbol{a} = [0, 11]$, $\boldsymbol{b} = [0, 15]$ and $\boldsymbol{c} = [0, 10]$ be three impossible transition points for the Sbox of `PRESENT`.

$$\mathcal{B}(1, [0, 11]) = \{[0, 11], [0, 10], [0, 9], [0, 15], [0, 3], [1, 11], [2, 11], [4, 11], [8, 11]\},$$
$$\mathcal{B}(1, [0, 15]) = \{[0, 15], [0, 14], [0, 13], [0, 11], [0, 7], [1, 15], [2, 15], [4, 15], [8, 15]\}$$
$$\mathcal{B}(1, [0, 10]) = \{[0, 10], [0, 11], [0, 8], [0, 14], [0, 2], [1, 10], [2, 10], [4, 10], [8, 10]\}.$$

Here, $\mathcal{P}_{\boldsymbol{a}} = \{[8, 11]\}$, $\mathcal{P}_{\boldsymbol{b}} = \{[8, 15]\}$ and $\mathcal{P}_{\boldsymbol{c}} = \{[2, 10], [4, 10]\}$. These sets correspond to all possible transitions inside each ball and therefore they should not be removed by the final inequality.

Then we also compute the three sets Q_1, Q_2 and Q_3, each one containing possible transitions but also impossible transition points that will unfortunatly not be discarded by the new inequality. By following notation we get that $\mathcal{Q}_1 = \{[8, 15]\}$, $\mathcal{Q}_2 = \mathcal{Q}_3 = \{[8, 10]\}$. With the set $\mathcal{Q}$, we construct the following distorted balls

$$\mathcal{B}(1, [0, 11]) \setminus (\mathcal{Q} \cup \{[0, 10]\}), \quad \mathcal{B}(1, [0, 15]) \setminus \mathcal{Q} \text{ and } \mathcal{B}(1, [0, 10]) \setminus \mathcal{Q}.$$

Algorithm 8 Create new inequalities from all possible triples of distorted balls.

input
  Set of impossible transitions $\mathcal{I} \subset \{0,1\}^m$.
output
  $\mathcal{C}$, a set of inequalities satisfied by all points in $\{0,1\}^m \setminus \mathcal{I}$.

$\mathcal{C} \leftarrow \emptyset$ ▷ Initialize the set of inequalities.
▷ For all triple of neighbouring distorted Hamming balls...
for all $(\boldsymbol{a}, \boldsymbol{b}, \boldsymbol{c}) \in \mathcal{I}$ s.t. $\mathrm{wt}(\boldsymbol{a} \oplus \boldsymbol{b}) = \mathrm{wt}(\boldsymbol{a} \oplus \boldsymbol{c}) = 1$ and $\boldsymbol{b} \neq \boldsymbol{c}$ **do**
▷ Initialize some sets.
  $\mathcal{P}_{\boldsymbol{a}} \leftarrow \mathcal{B}(1, \boldsymbol{a}) \setminus \mathcal{I}$, $\mathcal{P}_{\boldsymbol{b}} \leftarrow \mathcal{B}(1, \boldsymbol{b}) \setminus \mathcal{I}$
  $\mathcal{R}_{\boldsymbol{a}} \leftarrow \mathcal{P}_{\boldsymbol{a}} \cup \{c\}$, $\mathcal{R}_{\boldsymbol{b}} \leftarrow \mathcal{P}_{\boldsymbol{b}}$, $\mathcal{R}_{\boldsymbol{c}} \leftarrow \mathcal{B}(1, \boldsymbol{c}) \setminus \mathcal{I}$
  for all $p \in \mathcal{P}_{\boldsymbol{a}}$ **do**
    $\mathcal{R}_{\boldsymbol{b}} \leftarrow \mathcal{R}_{\boldsymbol{b}} \cup \{\boldsymbol{p} \oplus \boldsymbol{a} \oplus \boldsymbol{b}\}$
    $\mathcal{R}_{\boldsymbol{c}} \leftarrow \mathcal{R}_{\boldsymbol{c}} \cup \{\boldsymbol{p} \oplus \boldsymbol{a} \oplus \boldsymbol{c}\}$
  for all $\boldsymbol{p} \in \mathcal{P}_{\boldsymbol{b}}$ **do**
    $\mathcal{R}_{\boldsymbol{c}} \leftarrow \mathcal{R}_{\boldsymbol{c}} \cup \{\boldsymbol{p} \oplus \boldsymbol{b} \oplus \boldsymbol{c}\}$

  Let $A(\boldsymbol{x}) \geqslant 2$ removing $\mathcal{B}(1, \boldsymbol{a}) \setminus \mathcal{R}_{\boldsymbol{a}}$ ▷ with Proposition 4.3.
  Let $B(\boldsymbol{x}) \geqslant 2$ removing $\mathcal{B}(1, \boldsymbol{b}) \setminus \mathcal{R}_{\boldsymbol{b}}$.
  Let $C(\boldsymbol{x}) \geqslant 2$ removing $\mathcal{B}(1, \boldsymbol{c}) \setminus \mathcal{R}_{\boldsymbol{c}}$
▷ New and correct inequality with Proposition 4.4.
  $C_{new} \leftarrow A(\boldsymbol{x}) + B(\boldsymbol{x}) + C(\boldsymbol{x}) \geqslant 8$
  $\mathcal{C} \leftarrow \mathcal{C} \cup \{C_{new}\}$

Following the technique of Proposition 4.3 for each of the three distorted balls, we get the following three linear constraints:

$$\begin{aligned} x_0 + x_1 + x_2 + 2x_3 + 2(1 - y_0) + (1 - y_1) + y_2 + (1 - y_3) &\geqslant 2 \\ x_0 + x_1 + x_2 + 2x_3 + (1 - y_0) + (1 - y_1) + (1 - y_2) + (1 - y_3) &\geqslant 2 \\ x_0 + 2x_1 + 2x_2 + 2x_3 + y_0 + (1 - y_1) + y_2 + (1 - y_3) &\geqslant 2 \end{aligned}$$

Directly adding the three inequalities, while mathematically correct, usually yields inequalities that remove a smaller subset of impossible transitions than what we could get. For this reason, we use the following subtlety: we add 2 to the right-hand side of the sum. By doing so we get that

$$3x_0 + 4x_1 + 4x_2 + 6x_3 + 2(1 - y_0) + 3(1 - y_1) + y_2 + 3(1 - y_3) \geqslant 6$$

removes the 17 points in the set

$$\begin{aligned} &(\mathcal{B}(1, \boldsymbol{a}) \cup \mathcal{B}(1, \boldsymbol{b}) \cup \mathcal{B}(1, \boldsymbol{c})) \setminus \{[2, 10], [4, 10], [8, 10], [8, 11], [8, 15]\} \\ &\quad = \{[0, 2], [0, 3], [0, 7], [0, 8], [0, 9], [0, 10], [0, 11], [0, 13], [0, 14][0, 15], [1, 10], [1, 11], [1, 15]\} \\ &\qquad\qquad \cup \{[2, 11], [2, 15], [4, 11], [4, 15]\}. \end{aligned}$$

Table 4.4 – Comparing upper bounds on the number of inequalities needed to model the corresponding Sboxes. The combination of Algorithms 7 and 8 generates an initial set of inequalities.

	# Inequalities	
Sbox	[Abd+17]	Alg. 7 and 8
SKINNY-128	372	**302**
AES	8302	**2882**

Application to the AES and SKINNY-128

We applied Algorithm 8 together with Algorithm 7 and Proposition 4.3 to create a large set of inequalities for the Sboxes of SKINNY-128 and AES and applied a MILP minimization problem to find a small set of inequalities to represent each Sbox. The results can be visualized in Table 4.4. The resulting MILP problem took only a few seconds to terminate for SKINNY-128 while the optimization could not be terminated for AES. Even though, the number of inequalities that we got by stopping the optimization process after a few days of computation provided us with a much lower number than the best previous result. Of course, by pushing the optimization further, it is possible to get even fewer inequalities for AES but one has to remember that obtaining the absolute minimum does not usually lead to the quickest solving time.

Last, we also applied the combination of Algorithms 7 and 8 to all 4-bit Sboxes of Table 4.2 and for all of them we obtained the same number of inequalities as with Algorithm 6.

Proof for Proposition 4.4

We recall that $\boldsymbol{a}, \boldsymbol{b}, \boldsymbol{c}, \boldsymbol{a} \oplus \boldsymbol{b} \oplus \boldsymbol{c} \in \mathcal{I}, \boldsymbol{b} \neq \boldsymbol{c}, \mathrm{wt}\,(\boldsymbol{a} \oplus \boldsymbol{b}) = \mathrm{wt}\,(\boldsymbol{a} \oplus \boldsymbol{c}) = 1$,

$$\begin{aligned} \mathcal{P}_{\boldsymbol{a}} &= \mathcal{B}(1, \boldsymbol{a}) \setminus \mathcal{I} \subseteq \mathcal{S}(1, \boldsymbol{a}), & \mathcal{P}_{\boldsymbol{b}} &= \mathcal{B}(1, \boldsymbol{b}) \setminus \mathcal{I} \subseteq \mathcal{S}(1, \boldsymbol{b}), \\ \mathcal{P}_{\boldsymbol{c}} &= \mathcal{B}(1, \boldsymbol{c}) \setminus \mathcal{I} \subseteq \mathcal{S}(1, \boldsymbol{c}), & \mathcal{Q}_1 &= \mathcal{P}_{\boldsymbol{a}} \oplus \boldsymbol{a} \oplus \boldsymbol{b} \subseteq \mathcal{S}(1, \boldsymbol{b}), \\ \mathcal{Q}_2 &= \mathcal{P}_{\boldsymbol{a}} \oplus \boldsymbol{a} \oplus \boldsymbol{c} \subseteq \mathcal{S}(1, \boldsymbol{c}), & \mathcal{Q}_3 &= \mathcal{P}_{\boldsymbol{b}} \oplus \boldsymbol{b} \oplus \boldsymbol{c} \subseteq \mathcal{S}(1, \boldsymbol{c}), \end{aligned}$$

$A(\boldsymbol{x}) \geqslant 2, B(\boldsymbol{x}) \geqslant 2$ and $C(\boldsymbol{x}) \geqslant 2$ remove respectively $\mathcal{B}(1, \boldsymbol{a}) \setminus (\mathcal{Q} \cup \{c\}), \mathcal{B}(1, \boldsymbol{b}) \setminus \mathcal{Q}$ and $\mathcal{B}(1, \boldsymbol{c}) \setminus \mathcal{Q}$ and we introduce the notation

$$A(\boldsymbol{x}) = \sum_{i=0}^{m-1} \alpha_i (x_i \oplus a_i), \qquad B(\boldsymbol{x}) = \sum_{i=0}^{m-1} \beta_i (x_i \oplus b_i), \qquad C(\boldsymbol{x}) = \sum_{i=0}^{m-1} \gamma_i (x_i \oplus c_i).$$

We also denote the basis vectors as

$$\boldsymbol{e}_i = (0, \ldots, 0, \overset{i}{\underset{\downarrow}{1}}, 0 \ldots, 0)$$

for all $i \in [0, m-1]$. Finally, $j_{\boldsymbol{b}}$ and $j_{\boldsymbol{c}}$ will denote the indices such that $\boldsymbol{a} \oplus \boldsymbol{b} = \boldsymbol{e}_{j_{\boldsymbol{b}}}$ and $\boldsymbol{a} \oplus \boldsymbol{c} = \boldsymbol{e}_{j_{\boldsymbol{c}}}$.

We then have the following two lemmas:

Lemma 4.1. *The points $\boldsymbol{a}, \boldsymbol{b}, \boldsymbol{c}$ and $\boldsymbol{a} \oplus \boldsymbol{b} \oplus \boldsymbol{c}$ do not belong to $\mathcal{Q}$.*

Proof. First, since those points are impossible transitions, they cannot belong to $\mathcal{P}_a \cup \mathcal{P}_{\boldsymbol{b}} \cup \mathcal{P}_{\boldsymbol{c}}$ by definition. Then if $\boldsymbol{a} \in \mathcal{Q}_1$, then $\boldsymbol{b} \in \mathcal{P}_a$ which as said above is impossible. For all the other cases we have:

$$\begin{aligned}
&\boldsymbol{a} \in \mathcal{Q}_2 \Rightarrow \boldsymbol{c} \in \mathcal{P}_a, && \boldsymbol{a} \in \mathcal{Q}_3 \Rightarrow \boldsymbol{a} \oplus \boldsymbol{b} \oplus \boldsymbol{c} \in \mathcal{P}_{\boldsymbol{b}},\\
&\boldsymbol{b} \in \mathcal{Q}_2 \Rightarrow \boldsymbol{a} \oplus \boldsymbol{b} \oplus \boldsymbol{c} \in \mathcal{P}_a, && \boldsymbol{b} \in \mathcal{Q}_3 \Rightarrow \boldsymbol{c} \in \mathcal{P}_{\boldsymbol{b}},\\
&\boldsymbol{c} \in \mathcal{Q}_1 \Rightarrow \boldsymbol{a} \oplus \boldsymbol{b} \oplus \boldsymbol{c} \in \mathcal{P}_a.
\end{aligned}$$

□

Lemma 4.2. *It holds that* $\alpha_{j_{\boldsymbol{c}}} = 2$ *and* $\alpha_{j_{\boldsymbol{b}}} = \beta_{j_{\boldsymbol{b}}} = \beta_{j_{\boldsymbol{c}}} = \gamma_{j_{\boldsymbol{b}}} = \gamma_{j_{\boldsymbol{c}}} = 1$.

Proof. From Proposition 4.3 in the case $d = 1$, we have for all $i \in [0, m-1]$:

- $\alpha_i = 1 + \left((c_i \oplus a_i) \vee \bigvee_{\boldsymbol{q} \in \mathcal{Q} \cap \mathcal{B}(1,\boldsymbol{a})} q_i \oplus a_i \right)$ because $A(\boldsymbol{x}) \geqslant 2$ removes $\mathcal{B}(1, \boldsymbol{a}) \setminus (\mathcal{Q} \cup \{\boldsymbol{c}\})$,
- $\beta_i = 1 + \bigvee_{\boldsymbol{q} \in \mathcal{Q} \cap \mathcal{B}(1,\boldsymbol{b})} q_i \oplus b_i$,
- $\gamma_i = 1 + \bigvee_{\boldsymbol{q} \in \mathcal{Q} \cap \mathcal{B}(1,\boldsymbol{c})} q_i \oplus c_i$.

Since by definition, $\boldsymbol{a} \oplus \boldsymbol{c} = \boldsymbol{e}_{j_{\boldsymbol{c}}}$, $\alpha_{j_{\boldsymbol{c}}} = 2$. If $\alpha_{j_{\boldsymbol{b}}} = 2$, $\exists \boldsymbol{q} \in \mathcal{Q} \cap \mathcal{B}(1, \boldsymbol{a}) : q_{j_{\boldsymbol{b}}} \neq a_{j_{\boldsymbol{b}}}$, i.e. $\boldsymbol{q} = \boldsymbol{a} \oplus \boldsymbol{e}_{j_{\boldsymbol{b}}} = b$ but $b \notin \mathcal{Q}$. Hence $\alpha_{j_{\boldsymbol{b}}} = 1$. In the same way,

$$\begin{aligned}
&\beta_{j_{\boldsymbol{b}}} = 2 \iff \boldsymbol{b} \oplus \boldsymbol{e}_{j_{\boldsymbol{b}}} = \boldsymbol{a} \in \mathcal{Q}, && \beta_{j_{\boldsymbol{c}}} = 2 \iff \boldsymbol{b} \oplus \boldsymbol{e}_{j_{\boldsymbol{c}}} = \boldsymbol{a} \oplus \boldsymbol{b} \oplus \boldsymbol{c} \in \mathcal{Q},\\
&\gamma_{j_{\boldsymbol{b}}} = 2 \iff \boldsymbol{c} \oplus \boldsymbol{e}_{j_{\boldsymbol{b}}} = \boldsymbol{a} \oplus \boldsymbol{b} \oplus \boldsymbol{c} \in \mathcal{Q} && \text{and } \gamma_{j_{\boldsymbol{c}}} = 2 \iff \boldsymbol{c} \oplus \boldsymbol{e}_{j_{\boldsymbol{c}}} = \boldsymbol{a} \in \mathcal{Q}.
\end{aligned}$$

□

We are ready now to give a proof of Proposition 4.4.

Proof. Let $\boldsymbol{x} \notin \mathcal{B}(1, \boldsymbol{a}) \cup \mathcal{B}(1, \boldsymbol{b}) \cup \mathcal{B}(1, \boldsymbol{c})$.

- If $x_{j_{\boldsymbol{b}}} = a_{j_{\boldsymbol{b}}}$ and $x_{j_{\boldsymbol{c}}} = a_{j_{\boldsymbol{c}}}$. Then $\boldsymbol{x} \notin \mathcal{B}(1, \boldsymbol{a}) \Rightarrow \exists i_1, i_2 \notin \{j_{\boldsymbol{b}}, j_{\boldsymbol{c}}\} : x_{i_1} \neq a_{i_1}$ and $x_{i_2} \neq a_{i_2}$. Hence $A(\boldsymbol{x}) \geqslant \alpha_{i_1} + \alpha_{i_2} \geqslant 2$, $B(\boldsymbol{x}) \geqslant \beta_{i_1} + \beta_{i_2} + \beta_{j_{\boldsymbol{b}}} \geqslant 3$ and $C(\boldsymbol{x}) \geqslant \gamma_{i_1} + \gamma_{i_2} + \gamma_{j_{\boldsymbol{c}}} \geqslant 3$.
- If $x_{j_{\boldsymbol{b}}} = a_{j_{\boldsymbol{b}}}$ and $x_{j_{\boldsymbol{c}}} \neq a_{j_{\boldsymbol{c}}}$. Then $x_{j_{\boldsymbol{b}}} = c_{j_{\boldsymbol{b}}}$ and $x_{j_{\boldsymbol{c}}} = c_{j_{\boldsymbol{c}}}$ and $\boldsymbol{x} \notin \mathcal{B}(1, \boldsymbol{c}) \Rightarrow \exists i_1, i_2 \notin \{j_{\boldsymbol{b}}, j_{\boldsymbol{c}}\} : x_{i_1} \neq a_{i_1}$ and $x_{i_2} \neq a_{i_2}$. Hence $A(\boldsymbol{x}) \geqslant \alpha_{i_1} + \alpha_{i_2} + \alpha_{j_{\boldsymbol{c}}} \geqslant 4$, $B(\boldsymbol{x}) \geqslant \beta_{i_1} + \beta_{i_2} + \beta_{j_{\boldsymbol{b}}} + \beta_{j_{\boldsymbol{c}}} \geqslant 4$ and $C(\boldsymbol{x}) \geqslant \gamma_{i_1} + \gamma_{i_2} \geqslant 2$.
- If $x_{j_{\boldsymbol{b}}} \neq a_{j_{\boldsymbol{b}}}$ and $x_{j_{\boldsymbol{c}}} = a_{j_{\boldsymbol{c}}}$. Then $x_{j_{\boldsymbol{b}}} = b_{j_{\boldsymbol{b}}}$ and $x_{j_{\boldsymbol{c}}} = b_{j_{\boldsymbol{c}}}$ and $\boldsymbol{x} \notin \mathcal{B}(1, \boldsymbol{b}) \Rightarrow \exists i_1, i_2 \notin \{j_{\boldsymbol{b}}, j_{\boldsymbol{c}}\} : x_{i_1} \neq a_{i_1}$ and $x_{i_2} \neq a_{i_2}$. Hence $A(\boldsymbol{x}) \geqslant \alpha_{i_1} + \alpha_{i_2} + \alpha_{j_{\boldsymbol{b}}} \geqslant 3$, $B(\boldsymbol{x}) \geqslant \beta_{i_1} + \beta_{i_2} \geqslant 2$ and $C(\boldsymbol{x}) \geqslant \gamma_{i_1} + \gamma_{i_2} + \gamma_{j_{\boldsymbol{b}}} + \gamma_{j_{\boldsymbol{c}}} \geqslant 4$.
- If $x_{j_{\boldsymbol{b}}} \neq a_{j_{\boldsymbol{b}}}$ and $x_{j_{\boldsymbol{c}}} \neq a_{j_{\boldsymbol{c}}}$. Then $\boldsymbol{x} \notin \mathcal{B}(1, \boldsymbol{b}) \cup \mathcal{B}(1, \boldsymbol{c}) \Rightarrow \exists i_1, i_2 \notin \{j_{\boldsymbol{b}}, j_{\boldsymbol{c}}\} : x_{i_1} \neq a_{i_1}$ and $x_{i_2} \neq a_{i_2}$.
  - If $i_{\boldsymbol{b}} \neq i_{\boldsymbol{c}}$, $A(\boldsymbol{x}) \geqslant 4$, $B(\boldsymbol{x}) \geqslant 3$ and $C(\boldsymbol{x}) \geqslant 3$.

– Otherwise, $A(\boldsymbol{x}) = \alpha_{i_{\boldsymbol{b}}} + \alpha_{i_{\boldsymbol{c}}} + \alpha_{j_{\boldsymbol{c}}} \geqslant 4$, $B(\boldsymbol{x}) = \beta_{i_{\boldsymbol{b}}} + \beta_{j_{\boldsymbol{c}}} \geqslant 2$ and $C(\boldsymbol{x}) = \gamma_{i_{\boldsymbol{b}}} + \gamma_{j_{\boldsymbol{b}}} \geqslant 2$.

Therefore, in all the above cases the inequality $A(\boldsymbol{x}) + B(\boldsymbol{x}) + C(\boldsymbol{x}) \geqslant 8$ is verified.

Let now $\boldsymbol{x} \in (\mathcal{B}(1, \boldsymbol{a}) \cup \mathcal{B}(1, \boldsymbol{b}) \cup \mathcal{B}(1, \boldsymbol{c})) \cap \mathcal{Q}$.

- If $\boldsymbol{x} \in \mathcal{B}(1, \boldsymbol{a})$, since $\boldsymbol{a}, \boldsymbol{b}, \boldsymbol{c} \notin \mathcal{Q}$, $\exists i \notin \{j_{\boldsymbol{b}}, j_{\boldsymbol{c}}\} : \boldsymbol{x} = \boldsymbol{a} \oplus \boldsymbol{e}_i$. It immediatly follows that $\alpha_i = 2$. Moreover, $\boldsymbol{b} \oplus \boldsymbol{e}_i = \boldsymbol{x} \oplus \boldsymbol{a} \oplus \boldsymbol{b} \in \mathcal{B}(1, \boldsymbol{b}) \cap \mathcal{Q}$ so $\beta_i = 2$ and $\boldsymbol{c} \oplus \boldsymbol{e}_i = \boldsymbol{x} \oplus \boldsymbol{a} \oplus \boldsymbol{c} \in \mathcal{B}(1, \boldsymbol{c}) \cap \mathcal{Q}$ and thus $\gamma_i = 2$. Hence $A(\boldsymbol{x}) = \alpha_i = 2$, $B(\boldsymbol{x}) = \beta_i + \beta_{j_{\boldsymbol{b}}} = 3$ and $C(\boldsymbol{x}) = \gamma_i + \gamma_{j_{\boldsymbol{c}}} = 3$. Finally, $A(\boldsymbol{x}) + B(\boldsymbol{x}) + C(\boldsymbol{x}) = 8$.

- If $\boldsymbol{x} \in \mathcal{B}(1, \boldsymbol{b})$, since $\boldsymbol{a}, \boldsymbol{b}, \boldsymbol{a} \oplus \boldsymbol{b} \oplus \boldsymbol{c} \notin \mathcal{Q}$, $\exists i \notin \{j_{\boldsymbol{b}}, j_{\boldsymbol{c}}\} : \boldsymbol{x} = \boldsymbol{b} \oplus \boldsymbol{e}_i$. We have $A(\boldsymbol{x}) = \alpha_i + \alpha_{j_{\boldsymbol{b}}}$, $B(\boldsymbol{x}) = \beta_i = 2$ and $C(\boldsymbol{x}) = \gamma_i + \gamma_{j_{\boldsymbol{b}}} + \gamma_{j_{\boldsymbol{c}}}$. Since $\boldsymbol{x} = \boldsymbol{a} \oplus \boldsymbol{e}_i \oplus \boldsymbol{e}_{j_{\boldsymbol{b}}} = \boldsymbol{c} \oplus \boldsymbol{e}_i \oplus \boldsymbol{e}_{j_{\boldsymbol{b}}} \oplus \boldsymbol{e}_{j_{\boldsymbol{c}}}$, $\boldsymbol{x} \in \mathcal{Q}$ but $\boldsymbol{x} \notin \mathcal{S}(1, \boldsymbol{a}) \cup \mathcal{S}(1, \boldsymbol{c})$. Hence $\boldsymbol{x} \in \mathcal{P}_{\boldsymbol{b}}$ or $\boldsymbol{x} \in \mathcal{Q}_1$.

  – If $\boldsymbol{x} \in \mathcal{P}_{\boldsymbol{b}}$, $\boldsymbol{c} \oplus \boldsymbol{e}_i = \boldsymbol{x} \oplus \boldsymbol{b} \oplus \boldsymbol{c} \in \mathcal{Q}_3 \subseteq \mathcal{Q} \cap \mathcal{B}(1, \boldsymbol{c})$ so $\gamma_i = 2$.

  – If $\boldsymbol{x} \in \mathcal{Q}_1$, $\boldsymbol{a} \oplus \boldsymbol{e}_i = \boldsymbol{x} \oplus \boldsymbol{a} \oplus \boldsymbol{b} \in \mathcal{P}_a \subseteq \mathcal{Q} \cap \mathcal{B}(1, \boldsymbol{a})$ so $\alpha_i = 2$.

  In both cases, $A(\boldsymbol{x}) + B(\boldsymbol{x}) + C(\boldsymbol{x}) \geqslant 8$.

- If $\boldsymbol{x} \in \mathcal{B}(1, \boldsymbol{c})$, since $\boldsymbol{a}, \boldsymbol{c}, \boldsymbol{a} \oplus \boldsymbol{b} \oplus \boldsymbol{c} \notin \mathcal{Q}$, $\exists i \notin \{j_{\boldsymbol{b}}, j_{\boldsymbol{c}}\} : \boldsymbol{x} = \boldsymbol{c} \oplus \boldsymbol{e}_i$. We have $A(\boldsymbol{x}) = \alpha_i + \alpha_{j_{\boldsymbol{c}}} = \alpha_i + 2$, $B(\boldsymbol{x}) = \beta_i + \beta_{j_{\boldsymbol{b}}} + \beta_{j_{\boldsymbol{c}}}$ and $C(\boldsymbol{x}) = \gamma_i = 2$. Hence $A(\boldsymbol{x}) + B(\boldsymbol{x}) + C(\boldsymbol{x}) \geqslant 8$.

Finally, let $\boldsymbol{x} \in (\mathcal{B}(1, \boldsymbol{a}) \cup \mathcal{B}(1, \boldsymbol{b}) \cup \mathcal{B}(1, \boldsymbol{c})) \setminus \mathcal{Q}$.

- If $\boldsymbol{x} \in \{\boldsymbol{a}, \boldsymbol{b}, \boldsymbol{c}, \boldsymbol{a} \oplus \boldsymbol{b} \oplus \boldsymbol{c}\}$, one can easily check that $A(\boldsymbol{x}) + B(\boldsymbol{x}) + C(\boldsymbol{x}) \leqslant 7$.

- If $\boldsymbol{x} \in \mathcal{B}(1, \boldsymbol{a}) \setminus \{\boldsymbol{a}, \boldsymbol{b}, \boldsymbol{c}\}$, $\exists i \notin \{j_{\boldsymbol{b}}, j_{\boldsymbol{c}}\} : \boldsymbol{x} = \boldsymbol{a} \oplus \boldsymbol{e}_i$. $A(\boldsymbol{x}) = \alpha_i$, $B(\boldsymbol{x}) = \beta_i + \beta_{j_{\boldsymbol{b}}} = \beta_i + 1$ and $C(\boldsymbol{x}) = \gamma_i + \gamma_{j_{\boldsymbol{c}}} = \gamma_i + 1$. Since $\boldsymbol{x} = \boldsymbol{a} \oplus \boldsymbol{e}_i \notin \mathcal{Q}$, we have $A(\boldsymbol{x}) = \alpha_i = 1$.

- If $\boldsymbol{x} \in \mathcal{B}(1, \boldsymbol{b}) \setminus \{a, b, \boldsymbol{a} \oplus \boldsymbol{b} \oplus \boldsymbol{c}\}$, $\exists i \notin \{j_{\boldsymbol{b}}, j_{\boldsymbol{c}}\} : \boldsymbol{x} = \boldsymbol{b} \oplus \boldsymbol{e}_i$. $A(\boldsymbol{x}) = \alpha_i + \alpha_{j_{\boldsymbol{b}}} = \alpha_i + 1$, $B(\boldsymbol{x}) = \beta_i = 1$ and $C(\boldsymbol{x}) = \gamma_i + \gamma_{j_{\boldsymbol{b}}} + \gamma_{j_{\boldsymbol{c}}} = \gamma_i + 2$. If $\alpha_i = 2$, $\boldsymbol{a} \oplus \boldsymbol{e}_i \in \mathcal{Q}$ then $\boldsymbol{a} \oplus \boldsymbol{e}_i \in \mathcal{P}_a$ and $\boldsymbol{x} = (\boldsymbol{a} \oplus \boldsymbol{e}_i) \oplus \boldsymbol{a} \oplus \boldsymbol{b} \in \mathcal{Q}_1 \subseteq \mathcal{Q}$. Hence $\alpha_i = 1$.

- If $\boldsymbol{x} \in \mathcal{B}(1, \boldsymbol{c}) \setminus \{\boldsymbol{a}, \boldsymbol{b}, \boldsymbol{a} \oplus \boldsymbol{b} \oplus \boldsymbol{c}\}$, $\exists i \notin \{j_{\boldsymbol{b}}, j_{\boldsymbol{c}}\} : \boldsymbol{x} = \boldsymbol{b} \oplus \boldsymbol{e}_i$. $A(\boldsymbol{x}) = \alpha_i + \alpha_{j_{\boldsymbol{c}}} = \alpha_i + 2$, $B(\boldsymbol{x}) = \beta_i + \beta_{j_{\boldsymbol{b}}} + \beta_{j_{\boldsymbol{c}}} = \beta_i + 2$ and $C(\boldsymbol{x}) = \gamma_i = 1$. If $\alpha_i = 2$, $\boldsymbol{a} \oplus \boldsymbol{e}_i \in \mathcal{Q}$ then $\boldsymbol{a} \oplus \boldsymbol{e}_i \in \mathcal{P}_a$ and $\boldsymbol{x} = (\boldsymbol{a} \oplus \boldsymbol{e}_i) \oplus \boldsymbol{a} \oplus \boldsymbol{c} \in \mathcal{Q}_2 \subseteq \mathcal{Q}$. Hence $\alpha_i = 1$. If $\beta_i = 2$ then $\boldsymbol{b} \oplus \boldsymbol{e}_i \in \mathcal{Q}$.

  – If $\boldsymbol{b} \oplus \boldsymbol{e}_i \in \mathcal{P}_{\boldsymbol{b}}$, $\boldsymbol{x} \in \mathcal{Q}_3 \subseteq \mathcal{Q}$.

  – If $\boldsymbol{b} \oplus \boldsymbol{e}_i \in \mathcal{Q}_1$, $\boldsymbol{a} \oplus \boldsymbol{e}_i \in \mathcal{P}_a$ and $\boldsymbol{x} = (\boldsymbol{a} \oplus \boldsymbol{e}_i) \oplus \boldsymbol{a} \oplus \boldsymbol{c} \in \mathcal{Q}_2 \subseteq \mathcal{Q}$.

  Hence $\beta_i = 1$.

In conclusion, we always have in this last case that $A(\boldsymbol{x}) + B(\boldsymbol{x}) + C(\boldsymbol{x}) \leqslant 7$.

□

Table 4.5 – Number of inequalities for modeling various 5 and 6-bit Sboxes with four different methods.

			# Inequalities			
n	Sbox	Citation	Convex Hull	Alg. 7	Alg. 6	Alg. 7 and 8
5	Keccak	[Nise]	46	46	**34**	36
	ASCON	[Dob+19]	40	59	**32**	49
	Fides-5	[Bil+13]	79	124	64	**61**
	SC2000-5	[Shi+02]	82	123	66	**64**
6	APN-6 †	[Bro+10]	195	288	**167**	179
	Fides-6	[Bil+13]	223	455	**180**	194
	SC2000-6	[Shi+02]	241	567	218	**214**

†The concrete function analyzed here is the one given through its table representation by John Dillon in his talk at *Fq09* [Dil09].

4.2.7 Comparing different techniques for Sbox modeling

Using 5 or 6-bit Sboxes inside a cipher is less common than using 4-bit and 8-bit ones. However, some designs, e.g. Keccak, ASCON or Fides among others, use such Sboxes for different reasons each: good masking properties or optimal resistance against differential cryptanalysis among others. Indeed, APN permutations — i.e. permutations which have DDTs only composed of 0s and 2s and which consequently have the best possible differential properties — exist for 5 and 6 bits. We tested our algorithms on some Sboxes of this size, mainly for permitting comparison between the different developed methods. Indeed, these sizes, not as small as 4 bits and not as large as 8 bits are ideal for permitting all of the algorithms to run (even the ones based on the computation of the convex hull) and for providing non-trivial comparisons. The results are summarized in Table 4.5. The first method, that we note as Convex Hull, corresponds to the method based on the H-representation of the convex hull provided by Sun et al. in [Sun+14b].

As one can see from the above table, Algorithm 6 and the combination of Algorithm 7 and Algorithm 8 are the ones giving always the best results. Algorithm 6 is almost always better but it has the disadvantage that it cannot be applied to larger dimensions.

4.3 Linear layer modeling

As seen in the previous section, modeling propagations through 8-bit Sboxes may lead to large systems of $\mathbb{R}$-linear inequalities. One would think that modeling a linear layer is much easier. While this is true for simple linear layers as the ones of PRESENT or GIFT that consist in simple bit-permutations, modeling other linear layers can also lead to large systems of $\mathbb{R}$-linear inequalities. This is in particular due to the difficulty of efficiently modeling the xor operation that is the major component of most diffusion layers.

4.3.1 `xor` modeling

Modeling a linear layer is often related to how the `xor` operation is modeled. The following proposition gives an idea of how difficult it can be to efficiently model a linear layer without dummy variables.

Proposition 4.5. *The equation $x_0 \oplus x_1 \oplus \ldots \oplus x_{n-1} = 0$ needs at least 2^{n-1} $\mathbb{R}$-linear inequalities of the form*

$$\sum_{i=0}^{n-1} c_i x_i + d \geqslant 0, \quad c \in \mathbb{R}^n, d \in \mathbb{R}$$

to characterize the set of its solutions in $\mathbb{F}_2^n$.

Proof. First, we can exhibit such a set of inequalities. Indeed, for each $\boldsymbol{a} \in \mathbb{F}_2^n$ such that $a_0 \oplus a_1 \oplus \ldots \oplus a_{n-1} = 1$, we have seen in Section 4.2 that we can write down an inequality that only eliminates a from the set of possible solutions. Since there are exactly 2^{n-1} such points, we have 2^{n-1} inequalities modeling $\{ \boldsymbol{x} \in \mathbb{F}_2^n \mid x_0 \oplus \ldots \oplus x_{n-1} = 0 \}$.

Let us suppose now that there exists an inequality $\langle \boldsymbol{c} \mid \boldsymbol{x} \rangle + d \geqslant 0$ that eliminates at the same time two points $\boldsymbol{a}$ and $\boldsymbol{b}$ such that $a_i = 0$ and $b_i = 1$ for some $i \in [0, n-1]$. Then,

- if $c_i \leqslant 0$, $\langle \boldsymbol{c} \mid \boldsymbol{a} \oplus \boldsymbol{e}_i \rangle + d = \langle \boldsymbol{c} \mid \boldsymbol{a} \rangle + d + c_i < 0$.
- Otherwise, if $c_i > 0$, $\langle \boldsymbol{c} \mid \boldsymbol{b} \oplus \boldsymbol{e}_i \rangle + d = \langle \boldsymbol{c} \mid \boldsymbol{b} \rangle + d - c_i < 0$.

This means that if an inequality eliminates two different points on the cube $\{0,1\}^n$, it necessarily also eliminates two points at Hamming distance 1.

Moreover, two points $\boldsymbol{a}$ and $\boldsymbol{b}$ such that $\mathrm{wt}(\boldsymbol{a} \oplus \boldsymbol{b}) = 1$ cannot both be solutions of $x_1 \oplus \ldots \oplus x_n = 0$. This explains why we need as many $\mathbb{R}$-linear inequalities as points to eliminate, i.e. 2^{n-1}. □

Therefore, a naive way to model a linear layer is to model each `xor` operation in the way showed in Proposition 4.5. We will often refer to this as the *naive method*. The rest of this section is dedicated to the presentation of more efficient ways for modeling general linear layers.

4.3.2 Modeling matrices over $\mathbb{F}_2$

When modeling a mathematical operation for MILP with a system of linear inequalities, the input and output variables play the same role inside each inequality. This shows that modeling a matrix $\boldsymbol{M}$ means modeling the kernel of $\boldsymbol{A} = (\boldsymbol{M} | \boldsymbol{I})$, where $\boldsymbol{I}$ is the identity matrix. Indeed, for any matrix $\boldsymbol{M}$ with entries in $\mathbb{F}_2$,

$$\boldsymbol{M}\boldsymbol{x} = \boldsymbol{y} \iff \boldsymbol{M}\boldsymbol{x} \oplus \boldsymbol{y} = \boldsymbol{0} \iff \boldsymbol{A} \begin{pmatrix} \boldsymbol{x} \\ \boldsymbol{y} \end{pmatrix} = \boldsymbol{0}.$$

One can then model the equation given by each row of $\boldsymbol{A}$ with the `xor` modeling. But as we have just seen, the number of constraints for modeling one `xor` operation grows exponentially with the number of involved variables. Since our goal is to model the kernel of $\boldsymbol{A}$ and since it is known that for any invertible matrix $\boldsymbol{P} \in \mathrm{GL}_n(\mathbb{F}_2), \ker(\boldsymbol{P} \cdot \boldsymbol{A}) = \ker \boldsymbol{A}$, the idea is to find a matrix $\boldsymbol{P} \in \mathrm{GL}_n(\mathbb{F}_2)$ such that the rows of $\boldsymbol{P} \cdot \boldsymbol{A}$ have minimum Hamming weight and induce thus a minimal number of `xor` operations. More precisely, this means finding an invertible matrix $\boldsymbol{P}$ that minimizes

$$\sum_{i=1}^{n} 2^{\mathrm{wt}((\boldsymbol{P} \cdot \boldsymbol{A})_{i,*}) - 1}, \tag{4.4}$$

where $\mathrm{wt}\left((\boldsymbol{P}\cdot\boldsymbol{A})_{i,\star}\right)$ corresponds to the Hamming weight of the i-th row of $\boldsymbol{P}\cdot\boldsymbol{A}$. The question is now how to find such a matrix $\boldsymbol{P}$. A first idea would be to search for minimum-weight codewords in the linear code generated by the rows of $\boldsymbol{A}$, as this is done when computing the linear branch number of the matrix $\boldsymbol{M}$. For example, consider the matrix

$$\boldsymbol{M}_{\texttt{Midori}} = \begin{pmatrix} 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 0 \end{pmatrix} \qquad (\boldsymbol{M}_{\texttt{Midori}}|\boldsymbol{I}) = \begin{pmatrix} 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 1 & 0 & 0 & 1 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}$$

used in the linear layer of `Midori`. This matrix has branch number 4, which means that for all $\boldsymbol{x}\in\mathbb{F}_2^4$, $\mathrm{wt}\left(\boldsymbol{x}\cdot(\boldsymbol{M}|\boldsymbol{I})\right)\geqslant 4$. Hence we cannot hope for a better modeling of this linear operation based on the `xor` modeling than the one given by $(\boldsymbol{M}|\boldsymbol{I})$. However, let us take the example of the `SKINNY` `MixColumns` operation given by the matrix $\boldsymbol{M}_{\texttt{SKINNY}}$ below. In that case, the code generated by $(\boldsymbol{M}_{\texttt{SKINNY}}|\boldsymbol{I})$ has minimum distance 2 and it is equivalently generated by the matrix $\boldsymbol{A}_{\texttt{SKINNY}}$ obtained by adding in $\mathbb{F}_2$ the fourth line to the first line of $(\boldsymbol{M}|\boldsymbol{I})$.

$$\boldsymbol{M}_{\texttt{SKINNY}} = \begin{pmatrix} 1 & 0 & 1 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 0 \end{pmatrix} \qquad \boldsymbol{A}_{\texttt{SKINNY}} = \begin{pmatrix} 0 & 0 & 0 & 1 & 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}.$$

While the naive `xor` modeling of $(\boldsymbol{M}_{\texttt{SKINNY}}|\boldsymbol{I})$ would have needed $2^3+2+2^2+2^2=18$ inequalities, using the above matrix for the `xor` modeling only requires 14 inequalities.

To demonstrate that this representation is more efficient in practice compared to the naive approach, we computed the time it takes for the `Gurobi` Optimizer [GO20] to reach the minimum number of active `Sbox`es over several rounds of `SKINNY-128` for the two different modelings of `MixColumns`. In this experiment, in order to emphasize the impact of the linear layer modeling and to avoid correlations with the modelings of the other parts of the cipher, we used a very simple modeling for the 8-bit `Sbox`. This `Sbox` modeling, introduced in [ST17b] under the name of *arbitrary* `Sbox` *mode*, needs only $2n$ inequalities for the whole `Sbox` but only models the following behaviour: if at least one input bit is active then at least one output bit has to be active and vice versa. The timings (in seconds) of this experiment can be visualised in Table 4.6. We emphasize that the reported timings is the time for the solver to *reach* what we already know to be the minimum number of active `Sbox`es. Indeed, as even with the improved modeling the number of inequalities still remains high, our MILP solver takes too long for terminating and thus *proving* that the found upper bound on the minimum number of active `Sbox`es is tight. The minimum number of active `Sbox`es for `SKINNY` has been computed by its designers in [Bei+16] thanks to wordwise modelings.

It is obvious from this table that the new modeling of the `SKINNY` linear layer reduces importantly the solving time. We propose now a new algorithm, Algorithm 9, that given the matrix $\boldsymbol{A}=(\boldsymbol{M}\mid\boldsymbol{I})$, finds a matrix $\boldsymbol{P}$ that minimizes Equation (4.4). First, the matrix $\boldsymbol{P}$ is initialized to the identity matrix. Then, the algorithm proceeds in a row-wise manner and searches at each step to replace the current row with a better one. To start with, it searches to replace the first row of $\boldsymbol{A}$ with a codeword of the form

$$\boldsymbol{m}\cdot\boldsymbol{A},\ \boldsymbol{m}\in\{\,\boldsymbol{x}\in\mathbb{F}_2^n\mid x_1=1\,\}\ \text{and}\ \mathrm{wt}\left(\boldsymbol{m}\cdot\boldsymbol{A}\right)<\mathrm{wt}\left(\boldsymbol{A}_{1,*}\right).$$

Table 4.6 – Computation time in seconds for the `Gurobi` solver to find the minimum number of active `Sboxes` over r rounds for `SKINNY-128` with two different modelings for `MixColumns`.

Number of rounds	6	7	8	9	10
Minimum number of active `Sboxes`	16	26	36	41	46
Time to reach this minimum with the new modeling	0	0	0	16	5200
Lowest upper bound with naive linear layer or - if the minimum was reached	-	-	-	-	47
Time at which we stopped the experiment or time after which the minimum was reached	16	35	30	1862	14600

After this first step, the matrix $\boldsymbol{P}$ is updated as

$$\boldsymbol{P} = \begin{pmatrix} 1 & m_2 & \cdots & m_n \\ & 1 & & \\ & & \ddots & \\ & & & 1 \end{pmatrix}.$$

The algorithm then searches for a replacement for the second row of the matrix $\boldsymbol{P} \cdot \boldsymbol{A}$ in the same way and updates the matrix $\boldsymbol{P}$ if a lower weight codeword has been found.

Algorithm 9 Given a binary matrix $\boldsymbol{A}$ of size $n \times N$ returns $\boldsymbol{P} \in \mathrm{GL}_n(\mathbb{F}_2)$ minimizing Equation (4.4).

procedure FINDP($\boldsymbol{A}$)
  $\boldsymbol{A}^{\mathrm{mut}} \leftarrow \boldsymbol{A}$
  $\boldsymbol{P} \leftarrow \boldsymbol{I}_n$
  for $\ell \in \{1, \ldots, n\}$ **do**
    $\boldsymbol{m}_{\mathrm{best}} \leftarrow \boldsymbol{A}^{\mathrm{mut}}_{\ell}$
    for all $\boldsymbol{m} \in \{\, \boldsymbol{x} \in \mathbb{F}_2^n \mid x_\ell = 1 \,\}$ **do**
      if $\mathrm{wt}\left(\boldsymbol{m} \cdot \boldsymbol{A}^{\mathrm{mut}}\right) < \mathrm{wt}\left(m_{\mathrm{best}}\right) \cdot \boldsymbol{A}^{\mathrm{mut}}$ **then**
        $\boldsymbol{m}_{\mathrm{best}} \leftarrow \boldsymbol{m}$
    $\boldsymbol{A}^{\mathrm{mut}}_{\ell} \leftarrow \boldsymbol{m}_{\mathrm{best}} \cdot \boldsymbol{A}^{\mathrm{mut}}$
    $\boldsymbol{P} \leftarrow \begin{pmatrix} 1 & & & \\ & \ddots & & \\ & \boldsymbol{m}_{\mathrm{best}} & & \\ & & \ddots & \\ & & & 1 \end{pmatrix} \cdot \boldsymbol{P}$
  return $\boldsymbol{P}$

Algorithm 9, whose time complexity is $n2^{n-1}$ multiplications of a n-bit binary vector with a matrix with n rows and an arbitrary number of N columns, finds notably the previous result for `SKINNY` (14 inequalities). We also applied Algorithm 9 to the AES `MixColumns`. The naive `xor` modeling gives 2176 inequalities for this matrix and Algorithm 9 does not improve this quantity.

We develop now a different idea that permits in some cases to significantly improve the number of inequalities needed for modeling the linear layer, compared to both the naive approach and Algorithm 9.

4.3.3 Changing the Sbox modeling for improving the linear modeling

The idea of this approach consists in changing the Sbox modeling. Indeed, if we find an invertible block-diagonal matrix (with blocks having the size of the Sbox)

$$\boldsymbol{Q} = \begin{pmatrix} \boldsymbol{Q}_1 & & & \\ & \boldsymbol{Q}_2 & & \\ & & \ddots & \\ & & & \boldsymbol{Q}_{2b} \end{pmatrix}$$

where b is the number of words on which `MixColumns` operates (e.g. $b = 4$ for the AES, `SKINNY` or `Midori`), then changing the modeling of the Sbox into the modelings of $\boldsymbol{Q}_i^{-1} \circ \texttt{Sbox} \circ \boldsymbol{Q}_{i+b}^{-1}$ for all $i \in [1, b]$ allows for a new, potentially better `xor` modeling of the `MixColumns` operation with the equation $(\boldsymbol{M}|\boldsymbol{I}) \cdot \boldsymbol{Q} \cdot \boldsymbol{x} = 0$. Once a convenient matrix $\boldsymbol{Q}$ has been found, modeling $(\boldsymbol{M}|\boldsymbol{I}) \cdot \boldsymbol{Q} \cdot \boldsymbol{x} = 0$ can be done using Algorithm 9. In summary, with this method, the problem of finding a good `xor` modeling for the linear layer boils down to finding a matrix $\boldsymbol{P} \in \mathrm{GL}_n(\mathbb{R})$ and a block-diagonal matrix $\boldsymbol{Q} \in \mathrm{GL}_{2n}(\mathbb{R})$ which minimize

$$\sum_{i=1}^{n} 2^{\mathrm{wt}((\boldsymbol{P} \cdot (\boldsymbol{M}|\boldsymbol{I}) \cdot \boldsymbol{Q})_{i,*}) - 1}. \tag{4.5}$$

We propose Algorithm 10 for finding such matrices $\boldsymbol{P}$ and $\boldsymbol{Q}$. The idea of this algorithm is to iterate alternating searches for $\boldsymbol{P}$ and $\boldsymbol{Q}$, hoping that modifying one of $\boldsymbol{P}$ or $\boldsymbol{Q}$ will allow further optimization. This algorithm then takes as a parameter the number of desired iterations $p \in \mathbb{N}$. In practice however, for all of our experiments, we have never needed $p \geqslant 2$.

Applying Algorithm 10 on the AES `MixColumns` operation with parameter $p = 1$ gives $\boldsymbol{P}$ and $\boldsymbol{Q}$ such that the quantity of Equation (4.4) initially at 2176 drops down to 1088. However, Algorithm 10 does not give better results for `Midori` or Algorithm 9 for `SKINNY`. Matrices $\boldsymbol{P}$ and $\boldsymbol{Q}$ for the AES are given in Appendix A.2.

To measure at which point such a change in the modeling of the linear layer can be an improvement for the running time, we ran an experiment for the AES similar to the one above for `SKINNY-128`. One important difference is that wordwise modelings for the AES have not been able to provide tight lower bounds for the minimum number of active Sboxes because of byte multiplications needed in the `MixColumns` operation. For example in [Mou+11], the authors used the branch number of the `MixColumns` operation to obtain lower bounds. In Table 4.7, we hence give the lowest upper bound on the minimum number of active Sboxes reached with the improved linear-layer modeling after a given number of seconds and the same upper bound reached with the naive linear layer after a given number of seconds.

On the complexity of Algorithms 9 and 10

For our implementation of Algorithms 9 and 10, we represented binary matrices as vectors of 64-bit integers, each integer representing a row. The multiplication of an n-bit binary vector with a $n \times N$

Algorithm 10 Given a matrix $\boldsymbol{A}$ of size $n \times N$, find $\boldsymbol{P} \in \mathrm{GL}_n(\mathbb{F}_2)$ and block-diagonal $\boldsymbol{Q}$ with $2b$ blocks such that $(\boldsymbol{P}, \boldsymbol{Q})$ minimizes $\sum_{i=1}^{n} \mathrm{wt}\left((\boldsymbol{P} \cdot \boldsymbol{A} \cdot \boldsymbol{Q})_i\right)$.

procedure FINDQ($\boldsymbol{A}, p, b$)
  $\boldsymbol{P} \leftarrow$ FINDP($\boldsymbol{A}$)
  $\boldsymbol{A}^{\mathrm{mut}} \leftarrow \boldsymbol{P}\boldsymbol{A}^{\mathrm{mut}}$
  $\boldsymbol{Q} \leftarrow \boldsymbol{I}_{2n}$
  loop p times
    for all $i \in [1, 2b]$ **do**
      $C_i \leftarrow$ columns $\frac{n \cdot i}{b}, \cdots, \frac{n \cdot (i+1)}{b} - 1$ of $\boldsymbol{A}^{\mathrm{mut}}$
      $\boldsymbol{Q}_i \leftarrow \mathrm{FINDP}(C_i^T)^T$
    $\boldsymbol{Q} \leftarrow \boldsymbol{Q} \cdot \begin{pmatrix} \boldsymbol{Q}_1 & & & \\ & \boldsymbol{Q}_2 & & \\ & & \ddots & \\ & & & \boldsymbol{Q}_{2b} \end{pmatrix}$
    $\boldsymbol{A}^{\mathrm{mut}} \leftarrow \boldsymbol{A}^{\mathrm{mut}} \cdot \begin{pmatrix} \boldsymbol{Q}_1 & & & \\ & \boldsymbol{Q}_2 & & \\ & & \ddots & \\ & & & \boldsymbol{Q}_{2b} \end{pmatrix}$
    $\boldsymbol{P} \leftarrow \mathrm{FINDP}(\boldsymbol{A}^{\mathrm{mut}}) \cdot \boldsymbol{P}$
    $\boldsymbol{A}^{\mathrm{mut}} \leftarrow \boldsymbol{P}\boldsymbol{A}^{\mathrm{mut}}$
  return $\boldsymbol{P}, \boldsymbol{Q}$

Table 4.7 – Time in seconds to reach the lowest upper bound for the minimum number of active Sboxes for the AES.

Number of rounds	5	6	7	8
Lowest upper bound with improved linear layer	28	35	47	65
Time needed to reach this upper bound	105	117	5547	6900
Lowest upper bound with naive linear layer	31	40	49	69
Time at which we stopped the experiment	2300	2200	8200	10000

matrix then takes at most n `xor`s of 64-bit integers when $N \leqslant 64$. With this implementation, the running time of Algorithm 9 on the AES `MixColumns` matrix on a single core is 929 seconds. In Algorithm 10, the computation of $\boldsymbol{Q}$ applies Algorithm 9 on $2b$ matrices with $\frac{n}{b}$ rows. The complexity is then

$$2b \cdot \left(\frac{n}{b}\right)^2 \cdot 2^{\frac{n}{b}-1} \texttt{ XOR}\text{s of 64-bit integers.}$$

The running time of this computation is then negligible compared to the computation of $\boldsymbol{P}$ in practice. Indeed, the computation of $\boldsymbol{P}$ applies Algorithm 9 on a matrix with n rows, which has a complexity of

$$n^2 \cdot 2^{n-1} \texttt{ XOR}\text{s of 64-bit integers.}$$

Finally, the running time of Algorithm 10 is $(p+1)$ times the running time of Algorithm 9.

Table 4.8 – Number of inequalities to model a diffusion matrix for MILP without dummy variables with the naive way, and with the two algorithms developed in this section.

	# Inequalities		
Cipher	Naïve	Alg. 9	Alg. 10
`Midori`	**32**	32	32
`SKINNY`	18	**14**	14
AES	2176	2176	**1088**
`Aria`	**2048**	2048	2048
`Anubis`	7168	2032	**1680**
`Saturnin`	5632	**352**	352

Modeling of affine equivalent Sboxes

As seen in this section, to apply Algorithm 10, instead of modeling the cipher's original Sbox (denoted here as $\texttt{Sbox}_{\text{origin}}$), one will need to model one or more Sboxes that are linearly equivalent to $\texttt{Sbox}_{\text{origin}}$. Therefore, it is necessary to ensure that the modeling of these linearly equivalent Sboxes is not (much) worse than the modeling for $\texttt{Sbox}_{\text{origin}}$. This leads to the following more general question: "How does the modeling of an Sbox gets affected by affine equivalence?" In our experiments with AES, the only needed affine equivalent Sbox — $\texttt{Sbox}_{\text{AES}} \circ \boldsymbol{Q}_0$ where $\boldsymbol{Q}_0$ is given in Appendix A.2 — had a very similar modeling to the original $\texttt{Sbox}_{\text{AES}}$, leading notably to almost the same number of final inequalities. It is however not clear for us what happens in the general case and we believe that this constitutes an interesting open problem. For example, it could be useful to know for any given $\texttt{Sbox}_{\text{origin}}$ whether one can compute an affine equivalent $\texttt{Sbox}_{\text{equiv}}$ that can be modeled with much fewer inequalities. A related question is whether a lower bound on the number of needed inequalities in the modeling can be found accross the equivalence class of an Sbox.

4.3.4 Other applications

Besides AES, `SKINNY` and `Midori` we also applied Algorithms 9 and 10 to the linear layers of some other block ciphers. The obtained results are summarized in Table 4.8 and the linear layers of `Anubis`, `Aria` and `Saturnin` are briefly described below.

`Anubis` is a 128-bit block cipher designed by Barreto and Rijmen in 2000 in the context of the NESSIE project. `Anubis` follows the SPN construction and uses an involutive `MixColumns` operation, based on an MDS matrix

$$\boldsymbol{H} = \begin{pmatrix} 1 & 2 & 4 & 6 \\ 2 & 1 & 6 & 4 \\ 4 & 6 & 1 & 2 \\ 6 & 1 & 2 & 4 \end{pmatrix} \in \mathcal{M}_4\left(\mathbb{F}_{2^8}\right).$$

Each entry of $\boldsymbol{H}$ is an element $\sum x_i X^i$ of the finite field $\mathbb{F}_2[X]/(X^8+X^4+X^3+X^2+1)$ represented by the integer $\sum x_i 2^i$. For the `Anubis`'s `MixColumns` operation, the quantity of Equation (4.4) initially at 7168 drops to 2032 after applying Algorithm 9 and to 1680 after applying Algorithm 10.

`Saturnin` is a 2-round candidate of the NIST Lightweight Crypto Competition, designed by Canteaut et al. [Can+19]. Its main component is a block cipher with a `MixColumns` operation applying the transformation $\boldsymbol{M} : \left(\mathbb{F}_2{}^4\right)^4 \to \left(\mathbb{F}_2^4\right)^4$ (see Section A.3) to different parts of the state. Algorithm 9 allows the quantity of Equation (4.4) initially at 5632 to drop down to 352 but Algorithm 10 does not permit to reduce this quantity further.

`Aria`. Presented at ICISC in 2003 by Kwon et al. [Kwo+04], `Aria` is an SPN block cipher whose diffusion layer consists in applying a 16×16 binary matrix (given in Section A.3) to different parts of the state. The quantity of Equation (4.4) is 2048 and is not improved by our algorithms.

4.4 Impact of the new modelings on the solving time

In this section we analyze the impact of our new modelings on the running time of a MILP optimization problem. We chose to perform our experiments on the problem of deciding whether a differential $(\boldsymbol{a}, \boldsymbol{b})$ is possible, which in practice consists in finding a possible differential characteristic, if this one exists. This kind of computation is used in practice for impossible differential cryptanalysis. We will provide more details on the full problem in Section 4.5.

Description of the experiment. The goal of these experiments is to quantify in terms of solver computation time the impact of both the `Sbox` and the `MixColumns` modelings. For this, we considered two different modelings for the `Sbox`: the logical method of Section 4.2.4 removing sets of the form $\boldsymbol{a} \oplus \mathrm{Prec}\left(\boldsymbol{u}\right)$ and a combination of this method and the method of Section 4.2.5 removing merged distorted Hamming balls. For the linear layer we also considered two cases: in the first case we model the `MixColumns` matrix via the naïve `xor` modeling and in the second we model it with the improved `xor` modeling given by Algorithms 9 and 10. We then considered all four different combinations of these `Sbox` and `MixColumns` modelings. For launching the experiment, we chose a random set of input and output difference pairs $(\boldsymbol{a}, \boldsymbol{b})$ and asked `Gurobi` to verify whether the differential $(\boldsymbol{a}, \boldsymbol{b})$ is possible.

First observations. We ran this experiment for `SKINNY-128` and the AES and present in Table 4.9 basic statistics on the computation times for each cipher. One can see in those experiments that the computation can be very long and that the standard deviation is huge. We do not have any explanation for this fact. However, the means and quantiles behave coherently with each other, which shows that a different MILP modeling does have a clear influence on the solving time. For the AES, the computation time is dramatically higher when the input and output are sparse (have much more zeros than ones) than when they are dense, hence the need for two different tables. Our intuition for this fact is that there are many more possible differential characteristics inside the differential when the input and the output are dense, which makes it easy for the solver to find one in a few seconds. For the application on AES with sparse ends, we also indicate to our models which bytes are active and inactive in the first two and last two rounds. This reduces the running time of the solver by approximately a factor 10.

The case of `SKINNY-128`. As we can see, for `SKINNY-128` the change of modeling for the linear layer has a much bigger impact on the running time than the change of the `Sbox` modeling. There are in our opinion three possible reasons for that: First, the number of rounds under study for

Table 4.9 – Statistics on the computation time in seconds for 32 rounds of `SKINNY-128` and 5 rounds of AES.

(a) 32-round `SKINNY-128` on a sample of 1975 input/output pairs.

`Sbox`	Alg. 7		Alg. 7 and 8	
`MixColumns`	Naïve	Improved	Naïve	Improved
Mean	185	**21**	172	29
Std	359	26	364	34
Min	5	5	5	5
25 %	6	6	6	6
50 %	61	**6**	7	**6**
75 %	106	**7**	104	63
Max	1876	**120**	2084	292

(b) 5-round AES on a sample of 4594 dense input/output pairs.

`Sbox`	Alg. 7		Alg. 7 and 8	
`MixColumns`	Naïve	Improved	Naïve	Improved
Mean	43	40	**16**	25
Std	2	7	4	28
Min	22	17	**8**	**8**
25 %	40	40	**11**	19
50 %	42	41	**13**	20
75 %	44	44	**20**	22
Max	55	53	**52**	379

(c) 5-round AES on a sample of 259 sparse input/output pairs.

`Sbox`	Alg. 7		Alg. 7 and 8	
`MixColumns`	Naïve	Improved	Naïve	Improved
Mean	1599	1944	**441**	662
Std	1560	2051	166	274
Min	17	24	**7**	11
25 %	983	1047	**382**	460
50 %	1194	1465	**424**	702
75 %	2036	2347	**488**	822
Max	18237	23566	**1062**	2824

`SKINNY-128`, 32, is high enough for the modeling of the linear layer to have impact. Then, as one can see from Table 4.4, the difference between the two `Sbox` modelings is not significant. Finally, the number of inequalities per `Sbox` is rather low (< 400), which gives the linear layer modeling more impact as well.

The case of the AES. For the AES, the inverse phenomenon happens: the linear layer modeling does not have an important impact whereas the `Sbox` modeling divides the running time by 2. Again, we think that a possible explanation of this is that the number of rounds, 5, is too low for the linear layer to have an important impact, the difference between the two `Sbox` modelings is significant (see Table 4.4) and the number of inequalities per `Sbox` is rather high (≈ 3000) making that the `Sbox` contraints have a higher proportion in the global process.

4.5 Applications on impossible differential cryptanalysis

Impossible differential cryptanalysis, presented in Section 2.4, consists in finding impossible differentials, i.e. an input difference $\boldsymbol{a}$ and an output difference $\boldsymbol{b}$ that cannot be connected. Any impossible differential attack starts with the discovery of an impossible differential $(\boldsymbol{a}, \boldsymbol{b})$ covering a maximal number of rounds. Traditionally this was done with the $\mathcal{U}$-method [Kim+03] or its extensions [Luo+14; WW12]. However, in 2017, Sasaki and Todo [ST17b], showed that MILP can be successfully used to prove resistance against impossible differential attacks or for discovering new impossible differential distinguishers. For proving (partial) resistance against impossible differential cryptanalysis with MILP, and as briefly explained in Section 4.4, one can choose a set of input and output pairs and then solve a MILP differential propagation problem with the given cipher model for each of those pairs. The chosen set is typically composed of all possible inputs and outputs with exactly one active byte, i.e. for which exactly one byte has a difference. When all of those computations result in a valid differential transition, showing thus that the input and output differences can be connected, we consider that resistance against impossible differential cryptanalysis has been partially proven, where partially applies to the fact that the input and output spaces are restricted. However, as seen in Section 4.4, a single computation with a fixed input and output difference can be too long for permitting to do such kind of proofs for a large meaningful enough set of input/output pairs. To overcome this problem Sasaki and Todo introduced in [ST17b, Section 5], the *Differential Possibility Equivalence technique*. This technique reduced the number of MILP instances to solve and permitted to drastically reduce the overall running time of this process. In what follows we briefly present this technique. Then we show how to further improve this method for decreasing the running time further. We applied this technique to prove partial resistance against differential cryptanalysis for 5-round AES and 13-round `SKINNY-128`.

4.5.1 The Differential Possibility Equivalence technique

We consider for the rest of this section that an r-round MILP model is composed of r non-linear layers and begins and ends with a non-linear layer. It is then also obviously composed of $r-1$ linear layers.

Suppose that we search for r-round impossible differentials for an SPN cipher. Suppose further that the pairs of input and output differences are restricted in a set $\mathcal{S}$ and that we have computed a possible differential characteristic for r rounds $\boldsymbol{x}_0 \xrightarrow{r} \boldsymbol{y}_0$. The idea of the Differential Possibility

Equivalence technique is to exploit this possible differential characteristic to discard other possible transitions $(\boldsymbol{x}, \boldsymbol{y}) \in \mathcal{S}$ without computing at each time a new r-round differential characteristic $\boldsymbol{x} \xrightarrow{r} \boldsymbol{y}$ from scratch. To do so, one chooses r_{in} and r_{out} such that $r_{\text{in}} + r_{\text{out}} < r$ and gets the differences $\boldsymbol{x}'$ and $\boldsymbol{y}'$ in the already computed path

$$\boldsymbol{x}_0 \xrightarrow{r_{\text{in}}} \boldsymbol{x}' \xrightarrow{r-r_{\text{in}}-r_{\text{out}}} \boldsymbol{y}' \xrightarrow{r_{\text{out}}} \boldsymbol{y}_0.$$

One then discards all $(\boldsymbol{x}, \boldsymbol{y}) \in \mathcal{S}$ such that the transitions $\boldsymbol{x} \xrightarrow{r_{\text{in}}} \boldsymbol{x}'$ and $\boldsymbol{y}' \xrightarrow{r_{\text{out}}} \boldsymbol{y}$ are possible, which means computing two smaller differential paths on r_{in} and r_{out} rounds. Sasaki and Todo introduced this technique in [ST17b] with $\boldsymbol{x}'$ and $\boldsymbol{y}'$ respectively right before and right after the first and last non-linear layers (hence with $r_{\text{in}} = r_{\text{out}} = 1$), finding $\boldsymbol{x}$ and $\boldsymbol{y}$ by directly looking at the DDT of the `Sbox`es used in those non-linear layers. However, it can be interesting to try the same with other values for r_{in} and r_{out}, using MILP models for checking differential paths $\boldsymbol{x} \xrightarrow{r_{\text{in}}} \boldsymbol{x}'$ and $\boldsymbol{y}' \xrightarrow{r_{\text{out}}} \boldsymbol{y}$.

The choice of r_{in} and r_{out} for finding a minimal running time depends on the cipher and the running times for computing differential characteristics for respectively r, r_{in} and r_{out} rounds. Moreover, one should avoid to check differential transitions $\boldsymbol{x} \xrightarrow{r_{\text{in}}} \boldsymbol{x}'$ and $\boldsymbol{y}' \xrightarrow{r_{\text{out}}} \boldsymbol{y}$ when they have a low probability to be possible. In the search for impossible differentials with exactly one active input and one active output byte, it appears that it is rather efficient to restrain the checks for $(\boldsymbol{x}, \boldsymbol{y})$ such that $\boldsymbol{x}$ shares the same active byte as $\boldsymbol{x}_0$ and $\boldsymbol{y}$ shares the same active byte as $\boldsymbol{y}_0$. Indeed, for the AES with $r = 5, r_{\text{in}} = r_{\text{out}} = 2$, the computation of one r-round differential characteristic $\boldsymbol{x}_0 \xrightarrow{r} \boldsymbol{y}_0$ allows to discard all the other pairs $(\boldsymbol{x}, \boldsymbol{y})$ with the same active bytes. For `SKINNY-128` with $r = 13, r_{\text{in}} = r_{\text{out}} = 2$, for 2 checks $\boldsymbol{x} \xrightarrow{2} \boldsymbol{x}'$ or $\boldsymbol{y}' \xrightarrow{2} \boldsymbol{y}$, one possible transition gets discarded on average.

4.5.2 Applications to `SKINNY-128` and AES

In [ST17b] the authors used MILP for proving the maximal number of rounds for which impossible differentials exist for many different designs, if the input and output differences are restricted inside one word (bit, nibble or byte, depending on the design). Most of the analyzed ciphers are based on 4-bit `Sbox`es while for the analyzed designs using 8-bit `Sbox`es the details of the `Sbox`es are not taken into account. The only exception is `Midori` but whose `Sbox`es are constructed from two 4-bit ones.

Our new modelings for both big `Sbox`es and linear layers and the above generalization of the Differential Possibility Equivalence technique permitted us to complement the work done in [ST17b] for 8-bit-based ciphers, by taking in particular the `Sbox` details into account. For both 5-round AES and 13-round `SKINNY-128` [1], for each pair of input/output bytes (i, j), we ran our models with the Differential Possibility Equivalence technique on the set with 255×255 elements with byte i active in input and byte j active in output. Those computations provide us with proofs for partial resistance against impossible differential cryptanalysis.

The case of `SKINNY-128`. For the parameters $r = 13, r_{\text{in}} = r_{\text{out}} = 2$, the proof for each pair of input/output active byte has been completed in an average time of 15 minutes on 2 cores of Intel XEON E3-1240 v5. To the best of our knowledge, this proof is a new result.

[1] There is an ambiguity in the literature over the number of rounds on which impossible differentials are found for `SKINNY-128`. For us, 13 rounds means that there are 13 `Sbox` layers and 12 linear layers.

The case of the AES. For the parameters $r = 5, r_{\text{in}} = r_{\text{out}} = 2$, the proof for each pair of input/output active byte completed in an average time of 10 hours on 2 cores of Intel XEON E3-1240 v5. Sun et al. proved in [Sun+16b] without any restriction on the number of active input or output bytes, that there are no 5-round impossible differentials for the AES, unless the details of the `Sbox` are taken into account. By taking the details of the `Sbox` into account, we provide an extension of Sun et al.'s proof, in the case of one active input and one active output byte. Note however that Wang and Jin gave recently in [WJ19] a theoretical proof for 5 rounds of AES. Indeed, they proved, that for any number of active input/output bytes, all differentials are possible if round-keys are taken uniformly at random.

4.6 Conclusion

In this chapter, we presented new techniques for improving MILP models simulating differential properties through `Sbox`es and `MixColumns` operations. We found better modelings for various `Sbox`es with sizes ranging from 4 to 8 bits with three different approaches: adding inequalities given by the H-representation of the convex hull of possible transitions, packing impossible transitions in affine subspaces of the form $\boldsymbol{a} \oplus \text{Prec}(\boldsymbol{u})$ or packing them in (possibly "merged") distorted balls. Our techniques for `Sbox` modeling are very general: they basically give efficient algorithms to represent any subset of $\{0,1\}^n$ with a small number of inequalities. Those techniques could then have applications beyond differential cryptanalysis. We also introduced a link between the modeling of $\mathbb{F}_2$-linear operations and the search for a basis with minimal-weight codewords inside $\mathbb{F}_2$-linear codes. We gave algorithms based on this link to reduce the inequality cost of `MixColumns` modelings.

We then gave insights on how those techniques impact the performance of computing a differential characteristic with a fixed input and output for `SKINNY-128` and the AES. Those two cases respectively demonstrate the benefits of better modelings for the `MixColumns` operations and for the `Sbox`es. Finally, we applied those techniques to proving partial resistance of 13-round `SKINNY-128` and 5-round AES against impossible differential cryptanalysis with the specific properties of the `Sbox`es fully taken into account.

Lastly, our techniques could be applied in a straightforward manner to the search for best differential characteristics in various ciphers as explained in [Abd+17] and to similar problems of linear cryptanalysis. The optimal way of solving the *Efficient subset* problem to improve performance remains an open question.

Chapter 5

Algebraic Normal Form analysis

The work presented in this chapter is the result of personal work and experiments that have not been published elsewhere.

5.1 Introduction

Almost any symmetric primitive can be seen as a map $E : \mathbb{F}_2^n \times \mathbb{F}_2^m \to \mathbb{F}_2^s$ with public and private inputs, respectively denoted by $\boldsymbol{x} \in \mathbb{F}_2^n$ and $\boldsymbol{k} \in \mathbb{F}_2^m$. As will be explained in Section 5.2, a coordinate of such a map — called a Boolean function — can be seen as a polynomial over $\mathbb{F}_2$ with public variables $X_0, \ldots, X_{n-1}$ and private variables $K_0, \ldots, K_{m-1}$. In chosen-plaintext attacks for example, we want to recover some information on the key $\boldsymbol{k}$ — the private input — when we can query the value $E(\boldsymbol{x}, \boldsymbol{k})$ on the public input $\boldsymbol{x}$ of our choice. Each coordinate $E_i\left(X_0, \ldots, X_{n-1}, K_0, \ldots, K_{m-1}\right)$ is a Boolean function and its polynomial representation is called its Algebraic Normal Form (ANF).

Algebraic attacks. For $E(X, K)$ to be indistinguishable from random or to resist key-recovery attacks, it is important to understand the behaviour of its ANF. For example, if E is a random function, each coefficient in the ANFs of its coordinates E_i follows a uniform law of probability over $\mathbb{F}_2$. Therefore, detecting a biased property in some ANF, like a small degree, can lead to distinguishing attacks or even to key recoveries. However, since knowing (and storing) an ANF is equivalent to knowing the entire code-book of a Boolean function (see Proposition 5.1), many algebraic attacks try to avoid a precise analysis of the ANF. Indeed, here are a few examples of algebraic attacks.

- In classical algebraic attacks [CP02; CM03], the knowledge of the internals of the cipher and the knowledge of plaintext/ciphertext pairs provide a large multivariate polynomial system. The degrees of the equations and the number of variables will be very important for estimating the complexity of solving the polynomial system with Gröbner-basis algorithms like `F4` [Fau99] or `F5` [Fau02]. These attacks are hardly successful in general as either one of those two quantities grows rapidly with the number of rounds and since Gröbner-basis algorithms, while efficient, remain impractical in this situation.
- High-order differential cryptanalysis [Lai94; Knu95] aims at building an integral distinguisher by bounding the algebraic degree of the primitive under study: if $F : \mathbb{F}_2^n \to \mathbb{F}_2$ has degree at most d, then for all subspaces $\mathcal{V} \subseteq \mathbb{F}_2^n$ of dimension strictly bigger than d, $\sum_{\boldsymbol{u} \in \mathcal{V}} F\left(\boldsymbol{x} + \boldsymbol{u}\right) = 0$

for all $\boldsymbol{x} \in \mathbb{F}_2^n$. This family of attacks usually affects block ciphers whose round function has a relatively low algebraic degree.

- In their original form, interpolation attacks [JK97] aim at computing the coefficients of a sparse polynomial $Q \in \mathbb{F}_{2^n}[X]$ such that for all known plaintext/ciphertex pairs $(p, c) \in \mathbb{F}_{2^n}^2$, $Q(p) = c$.

  In more elaborate variants over $\mathbb{F}_2$ [Din+15], interpolation attacks leverage a family of integral distinguishers

  $$\left(\sum_{\boldsymbol{x} \in \mathcal{X}_\ell} G_i(\boldsymbol{k}, \boldsymbol{x}) = 0 \right)_\ell$$

  on one intermediate state bit G_i in the block cipher $E_{\boldsymbol{k}} = F_{\boldsymbol{k}} \circ G_{\boldsymbol{k}}$. Then by writing $\boldsymbol{y} = E_{\boldsymbol{k}}(\boldsymbol{x})$, $G_i(\boldsymbol{k}, \boldsymbol{x}) = \left(F_{\boldsymbol{k}}^{-1}\right)_i(\boldsymbol{y})$ and the ANF of $\left(F_{\boldsymbol{k}}^{-1}\right)_i$ as $\left(F_{\boldsymbol{k}}^{-1}\right)_i(\boldsymbol{y}) = \sum_{\boldsymbol{u} \in \mathbb{F}_2^n} c_{\boldsymbol{u}}(\boldsymbol{k}) \boldsymbol{y}^{\boldsymbol{u}}$, the equations

  $$\left(\sum_{\boldsymbol{u} \in \mathbb{F}_2^n} c_{\boldsymbol{u}}(\boldsymbol{k}) \cdot \sum_{\boldsymbol{y} \in E_{\boldsymbol{k}}(\mathcal{X}_\ell)} \boldsymbol{y}^{\boldsymbol{u}} = 0 \right)_\ell$$

  make a linear system in the unknown coefficients $c_{\boldsymbol{u}}(\boldsymbol{k})$. Consequently, if the number of these coefficients is low enough — i.e. the ANF of $F_{\boldsymbol{k}}^{-1}$ is sparse enough — a resolution of the system gives the values of the secret coefficients $c_{\boldsymbol{u}}(\boldsymbol{k})$ and some of them should give useful information about the key.

Cube attacks. In 2009, Dinur and Shamir proposed generic key recovery attacks, called *cube attacks* [DS09], mostly suited for stream ciphers and based on ANF analysis. Indeed, in its original version, the intermediate goal of a cube attack is to build a solvable polynomial system verified by the key with equations being of the form

$$P_{i,\boldsymbol{u}}(K) = P_{i,\boldsymbol{u}}(\boldsymbol{k}) \qquad \text{where} \qquad E_i(X, K) = \sum_{\boldsymbol{u} \in \mathbb{F}_2^n} P_{i,\boldsymbol{u}}(K) X^{\boldsymbol{u}}.$$

Statistical variants of cube attacks, *cube testers* [Aum+09] and *dynamic cube attacks* [DS11], pushed the applications of ANF analysis further but all known techniques relied on some practical computations of the Möbius transform (see Proposition 5.1). This had the major drawback of making the security analysis of stream ciphers against cube attacks only possible in the range of practically feasible computations.

Monomial trails. With the developpement of the *division property*, originally introduced by Todo in [Tod15] to build integral distinguishers, new techniques dedicated to ANF analysis emerged. In particular, their application to cube attacks, first proposed in [Tod+17], finally allowed more convincing security analysis against cube attacks beyond practical range. Following the terminology of [Hu+20], these techniques are based on the computation of *monomial trails*, which trace the existence of a given monomial in input variables $X^{\boldsymbol{u}} K^{\boldsymbol{v}}$ through the successive rounds and operations of the primitive. If monomial trails can be considered as the algebraic counterpart of differential or linear trails, there is a notable difference: when computing some ANF coefficients with monomial trails, *all* the corresponding monomial trails have to be computed.

Our contributions. In this chapter, we propose a coherent state of the art about cube attacks and we investigate a new idea for ANF analysis which could help to surpass the limit of monomial trails. In a circuit evaluating a Boolean function $F : \mathbb{F}_2^n \to \mathbb{F}_2$, replacing the addition and the multiplication of $\mathbb{F}_2$ by the addition and the multiplication of the ring of integers $\mathbb{Z}$ builds a new object, a callable function $F^{\mathbb{Z}} : \mathbb{Z}^n \to \mathbb{Z}$, whose analysis might reveal valuable information on the ANF of the original F. In particular, we adapt the Möbius transform to this new object but practical applications of this new idea have not been investigated so far.

Organization of the chapter. Section 5.2 goes through the important definitions, properties and algorithms about Boolean functions and their ANFs. Section 5.3 presents cube attacks and proposes a bird's eye view on their variants. In Section 5.5, we explore how transposing a Boolean circuit to the ring of integers could help to exploit a potential structural weakness of ANFs, low density, that could also be applied in cube attacks.

5.2 Preliminaries on Boolean functions

5.2.1 Algebraic definition of Boolean functions

Definition 5.1 (Boolean function)**.** We call a Boolean function of n variables any polynomial $P \in \mathbb{F}_2[X_0, X_1, \ldots, X_{n-1}]$.

Since the base field of Boolean functions is $\mathbb{F}_2$ and that we only intend to use them on $\mathbb{F}_2^n$ — where each scalar verifies $x^2 + x = 0$ — , Boolean functions are mostly considered modulo $X_i^2 + X_i$ for each variable X_i. This leads to the following notation for binary monomials.

Notation 5.1. For any $\boldsymbol{u} \in \mathbb{F}_2^n$, we define the binary monomial

$$X^{\boldsymbol{u}} \stackrel{\text{def}}{=} \prod_{i=0}^{n-1} X_i^{u_i}.$$

It is useful to note that for all $\boldsymbol{u}, \boldsymbol{x} \in \mathbb{F}_2^n$, $\boldsymbol{x}^{\boldsymbol{u}} = 1 \iff \boldsymbol{u} \preceq \boldsymbol{x}$.

Definition 5.2 (ANF)**.** The *Algebraic Normal Form* of a Boolean Function P is its representation as a sum of binary monomials.

$$P = \sum_{\boldsymbol{u} \in \mathbb{F}_2^n} p_{\boldsymbol{u}} X^{\boldsymbol{u}}.$$

It equivalently refers to the binary vector of length 2^n of the coefficients $p_{\boldsymbol{u}}$. The *degree* of P is then defined as

$$\deg P \stackrel{\text{def}}{=} \max \{ \operatorname{wt}(\boldsymbol{u}) \mid p_{\boldsymbol{u}} = 1 \}.$$

As a polynomial, a Boolean function P can be evulated on any vector $\boldsymbol{x} \in \mathbb{F}_2^n$ and with Notation 5.1, we explicitly have that

$$P(\boldsymbol{x}) = \sum_{\boldsymbol{u} \in \mathbb{F}_2^n} p_{\boldsymbol{u}} \boldsymbol{x}^{\boldsymbol{u}} = \sum_{\boldsymbol{u} \preceq \boldsymbol{x}} p_{\boldsymbol{u}}.$$

Definition 5.3 (Value vector)**.** The *value vector* of a Boolean function P is the binary vector of length 2^n of all values $P(\boldsymbol{x})$ when $\boldsymbol{x} \in \mathbb{F}_2^n$. It is also called the *truth table* of P.

Table 5.1 – Example of a Boolean function P of 3 variables.

(a) ANF of P

$\boldsymbol{u}$	0	1	2	3	4	5	6	7
u_0	0	1	0	1	0	1	0	1
u_1	0	0	1	1	0	0	1	1
u_2	0	0	0	0	1	1	1	1
$p_{\boldsymbol{u}}$	1	1	0	0	1	0	1	1

(b) Value vector of P

$\boldsymbol{x}$	0	1	2	3	4	5	6	7
x_0	0	1	0	1	0	1	0	1
x_1	0	0	1	1	0	0	1	1
x_2	0	0	0	0	1	1	1	1
$P(\boldsymbol{x})$	1	0	1	0	0	1	1	1

Example 5.1. The ANF vector and the value vector of the Boolean function $P \in \mathbb{F}_2[X_0, X_1, X_2]$ defined by

$$P = 1 + X_0 + X_2 + X_1X_2 + X_0X_1X_2$$

are given in Table 5.1.

The following proposition defines a very important bijection between ANFs and value vectors, which means that Boolean functions are uniquely defined by their value vector as well. It also means that we can *interpolate* — recover the coefficients of — the ANF of a Boolean function given its value vector.

Proposition 5.1 (Möbius transform)**.** *The map*

$$\Phi : \left\{ \begin{array}{ccc} \mathbb{F}_2^{2^n} & \longrightarrow & \mathbb{F}_2^{2^n} \\ \boldsymbol{a} & \longmapsto & \boldsymbol{b} \text{ with } b_{\boldsymbol{u}} = \sum_{\boldsymbol{x} \preceq \boldsymbol{u}} a_{\boldsymbol{x}} \text{ for all } \boldsymbol{u} \in \mathbb{F}_2^n \end{array} \right.$$

is an involution. It is called the Möbius transform*.*

Proof. Indeed, let $\boldsymbol{a} \in \mathbb{F}_2^{2^n}$ and $\boldsymbol{b} = \Phi(\boldsymbol{a})$. Let $\boldsymbol{u} \in \mathbb{F}_2^n$.

$$\begin{aligned} \Phi(\boldsymbol{b})_{\boldsymbol{u}} &= \sum_{\boldsymbol{x} \preceq \boldsymbol{u}} b_{\boldsymbol{x}} = \sum_{\boldsymbol{x} \preceq \boldsymbol{u}} \sum_{\boldsymbol{y} \preceq \boldsymbol{x}} a_{\boldsymbol{y}} \\ &= \sum_{\boldsymbol{y} \preceq \boldsymbol{u}} a_{\boldsymbol{y}} \cdot \#\{ \boldsymbol{x} \in \mathbb{F}_2^n \mid \boldsymbol{y} \preceq \boldsymbol{x} \preceq \boldsymbol{u} \} = \sum_{\boldsymbol{y} \preceq \boldsymbol{u}} a_{\boldsymbol{y}} \cdot 2^{\mathrm{wt}(\boldsymbol{u}+\boldsymbol{y})} \\ &= a_{\boldsymbol{u}} \text{ because } 2^{\mathrm{wt}(\boldsymbol{u}+\boldsymbol{y})} = 1 \Leftrightarrow \boldsymbol{y} = \boldsymbol{u}. \end{aligned}$$

Hence $\Phi \circ \Phi(\boldsymbol{a}) = \boldsymbol{a}$. □

5.2.2 Computing the Möbius transform

5.2.2.1 The fast Möbius transform

Consider we have the vector $\boldsymbol{a} \in \mathbb{F}_2^{2^n}$ and we want to compute the Möbius transform of $\boldsymbol{a}$. The straightforward approach would be to apply the formula of Proposition 5.1 for each output bit. Computing $b_{\boldsymbol{u}}$ costs $2^{\mathrm{wt}(\boldsymbol{u})}$ Boolean operations hence computing the entire output vector $\boldsymbol{b}$ costs

$$\sum_{i=0}^{n} \binom{n}{i} 2^i = 3^n \text{ operations.}$$

Algorithm 11 Möbius transform algorithms.

Recursive Möbius transform

```
1: function Φ(a, n)                                              ▷ a ∈ F_2^{2^n}
2:     if n = 0 then
3:         return a
4:     else
5:         b_left ← Φ(a_left, n − 1)                  ▷ a_left ≝ (a_0, a_1, ..., a_{2^{n−1}−1})
6:         b_right ← b_left + Φ(a_right, n − 1)
7:         return ( b_left | b_right )
```

In-place imperative Möbius transform

```
1: procedure Φ(a, n)                                             ▷ a ∈ F_2^{2^n}
2:     for k from 1 to n do                                      ▷ k is the depth.
3:         for i from 0 to 2^{n−k} − 1 do          ▷ i is the index of a 2^k-size chunk of a
4:                                  ▷ Add the left-hand side of chunk i to its right-hand side
5:             start ← i · 2^k
6:             for j from start to start + 2^{k−1} − 1 do
7:                 b_{j+2^{k−1}} ← b_{j+2^{k−1}} + b_j
```

However, we can use the recursive divide-and-conquer approach sketched in Algorithm 11 (more details can be found in [Jou09, Chapter 9]). When written as an in-place imperative algorithm we get a complexity of $n \cdot 2^{n-1}$ `xor` operations.

5.2.2.2 Partial Möbius transform

Let P be a Boolean function. We have access to an oracle that evaluates P on input $\boldsymbol{x} \in \mathbb{F}_2^n$. If n is too large, we cannot store the ANF vector of P of length 2^n but we could still be interested in computing a small part of it. More precisely, if $\boldsymbol{u} \in \mathbb{F}_2^n$, we want to compute the coefficients $p_{\boldsymbol{v}}$ for all $\boldsymbol{v}$ such that $\boldsymbol{u} \preceq \boldsymbol{v}$ under the assumption that we can store $2^{n-\mathrm{wt}(\boldsymbol{u})}$ bits.

We note $m \stackrel{\text{def}}{=} \mathrm{wt}\,(u)$ and we suppose without loss of generality that $u_i = 1 \Leftrightarrow i \leqslant m-1$. Our problem is equivalent to computing the ANF of the Boolean function $P_{\boldsymbol{u}} \in \mathbb{F}_2[X_{m,\dots,n-1}]$ defined by

$$P_{\boldsymbol{u}} = \sum_{\boldsymbol{v}:\ \boldsymbol{u} \preceq \boldsymbol{v}} p_{\boldsymbol{v}} X^{\boldsymbol{u}+\boldsymbol{v}} \quad \text{or} \quad P = P_{\boldsymbol{u}} \cdot X^{\boldsymbol{u}} + \sum_{\boldsymbol{w}:\ \boldsymbol{u} \not\preceq \boldsymbol{w}} p_{\boldsymbol{w}} X^{\boldsymbol{w}}.$$

First we can verify that if there exists i such that $u_i = 1$ and $w_i = 0$, i.e. $\boldsymbol{u} \not\preceq \boldsymbol{w}$,

$$\begin{aligned}
\sum_{\boldsymbol{u}' \in \mathbb{F}_2^m} (\,\boldsymbol{u}' \mid X_{m,\dots,n-1}\,)^{\boldsymbol{w}} &= \sum_{\boldsymbol{u}':\ u'_i=0} (\,\boldsymbol{u}' \mid X_{m,\dots,n-1}\,)^{\boldsymbol{w}} + \sum_{\boldsymbol{u}':\ u'_i=1} (\,\boldsymbol{u}' \mid X_{m,\dots,n-1}\,)^{\boldsymbol{w}} \\
&= \sum_{\boldsymbol{u}':\ u'_i=0} (\,\boldsymbol{u}' \mid X_{m,\dots,n-1}\,)^{\boldsymbol{w}} + (\,\boldsymbol{u}' + \boldsymbol{e}_i \mid X_{m,\dots,n-1}\,)^{\boldsymbol{w}} \\
&= 0 \text{ since } w_i = 0.
\end{aligned}$$

Then we have the following relation.

$$\begin{aligned}\sum_{\boldsymbol{u}'\in\mathbb{F}_2^m} P\left(\boldsymbol{u}' \mid X_{m,\ldots,n-1}\right) &= \sum_{\boldsymbol{u}'\in\mathbb{F}_2^m}\left(P_{\boldsymbol{u}}\cdot \overbrace{\left(\boldsymbol{u}' \mid X_{m,\ldots}\right)^{\boldsymbol{u}}}^{1 \text{ if } \boldsymbol{u}'=(1,\ldots,1), \text{ else } 0} + \sum_{\boldsymbol{w}:\ \boldsymbol{u}\not\preceq\boldsymbol{w}} p_{\boldsymbol{w}}\cdot\left(\boldsymbol{u}' \mid X_{m,\ldots}\right)^{\boldsymbol{w}}\right)\\ &= P_{\boldsymbol{u}} + \sum_{\boldsymbol{w}:\ \boldsymbol{u}\not\preceq\boldsymbol{w}} p_{\boldsymbol{w}} \underbrace{\sum_{\boldsymbol{u}'\in\mathbb{F}_2^m}\left(\boldsymbol{u}' \mid X_{m,\ldots}\right)^{\boldsymbol{w}}}_{0}\\ &= P_{\boldsymbol{u}}(X_m,\ldots,X_{n-1}).\end{aligned}$$

Therefore we can call the oracle that evaluates P 2^m times to evaluate $P_{\boldsymbol{u}}$. We can then compute the entire value vector of $P_{\boldsymbol{u}}$ of length 2^{n-m} with 2^n evaluations of P. Then the fast Möbius transform computes the ANF vector of $P_{\boldsymbol{u}}$ in $(n-m)2^{n-m-1}$ operations. This means that we actually use the oracle to compute the entire value vector of P but since it is too large to be stored, we have to perform the first steps of the Möbius transform online.

5.2.2.3 Parallel algorithm on shared memory

When several processes have access to a shared memory, it can be useful to adapt the imperative procedure of Algorithm 11. Let 2^m be the number of processes with $m \leqslant n$.

Basically, during the fast Möbius transform, processes just have to perform `xor` operations on halves of chunks. At step k in Algorithm 11, chunks have size 2^k. Then there are two cases.

- $k \leqslant n-m$. The number of chunks of size 2^k, 2^{n-k}, is then bigger than the number of processes 2^m. Hence processes can operate independently on independent chunks. Each process will serially `xor` left-hand sides on right-hand sides of 2^{n-k-m} chunks.

- $k \geqslant n-m$. The number of chunks is smaller than the number of processes. If we want to continue taking advantage of all the processes, we will rather operate on chunks one after another: for each chunk — of size $2^k > 2^{n-m}$ — each process will `xor` $2^{k-n+m-1}$ bits of the left-hand side to the $2^{k-n+m-1}$ corresponding bits of the right-hand side.

This gives Algorithm 12. Of course this algorithm can be optimized: the two cases can be easily adapted to any number of processes (some will just have a bit more work than others). Finally, the first steps in the second case could be easily written as more cache-friendly with a hybrid approach with the first case.

5.3 Cube attacks

In the context of a symmetric cryptographic scheme

$$E : \left\{\begin{array}{rcl}\mathbb{F}_2^n \times \mathbb{F}_2^m & \longrightarrow & \mathbb{F}_2^s\\ (\boldsymbol{x},\boldsymbol{k}) & \longmapsto & (E_0(\boldsymbol{x},\boldsymbol{k}),\ldots,E_{s-1}(\boldsymbol{x},\boldsymbol{k}))\end{array}\right. , \quad E_i = \sum_{\boldsymbol{u}\in\mathbb{F}_2^n} P_{i,\boldsymbol{u}}(K_{0,\ldots,m-1})X^{\boldsymbol{u}},$$

Algorithm 12 Parallel Möbius transform on shared memory.

1: **procedure** $\Phi(\boldsymbol{a}, n,$ processes $p_0, \ldots, p_{2^{m-1}-1})$ ▷ $\boldsymbol{a} \in \mathbb{F}_2^{2^n}, m \leqslant n$
2: **for** k from 1 to $n-m$ **do**
3: **for** $\ell \in [0, 2^{m-1}-1]$, in parallel for each process p_ℓ **do**
4: **for** i from $\ell \cdot 2^{n-k}$ to $(\ell+1)\cdot 2^{n-k}-1$ **do**
5: Same as lines 4 – 7 in Algorithm 11.
6: **for** k from $n-m+1$ to n **do**
7: **for** i from 0 to $2^{n-k}-1$ **do**
8: **for** $\ell \in [0, 2^{m-1}-1]$, in parallel for each process p_ℓ **do**
9: $\texttt{start} \leftarrow i \cdot 2^k + \ell \cdot 2^{k-n+m}$
10: **for** j from $\texttt{start}$ to $\texttt{start} + 2^{k-n+m-1} - 1$ **do**
11: $b_{j+2^{k-n+m-1}} \leftarrow b_{j+2^{k-n+m-1}} + b_j$

cube attacks aim at finding a set $\mathcal{S}$ of pairs $(i, \boldsymbol{u})$ where $i \in [0, s-1]$ is an output coordinate and $\boldsymbol{u} \in \mathbb{F}_2^n$ is the exponent of a monomial in public variables, such that we can solve the polynomial system

$$P_{i,\boldsymbol{u}}(K_{0,\ldots,m-1}) = P_{i,\boldsymbol{u}}(\boldsymbol{k}), \quad (i,\boldsymbol{u}) \in \mathcal{S}$$

with m unknown variables $K_0, \ldots, K_{m-1}$ and a secret value $\boldsymbol{k}$ fixed. More precisely, cube attacks are chosen plaintext/IV attacks that have two phases.

1. The *preprocessing* phase, also called the *offline* phase, consists in finding a nice set $\mathcal{S}$ of pairs $(i, \boldsymbol{u})$ and in computing the corresponding polynomials $P_{i,\boldsymbol{u}}(K)$.

2. The *online* phase consists of a game where the challenger chooses a private value $\boldsymbol{k}$ and answers to queries $\boldsymbol{x} \mapsto E(\boldsymbol{x}, \boldsymbol{k})$. We have seen in Section 5.2.2.2 that for all i and $\boldsymbol{u}$,

$$P_{i,\boldsymbol{u}}(\boldsymbol{k}) = \sum_{\boldsymbol{x} \preceq \boldsymbol{u}} E_i(\boldsymbol{x}, \boldsymbol{k}).$$

   Hence, with the allowed queries of the online phase, the adversary can compute the value $P_{i,\boldsymbol{u}}(\boldsymbol{k})$ even if $\boldsymbol{k}$ remains secret. However, this computation takes $2^{\mathrm{wt}(\boldsymbol{u})}$ queries and `xor` operations. The adversary then has to find the secret $\boldsymbol{k}$ by solving the polynomial system.

Remark 5.1. In practice, solving the polynomial system does not give a unique solution but rather gives a search space much smaller than the original key space. For example, if the polynomial system does not have enough linear or balanced equations — which are interesting because once the value ($P_{i,\boldsymbol{u}}(\boldsymbol{k})$ here) of a balanced equation is known, half the keys are discarded. The attacker will then have to complete the online phase with an exhaustive search in that smaller search space.

The dual approach to cube attacks would be to find a set $\mathcal{S}'$ of pairs $(i, \boldsymbol{x})$ such that we can solve the polynomial system

$$E_i(\boldsymbol{x}, K_{0,\ldots,m-1}) = E_i(\boldsymbol{x}, \boldsymbol{k}), \quad (i,\boldsymbol{x}) \in \mathcal{S}'$$

for a fixed secret value $\boldsymbol{k}$. It is unfortunately impractical because the polynomials $E_i(\boldsymbol{x}, K)$ that compose this system are too big to allow solving the system with complexity less than exhaustive search or even too big to be stored. This is why in cube attacks we accept the trade-off of those $2^{\mathrm{wt}(\boldsymbol{u})}$ queries to try to build systems with simpler polynomials.

Therefore, a basic requirement for a cube attack to be practical is that we have to be able to compute the ANF of $P_{i,\boldsymbol{u}}$ which can have as many as 2^m monomials in $K_0, \ldots, K_{m-1}$. Moreover, even when the polynomials occuring in the system are entirely known, solving an arbitrary system of polynomial equations over $\mathbb{F}_2$ is very hard. For example, solving quadratic systems with N unknowns over $\mathbb{F}_2$ is an NP-complete problem even if the size of the system is $O(N^2)$ [Bou+10]. When cube attacks were introduced in [DS09] by Dinur and Shamir, they searched for polynomials $P_{i,\boldsymbol{u}}$ of degree 1 to get a solvable linear system. The next section explains parts of this original work.

5.3.1 Cube attacks on black-box polynomials

Dinur and Shamir published the first cube attacks in 2009 [DS09]. They applied them in a specific setting they called "tweakable black-box polynomial" in which the internal details of a cipher E are hidden to the adversary — hence the "black box" — but in which the adversary has the possibility to query the value $E(\boldsymbol{x}, \boldsymbol{k})$ for a chosen pair $(\boldsymbol{x}, \boldsymbol{k})$ during the preprocessing phase. The polynomials $E_i(X, K)$ are then said "tweakable" because giving values $\boldsymbol{x}$ to the public variables X gives many different polynomials $E_i(\boldsymbol{x}, K)$, all derived from the original $E_i(X, K)$. In order to show a generic practical setting, they also require that the polynomials $E_i(X, K)$ under study have a degree d in public and private variables X and K of practical size $d \lessapprox 40$. Since $E(X, K)$ is given as a black box, the exact value of d is unknown. Since the ideas presented here do not depend on which coordinate of $E(X, K)$ we work on, we denote by $F(X, K)$ one of the Boolean functions $E_i(X, K),\ i \in [0, s-1]$.

We consider in the rest of this section that $F(X, K)$ is an unknown Boolean function of unknown degree $d \lessapprox 40$. In this setting, the (intermediate) goal of the adversary is to find linear expressions $L(K)$ such that, from queries $\boldsymbol{x} \mapsto F(\boldsymbol{x}, \boldsymbol{k})$, she can compute $L(\boldsymbol{k})$ for the unknown value $\boldsymbol{k}$ chosen by the challenger. This allows the adversary to build a linear system with equations $L(K_0, \ldots, K_{m-1}) = L(\boldsymbol{k})$. The adversary is allowed to make queries $(\boldsymbol{x}, \boldsymbol{k}) \mapsto F(\boldsymbol{x}, \boldsymbol{k})$ during the preprocessing — before the challenger chooses a random value $\boldsymbol{k}$.

The authors introduce in this work the following terminology. We have used a vector-based notation from the beginning of this chapter but it is sometimes easier to understand the following concepts with an index-based notation: if $\boldsymbol{u} \in \mathbb{F}_2^n$ compactly represents the monomial $X^{\boldsymbol{u}}$, sometimes working with the indices in $\operatorname{supp}(\boldsymbol{u})$ is easier.

Definition 5.4 (Cube)**.** *Cube variables* are public variables chosen by the adversary. The set of their indices will often be denoted by $\mathcal{I}$. *Static variables* are the other public variables, to which a (Boolean) value is given by the adversary. The index set of static variables with value 1 will often be denoted by $\mathcal{J}$. Of course we must have $\mathcal{I} \cap \mathcal{J} = \emptyset$.

The *Cube* $\mathcal{C}$ of $(\mathcal{I}, \mathcal{J})$ is the set $\boldsymbol{e}_{\mathcal{J}} + \operatorname{Prec}(\boldsymbol{e}_{\mathcal{I}})$, i.e. the set

$$\{\, \boldsymbol{x} \in \mathbb{F}_2^n \mid x_i = 1 \text{ if } i \in \mathcal{J}, x_i = 0 \text{ if } i \notin \mathcal{I} \cup \mathcal{J} \,\}.$$

The dimension of $\mathcal{C}$ is $|\mathcal{I}|$.

Example 5.2. Consider the function $H : \mathbb{F}_2^5 \to \mathbb{F}_2$ from Section 1.3.4 with the first three variables public and the last two private.

$$H(X_0, X_1, X_2, K_0, K_1) = K_1 + K_0K_1 + X_0K_0 + X_1 + X_2(K_0 + K_0K_1) + X_0X_2(K_0 + K_1) + X_1X_2K_1 + X_0X_1X_2.$$

The cube $\mathcal{C}_1$ of $(\mathcal{I}_1, \mathcal{J}_1)$ with $\mathcal{I}_1 \stackrel{\text{def}}{=} \{2\}$ and $\mathcal{J}_1 \stackrel{\text{def}}{=} \{0\}$ is the set $\{(1,0,0),(1,0,1)\}$. The cube $\mathcal{C}_2$ of $(\mathcal{I}_2, \mathcal{J}_2)$ with $\mathcal{I}_2 \stackrel{\text{def}}{=} \{0,2\}$ and $\mathcal{J}_2 \stackrel{\text{def}}{=} \emptyset$ is the set $\{(0,0,0),(1,0,0),(0,0,1),(1,0,1)\}$.

Remark 5.2. Since the set $\boldsymbol{e}_{\mathcal{J}} + \text{Prec}\,(\boldsymbol{e}_{\mathcal{I}})$ characterizes $(\mathcal{I}, \mathcal{J})$, the word cube can also point to the pair $(\mathcal{I}, \mathcal{J})$.

Definition 5.5 (Superpoly)**.** Given a cube $\mathcal{C} = (\mathcal{I}, \mathcal{J})$, its *superpoly* in $F(X, K)$ is the polynomial

$$P_{\mathcal{C}}(K) = \sum_{\boldsymbol{x} \in \boldsymbol{e}_{\mathcal{J}} + \text{Prec}(\boldsymbol{e}_{\mathcal{I}})} F(\boldsymbol{x}, K).$$

Another notation useful for Section 5.3.2 and Section 5.4.5 is

$$\sigma_{\mathcal{C}}^{-1} : \left\{ \begin{array}{rcl} \boldsymbol{e}_{\mathcal{J}} + \text{Prec}\,(\boldsymbol{e}_{\mathcal{I}}) & \longrightarrow & \mathbb{F}_2^{|\mathcal{I}|} \\ \boldsymbol{x} & \longmapsto & \boldsymbol{y} \text{ where } y_\ell = x_{i_\ell} \text{ if } i_\ell \in \mathcal{I}. \end{array} \right.$$

Then

$$P_{\mathcal{C}}(K) = \sum_{\boldsymbol{y} \in \mathbb{F}_2^{|\mathcal{I}|}} F(\sigma_{\mathcal{C}}(\boldsymbol{y}), K).$$

Remark 5.3. With the above definition, we see that the superpoly of the cube $\text{Prec}\,(\boldsymbol{u})$ in $E_i(X, K)$ is the polynomial $P_{i,\boldsymbol{u}}(K)$ defined in the introduction of Section 5.3. In practice, the adverary will give the value 0 by default to static variables as we expect the superpoly to be simpler that way. Indeed, $P_{\mathcal{C}}(K) = \sum_{\boldsymbol{w} \preceq \boldsymbol{e}_{\mathcal{J}}} P_{\text{Prec}(\boldsymbol{e}_{\mathcal{I}} + \boldsymbol{w})}$. But there can be exceptions for which fixing some variables to the value 1 is interesting, as explained in the paragraph Section 5.3.1.4.

Definition 5.6. $(\mathcal{I}, \mathcal{J})$ is called a *maxterm* of $F(X, K)$ when $\deg P_{\mathcal{C}}(K) = 1$, i.e. when the cube gives a non-constant affine superpoly.

Example 5.3. Going back to Example 5.2, we have that the superpolys of the cubes $\mathcal{C}_1$ and $\mathcal{C}_2$ are respectively $P_{\mathcal{C}_1}(K) = K_0 + K_0K_1 + K_0 + K_1 = K_1 + K_0K_1$ and $P_{\mathcal{C}_2}(K) = K_0 + K_1$. Both $\mathcal{C}_1$ is not a maxterm but $\mathcal{C}_2$ is a maxterm. Since the superpoly of $\mathcal{C}_3 \stackrel{\text{def}}{=} (\{0,1\}, \emptyset)$ is the constant 0, $\mathcal{C}_3$ is not a maxterm.

5.3.1.1 Finding maxterms

Maxterms will have to be found and their superpolys will have to be computed during the preprocessing phase for building a system of linear equations in the secret variables. The idea of Dinur and Shamir is that if $F(X, K)$ has degree d in the variables $X_0, \ldots, X_{n-1}, K_0, \ldots, K_{m-1}$ then for any cube $\mathcal{C}$ of dimension $d-1$, $\deg P_{\mathcal{C}}(K) \leqslant 1$. Therefore, as long as 2^d queries are possible, there is hope to find maxterms — from cubes of dimension $d-1$ — and to compute their degree 1 superpolys. There is a subtlety here that if $\deg P_{\mathcal{C}} = 0$, it will not give any information on the private variables and will not be useful for building the linear system. Since the attacker does not know d, the authors propose the following steps for estimating it in the preprocessing phase. It is based on the linearity test BLR [BLR93] and needs a starting guess for d: $h \in [1, n-1]$. We give more details on the BLR test in Section 5.3.2.1.

1. Choose $\mathcal{I} \subset [0, n-1]$ at random with $\#\mathcal{I} = h - 1$ and fix $\mathcal{J} = \emptyset$.

2. Run a linearity BLR test on $P_{\mathcal{C}}(K)$ where you can evaluate $\boldsymbol{k} \mapsto P_{\mathcal{C}}(\boldsymbol{k}) = \sum_{\boldsymbol{x} \preceq e_{\mathcal{I}}} F(\boldsymbol{x}, \boldsymbol{k})$ with queries $(\boldsymbol{x}, \boldsymbol{k}) \mapsto F(\boldsymbol{x}, \boldsymbol{k})$.

3. The results of the test can be interpreted as follows.

   - If $h < d-1$, then $P_{\mathcal{C}}(K)$ is likely to have degree strictly greater than 1 and fail the linearity test.
   - If $h > d-1$, $P_{\mathcal{C}}(K)$ will be constant. The linearity test will pass but the queries used for the test allow to detect that case.
   - If the linearity test passes and $P_{\mathcal{C}}$ is not constant, you can assume $h = d-1$ and a maxterm $\mathcal{I}$ has been found.

   Therefore, there are different actions depending on the output of the test.

   - If the test fails, choose $i \notin \mathcal{I}$, replace $\mathcal{I}$ by $\mathcal{I} \cup \{i\}$ and go back to Step 2.
   - If the test gives constant, choose $i \in \mathcal{I}$, replace $\mathcal{I}$ by $\mathcal{I} \backslash \{i\}$ and go back to Step 2. It is important to vary the consecutive choices of index in this Step 3 to avoid getting stuck to the pathological case where the polynomial $P_{\mathcal{C}}(K)$ is constant despite $\#\mathcal{I} = d-1$. The authors note that in this pathological case, it can be useful to use a cube $(\mathcal{I}, \mathcal{J})$ with $\mathcal{J} \neq \emptyset$ to make the superpoly non-zero.
   - If the test passes and gives non constant, return $\#\mathcal{I} + 1$ as a good guess for the degree d and the maxterm $\mathcal{I}$.

Once d is known, the attacker can easily compute maxterms and their superpolys, build an invertible linear system of equations in private variables and even precompute the inverse matrix for solving the system. Of course the degree d, the number of variables n and m should allow the existence of enough maxterms to build such a system.

Finally, we note that this generic setting with $F(X, K)$ having a practically low degree d allows to be confident that there exist maxterms of dimension at most $d-1$ and that the steps above will explicit one of them. However, one can easily imagine that maxterms of practical dimension h exist ($h \lessapprox 40$) even if the degree d is too high for 2^{d-1} to be practical. This means that finding maxterms is in fact the major challenge of cube attacks. In that respect, maxterms in cube attacks play a similar role as good-probability characteristics in differential/linear cryptanalysis. There is a major difference though: it is impossible at that point to have any information on maxterms and superpolys beyond the practical range since they need to be explicitly computed. This means cube attacks are purely experimental attacks and fail to provide a meaningful security analysis of a non-black-box cipher if the attacker cannot find maxterms experimentally. This caveat has begun to find answers in 2017 with the use of division property to get information on superpolys [Tod+17]. We will elaborate on this technique in Section 5.4.5. We will see in Section 5.3.1.4 the first ideas that have been explored to guess cubes of maxterms when the polynomials are not black box.

5.3.1.2 Block or stream ciphers for cube attacks

Cube attacks can theoretically be applied to block ciphers to mount chosen-plaintext key-recovery attacks: public variables are plaintext variables and private variables are key variables. Actually, a cube attack on a block cipher would be very similar to a high-order differential attack. However the authors of [DS09] note that in a block cipher, the degree grows exponentially and a practical

application is hard to obtain. Therefore they prefered applying cube attacks to stream ciphers to mount chosen-IV attacks where public variables are IV variables. In particular, they successfully applied it on Trivium with 767 initialization rounds out of 1152. They found 35 maxterms of dimension at most 31. Since the key has 80 bits, the corresponding superpolys define a vector space of solutions of dimension 45. The overall complexity is then dominated by the exhaustive search for the 2^{45} remaining possibilities.

5.3.1.3 Directions

In Appendix A of [DS09], Dinur and Shamir propose a few directions for future work. They write that attackers cannot only recognize affine superpolys but also quadratic superpolys, and more generally low-degree superpolys with the tests given in [Alo+03]. This direction has been followed by Mroczkowski and Szmidt who used linearity and quadraticity tests to mount a key recovery on Trivium with 709 initialization rounds [MS11] and by Fouque and Vannet who also interpolated quadratic superpolys to mount key recoveries on Trivium with up to 799 initialization rounds [FV14]. Dinur and Shamir also write that the attacker can exploit any non-linear superpoly he can find and compactly represent. This last idea is at the heart of the work presented in Section 5.5.1.

5.3.1.4 Choosing cubes for non-black-box polynomials

In a subsequent work [Aum+09], Aumasson, Dinur, Meier and Shamir mounted a key-recovery attack on 14 rounds of the hash function MD6 with 2^{22} queries and operations and showed how to use the knowledge on the circuit that evaluates the polynomials to choose the cubes carefully.

Let us assume that we know a decomposition

$$E(X,K) = E'\left(f_{\text{mix}}(X_{<n'},K) + f_{\text{pub}}(X_{\geqslant n'}) + f_{\text{key}}(K) + (1+X_0)\cdot g(X,K)\right)$$

thanks to the knowledge of the circuit that evaluates E, where

$$f_{\text{mix}} : \mathbb{F}_2^{n'+m} \to \mathbb{F}_2^{s'}, \quad f_{\text{pub}} : \mathbb{F}_2^{n-n'} \to \mathbb{F}_2^{s'}, \quad f_{\text{key}} : \mathbb{F}_2^{m} \to \mathbb{F}_2^{s'}, \quad g : \mathbb{F}_2^{n+m} \to \mathbb{F}_2^{s'},$$

describe the initial diffusion rounds and $E' : \mathbb{F}_2^{s'} \to \mathbb{F}_2^{s}$ is the rest of the circuit. The authors give two ideas to make a good guess for maxterms.

- The cube variables should not mix with the private variables in the initial rounds. In our example, this means the cube variables should have indices in $[n', n-1]$.
- The static variables should be assigned constant values that help limiting the diffusion. In our example, this means that X_0 should be set to 1.

With our notation, this means that good guesses for cubes should be of the form $\boldsymbol{e}_{\mathcal{J}} + \text{Prec}\left(\boldsymbol{e}_{\mathcal{I}}\right)$ with $\mathcal{J} = \{0\}$ and $\mathcal{I} \subseteq [n', n-1]$. The authors used the same ideas to find cubes for competitive cube-based distinguishers on reduced-round Trivium. They used those cubes with a technique they named *cube testers*.

5.3.2 Cube testers

In Section 5.3.1 the BLR linearity test was used in the preprocessing phase to detect the linearity of superpolys in private variables. The other main idea of [Aum+09] was to use property-testing

Algorithm 13 BLR linearity test [BLR93].

input
    Oracle that answers queries $\boldsymbol{x} \mapsto F(\boldsymbol{x})$
    ϵ: distance parameter.
output
    `true` with probability 1 if F is linear;
    `false` with probability higher than $2/3$ if F is ϵ-far from linear functions;

for i from 1 to $\lceil \epsilon^{-1} \rceil$ **do**
    Draw $(\boldsymbol{x}, \boldsymbol{y})$ uniformly at random in $\mathbb{F}_2^n \times \mathbb{F}_2^n$.
    Query $F(\boldsymbol{x})$ and $F(\boldsymbol{y})$.
    if $F(\boldsymbol{x}+\boldsymbol{y}) \neq F(\boldsymbol{x}) + F(\boldsymbol{y})$ **then**
        return `false`
return `true`

on superpolys to detect a non-random behaviour of the cipher or to mount distinguishers. Indeed, linearity or even low degree are very strong properties and outside the generic setting of a practical-degree polyomial, such superpolys might not even exist and, if they exist, will be very hard to find. Hence the idea of testing more common properties which are rare enough not to be verified by a random Boolean function. We first explain the basics of property testing before explaining how cube testers work in Section 5.3.2.3.

5.3.2.1 Property testing

For any two functions $F, G : \mathbb{F}_2^n \to \mathbb{F}_2$, their relative distance is defined by

$$\mathrm{d}(F, G) = \frac{|\{\, \boldsymbol{x} \in \mathbb{F}_2^n \mid F(\boldsymbol{x}) \neq G(\boldsymbol{x}) \,\}|}{2^n}.$$

Let $\epsilon \in]0, 1[$. For a function F and a family of functions $\mathcal{P}$, we say that F is ϵ-far from $\mathcal{P}$ if

$$\forall G \in \mathcal{P},\ \mathrm{d}(F, G) > \epsilon.$$

With a query access to a function F and a distance parameter ϵ, a *tester* for the family $\mathcal{P}$ is a probabilistic algorithm which returns `true` with probability at least $2/3$ if $F \in \mathcal{P}$ and `false` with probability at least $2/3$ if F is ϵ-far from $\mathcal{P}$. When the tester returns `true` with probability 1 if $F \in \mathcal{P}$, it is called a *one-sided* tester. The BLR test (Algorithm 13) is a one-sided tester for linear functions with query complexity $O(\epsilon^{-1})$.

As explained in [Aum+09], the following properties for Boolean functions are similarly testable.

- Balance: $|\{\, \boldsymbol{x} \in \mathbb{F}_2^n \mid F(\boldsymbol{x}) = 0 \,\}| = 2^{n-1}$.
- Constantness: $|\{\, \boldsymbol{x} \in \mathbb{F}_2^n \mid F(\boldsymbol{x}) = 0 \,\}| \in \{0, 2^n\}$.
- Low degree: $\deg F \leqslant d$ (with the tester of [Alo+03]).
- Presence of linear or neutral variables: respectively

$$\exists i :\ F(X) = X_i + \sum_{\boldsymbol{u} \in \mathbb{F}_2^n :\ u_i = 0} p_{\boldsymbol{u}} X^{\boldsymbol{u}} \qquad \text{and} \qquad \exists i :\ F(X) = \sum_{\boldsymbol{u} \in \mathbb{F}_2^n :\ u_i = 0} p_{\boldsymbol{u}} X^{\boldsymbol{u}}.$$

5.3.2.2 Testing superpolys for non randomness

In [Aum+09], the authors used property-testing on superpolys to observe the non-randomness of a Boolean function given as an oracle. Let $F \in \mathbb{F}_2[X_0, \ldots, X_{n-1}]$. In this setting, the variables are all public and as in Section 5.3.1, we can choose cube variables and static variables among them. But to define a superpoly on such a cube $\mathcal{C}$, we will need to specify which variables it has since private variables do not exist anymore. The authors hence introduce a third category of variables called *superpoly variables* whose indices define a new set $\mathcal{S}$. To be more explicit, we choose without loss of generality the variables $X_0, \ldots, X_{m-1}$ for some $m < n$, i.e. $\mathcal{S} = [0, m-1]$. We also have to choose cube variables with indices in $[m, n-1]$. Those indices define the set $\mathcal{I}$. Finally, we can write that

$$F(X) = P(X_{0,\ldots,m-1}) \cdot X^{\boldsymbol{e}_{\mathcal{I}}} + \sum_{\boldsymbol{w}:(0|\boldsymbol{e}_{\mathcal{I}}) \not\preceq \boldsymbol{w}} a_{\boldsymbol{w}} X^{\boldsymbol{w}},$$
$$\text{and } P(X_{0,\ldots,m-1}) = \sum_{\boldsymbol{x} \preceq \boldsymbol{e}_{\mathcal{I}}} F\left(X_{0,\ldots,m-1} | \boldsymbol{x}\right).$$

This defines the superpoly P in the public variables $X_0, \ldots, X_{m-1}$. We can of course fix non-zero values for public variables outside cube and superpoly variables if relevant. We sum that up in the following formal definition of superpolys.

Definition 5.7 (Superpoly of public variables)**.** Let $\mathcal{S} \subset [0, n-1]$ be a set of indices of public variables. Let $\mathcal{C} = (\mathcal{I}, \mathcal{J})$ be a cube such that $\mathcal{S} \cap (\mathcal{I} \cup \mathcal{J}) = \emptyset$. In order to organize the public variables with respect to the sets of $\mathcal{C}$ and $\mathcal{S}$, we need

$$\sigma^{-1}_{\mathcal{C},\mathcal{S}} : \left\{ \begin{array}{rcl} \boldsymbol{e}_{\mathcal{J}} + \mathrm{Prec}\left(e_{\mathcal{I}\cup\mathcal{S}}\right) & \longrightarrow & \mathbb{F}_2^{|\mathcal{I}|} \times \mathbb{F}_2^{|\mathcal{S}|} \\ \boldsymbol{x} & \longmapsto & (\boldsymbol{y}, \boldsymbol{z}) \text{ where } y_\ell = x_{i_\ell}, i_\ell \in \mathcal{I} \text{ and } z_\ell = x_{j_\ell}, j_\ell \in \mathcal{S} \end{array} \right. .$$

The *superpoly* $(\mathcal{C}, \mathcal{S})$ in the Boolean function $F(X)$ is the polynomial

$$P_{\mathcal{C},\mathcal{S}}(X_{\mathcal{S}}) \stackrel{\text{def}}{=} \sum_{\boldsymbol{y} \in \mathbb{F}_2^{|\mathcal{I}|}} F\left(\sigma_{\mathcal{C},\mathcal{S}}(\boldsymbol{y}, X_{\mathcal{S}})\right).$$

When the context is clear, we can write $P_{\mathcal{C}}(X_{\mathcal{S}})$.

The strategy of Aumasson, Dinur, Meier and Shamir to observe the non randomness of $F(X)$ is to find a cube $\mathcal{C}$ and a superpoly $P_{\mathcal{C}}(X_{\mathcal{S}})$ for some set of variables $\mathcal{S}$ that verify a testable property $\mathcal{P}$.

Definition 5.8 (Cube tester)**.** A cube tester for a Boolean function $F(X_0, \ldots, X_{n-1})$ is the collection of a superpoly of public variables in F, $P_{\mathcal{C}}(X_{\mathcal{S}})$, and a testable property $\mathcal{P}$ such that $P_{\mathcal{C}}(X_{\mathcal{S}})$ verifies $\mathcal{P}$ whereas a random Boolean function $f(X_{\mathcal{S}})$ does not verify $\mathcal{P}$ with good probability.

For example, they found many coordinate functions in a round-reduced version of `MD6` for which a carefully chosen superpoly — with the ideas described in Section 5.3.1.4 — is strongly unbalanced. Moreover, they experimented the same ideas in a non-realistic scenario on reduced-round `Trivium`: they allowed themselves to choose some superpoly variables among key variables and fixed the rest of the key either to a random value, either to zero which is much weaker. They found for example that the key bits $0, 3$ and 4 are neutral in the superpoly $P_{\mathcal{C}}(X_{[0,79]\setminus\mathcal{C}}, K_{0,3,4})$ after 885 initialisation rounds with the other key bits fixed to 0 and a cube $\mathcal{C}$ of dimension 27.

The authors then went to a more realistic scenario for ciphers to mount distinguishers based on cube testers.

5.3.2.3 Distinguishers with keyed superpolys

In Section 5.3.1, these attacks needed a difference between public and private variables. To get polynomial equations in the private variables, some public variables would then be used as cube variables. For distinguishers based on cube testers, we will not try to get polynomial equations in the private variables but we will test a property on a keyed Boolean function of some public variables. As in Section 5.3.2.2, the idea is then to create a third category of public variables — the superpoly variables — that will play the role of private variables. More precisely, we now consider $F(X, \boldsymbol{k})$ as a keyed Boolean function, i.e. a member of the family of Boolean functions $\{ F(X, \boldsymbol{k}) \in \mathbb{F}_2^n \mid \boldsymbol{k} \in \mathbb{F}_2^m \}$ rather than a polynomial $F(X, K)$ with private variables. A keyed superpoly is then defined as a superpoly of public variables in the keyed function $F(X, \boldsymbol{k})$:

$$P_{\mathcal{C},\mathcal{S}}(X_{\mathcal{S}}, \boldsymbol{k}) \stackrel{\text{def}}{=} \sum_{\boldsymbol{y} \in \mathbb{F}_2^{|\mathcal{I}|}} F\left(\sigma_{\mathcal{C},\mathcal{S}}(\boldsymbol{y}, X_{\mathcal{S}}), \boldsymbol{k}\right)$$

is the keyed superpoly in variables $X_{\mathcal{S}}$ of the cube $\mathcal{C}$.

To mount a distinguisher on the family $\{ F(X, \boldsymbol{k}) \mid \boldsymbol{k} \in \mathbb{F}_2^m \}$, the idea of Aumasson, Dinur, Meier and Shamir is to find a superpoly $P_{\mathcal{C},\mathcal{S}}$ such that testing a well-chosen property provides a distinguisher on the family $\{ P_{\mathcal{C}}(X_{\mathcal{S}}, \boldsymbol{k}) \mid \boldsymbol{k} \in \mathbb{F}_2^m \}$. More precisely, the tested property $\mathcal{P}$ and the keyed superpoly $P_{\mathcal{C},\mathcal{S}}$ should be such that

$$\mathbb{P}\left(P_{\mathcal{C}}(X_{\mathcal{S}}, \boldsymbol{k}) \text{ verifies } \mathcal{P} \,|\, \boldsymbol{k} \sim \mathcal{U}\left(\mathbb{F}_2^m\right)\right) \text{ is far from } \mathbb{P}\left(f \text{ verifies } \mathcal{P} \,\middle|\, f \sim \mathcal{U}\left(\mathbb{F}_2^{|\mathcal{S}|} \to \mathbb{F}_2\right)\right).$$

For example in `Trivium`, the authors found that for a specific cube $\mathcal{C}$ with 30 variables and after 790 initialization rounds, the superpoly $P_{\mathcal{C}}(X_{[0,79]\setminus\mathcal{C}}, \boldsymbol{k})$ is constant.

It is important to understand for Section 5.3.2.4 that the chosen property tester could succeed for most of the keys $\boldsymbol{k}$ but still fail for a non-negligeable proportion of the keys.

5.3.2.4 Dynamic cube attacks

In 2011, Dinur and Shamir published in [DS11] a generic method for turning a cube-tester-based distinguisher into a key-recovery attack. They called this method a *dynamic cube attack*. The motivation is that cube testers, which only gave distinguishers at best, have the strength of working with a wide choice of properties to test on superpolys. On the contrary cube attacks, which are key-recovery attacks, mostly work with linearity. Dynamic cube attacks are then key-recovery attacks that benefit from this strength of cube testers. They however need a careful analysis of the cipher.

With this method, Dinur, Güneysu, Paar, Shamir and Zimmermann published in [Din+11] a key-recovery attack on the full stream cipher `Grain-128` [Hel+06] with 128-bit keys in approximatly 2^{90} queries which succeeds for 7.5% of the keys. In [Hao+20a], Hao, Jiao, Li, Meier, Todo and Wang proposed another dynamic cube attack on the same cipher in 2^{125} queries which succeeds for 99.83% of the keys. Their analysis of the cipher used methods based on the division property introduced in the next section.

Note that with dynamic cube attacks, cryptographers can show the existence of an attack with complexity beyond their limited computational power but better than exhaustive search.

5.4 Monomial trails

Differential and linear cryptanalysis heavily rely on *trails* and their probabilities. The goal of this section is to define the equivalent of differential and linear trails for ANF analysis, *monomial trails*, for which there will be no probabilities involved.

5.4.1 Motivation

A very important trend to mount cube attacks or distinguishers since 2017 is to use algorithms based on the *division property* to analyze ANFs of superpolys. The division property was first introduced by Todo in 2015 [Tod15] as a generalization of the integral property *balance* to propose new integral distinguishers which take advantage of the algebraic degrees of the components of a cipher. In a nutshell, the set $\mathcal{X} \in \mathbb{F}_2^n$ has the balance integral property if $\sum_{\boldsymbol{x}\in\mathcal{X}} \boldsymbol{x} = 0$. If there exists a set of plaintexts $\mathcal{X}$ such that for all keys $\boldsymbol{k}$ the set $\{ E(\boldsymbol{x}, \boldsymbol{k}) \mid \boldsymbol{x} \in \mathcal{X} \}$ has the balance property, we have an integral distinguisher on the cipher E. The generalisation of the balance property given by the division property basically allows to compute an integral property on a set of ciphertexts from an initial integral property on the set of chosen plaintexts by following the propagation of the division property through the rounds of the cipher. In 2016, Xiang, Zhang, Bao and Lin [Xia+16] showed how to model the propagation of the division property in MILP to automate the search for integral distinguishers. In 2017, Todo, Isobe, Hao and Meier [Tod+17] applied those techniques to cube attacks for the first time: their idea was to detect neutral private variables in the superpoly of a given cube by modeling the propagation of the division property in the cipher under study as a MILP problem. That allowed them to work with another family of superpolys — superpolys which depend on just a few private variables — and to propose security analysis for the stream ciphers `Trivium`, `Grain-128a` and `Acorn` beyond practical complexities. These work paved a way for the use of the division property and MILP models as tools for the analysis of ANFs.

Our personal grasp on the subject mostly comes from the work of Boura and Canteaut in [BC16] who presented the division property in a more ANF-oriented way and from the work of Hebborn, Lambin, Leander and Todo in [Heb+20b] who propose division-property- and MILP-based methods to compute lower bounds on the degree of block ciphers. It converged with the one of Hu, Sun, Wang and Wang in [Hu+20] and we will use a similar terminology as it is more consistent with this chapter even if it is not widespread. We will not present how the division property evolved since the original work of Todo in 2015 [Tod15] nor the different variants that have been used to mount cube attacks. What we present as *monomial trails* is equivalent to the most precise variants: the *parity sets* from [BC16] or the *three-subset division property without unknown subset* from [Hao+20b].

5.4.2 Basic definitions and properties

Let us start with an example on a Boolean function $F : \mathbb{F}_2^n \to \mathbb{F}_2$ of which we do not know the ANF but we know a decomposition $F = H \circ G$ with $G : \mathbb{F}_2^n \to \mathbb{F}_2^m$ and $H : \mathbb{F}_2^m \to \mathbb{F}_2$. We assume we know the ANFs of G and H.

$$G(X) = \sum_{\boldsymbol{u}\in\mathbb{F}_2^n} g_{\boldsymbol{u}} \cdot X^{\boldsymbol{u}} \qquad \text{and} \qquad H(Y) = \sum_{\boldsymbol{v}\in\mathbb{F}_2^m} h_{\boldsymbol{v}} \cdot Y^{\boldsymbol{v}}.$$

The question is: does a given monomial $X^{\boldsymbol{u}}$ appear in the ANF of F? The study of monomial trails will be a tool that allows to answer this question without using classical coefficient computations

with the Möbius transform in $2^{\mathrm{wt}(\boldsymbol{u})}$ operations, which could indeed be too heavy in cryptographic applications. Instead, we will write

$$F(X) = H(G(X)) = \sum_{\boldsymbol{v} \in \mathbb{F}_2^m} h_{\boldsymbol{v}} \cdot G^{\boldsymbol{v}}(X)$$

and compute the set $\mathcal{V}$ of vectors $\boldsymbol{v}$ such that the monomial $X^{\boldsymbol{u}}$ belongs to the ANF of $G^{\boldsymbol{v}}(X)$. In cryptographic applications, this kind of computation may be possible because the ciphers are built with small components. Finally, we know that $X^{\boldsymbol{u}}$ appears in the ANF of F if and only if the number of vectors $\boldsymbol{v}$ from $\mathcal{V}$ such that $h_{\boldsymbol{v}} = 1$ is odd. Indeed, the coefficient of $X^{\boldsymbol{u}}$ in F is equal to $\sum_{\boldsymbol{v} \in \mathcal{V}} h_{\boldsymbol{v}}$. In the following, we formalize this idea for more complex constructions and introduce the notation $X^{\boldsymbol{u}} \in G(X)^{\boldsymbol{v}}$ if $\boldsymbol{v} \in \mathcal{V}$, $X^{\boldsymbol{v}} \in H(X)$ if $h_v = 1$ and $\ell((G,H), \boldsymbol{u} \to 1) = |\{\, \boldsymbol{v} \in \mathbb{F}_2^m \mid X^{\boldsymbol{u}} \in G(X)^{\boldsymbol{v}} \text{ and } X^{\boldsymbol{v}} \in H(X) \,\}|$.

Notation 5.2. Let $F : \mathbb{F}_2^n \to \mathbb{F}_2^m$, $\boldsymbol{u} \in \mathbb{F}_2^n$ and $\boldsymbol{v} \in \mathbb{F}_2^m$. If the monomial $X^{\boldsymbol{u}}$ appears in the ANF of the Boolean function $F^{\boldsymbol{v}}(X) = \prod_{i=0}^{m-1} F_i(X)^{v_i}$ then we write $X^{\boldsymbol{u}} \in F(X)^{\boldsymbol{v}}$.

Example 5.4 (Propagation rules)**.** We illustrate Notation 5.2 with a few elementary functions. The relations between vectors $\boldsymbol{u}$ and $\boldsymbol{v}$ are called propagation rules.

- $\mathtt{xor}(X_0, \ldots, X_{n-1}) = X_0 + \cdots + X_{n-1}$.

$$X^{\boldsymbol{u}} \in \mathtt{xor}(X)^v \iff \begin{cases} v = 0 & \text{and} \quad \boldsymbol{u} = (0, 0, \ldots, 0) \text{ or} \\ v = 1 & \text{and} \quad \mathrm{wt}(\boldsymbol{u}) = 1. \end{cases}$$

- `xor` with a constant: $F(X) = X + 1$.

$$X^{\boldsymbol{u}} \in F(X)^v \iff u = 0 \text{ or } u = v = 1.$$

- $\mathtt{and}(X_0, \ldots, X_{n-1}) = X_0 X_1 \cdots X_{n-1}$.

$$X^{\boldsymbol{u}} \in \mathtt{and}(X)^v \iff \boldsymbol{u} = (v, v, \ldots, v).$$

- $\mathtt{copy}(X) = (X, X, \ldots, X)$. Since $\mathtt{copy}(X)^{\boldsymbol{v}} = X$ for all $\boldsymbol{v} \neq \mathbf{0}$,

$$X^u \in \mathtt{copy}(X)^{\boldsymbol{v}} \iff u = v_0 \vee v_1 \vee \cdots \vee v_{n-1}$$

- Concatenation with constant $F(X_0, \ldots, X_{n-1}) = (\, X \mid \boldsymbol{c} \,)$.

$$X^{\boldsymbol{u}} \in F(X)^{(\, \boldsymbol{v} \mid \boldsymbol{v}' \,)} \iff \boldsymbol{u} = \boldsymbol{v} \text{ and } \boldsymbol{v}' \preceq \boldsymbol{c}.$$

Definition 5.9 (Boolean circuit)**.** Let $r \in \mathbb{N}^*$ and $n^{(i)} \in \mathbb{N}$ for $i \in [0, r]$. A Boolean circuit F of depth r is an r-tuple of composable maps

$$\left(F^{(i)} : \mathbb{F}_2^{n^{(i-1)}} \to \mathbb{F}_2^{n^{(i)}} \;\middle|\; i \in [1, r] \right).$$

F can also denote the map $F^{(r)} \circ \cdots \circ F^{(1)}$.

Example 5.5. The architecture of a key-alternating block cipher naturally corresponds to a Boolean circuit. Let n be the block size, m the master-key size, $R^{(i)} : \mathbb{F}_2^n \to \mathbb{F}_2^n$, $i \in [1, r]$ the r round transformations and $Z : \mathbb{F}_2^m \to \mathbb{F}_2^{n(r+1)}$ the key expansion. The first map of the circuit is

$$F^{(1)} : \left\{ \begin{array}{ccc} \mathbb{F}_2^{m+n} & \longrightarrow & \mathbb{F}_2^{n(r+2)} \\ (\, \boldsymbol{k} \mid \boldsymbol{x} \,) & \longmapsto & (\, Z(\boldsymbol{k}) \mid \boldsymbol{x} \,). \end{array} \right.$$

Then

$$F^{(2)} : \left\{ \begin{array}{ccc} \mathbb{F}_2^{n(r+2)} & \longrightarrow & \mathbb{F}_2^{n(r+1)} \\ \left(\boldsymbol{k}^{(0)} \,\middle|\, \cdots \,\middle|\, \boldsymbol{k}^{(r)} \,\middle|\, \boldsymbol{x} \right) & \longmapsto & \left(\boldsymbol{k}^{(1)} \,\middle|\, \cdots \,\middle|\, \boldsymbol{k}^{(r)} \,\middle|\, \boldsymbol{k}^{(0)} + \boldsymbol{x} \right), \end{array} \right.$$

and for all $i \in [1, r]$,

$$F^{(i+2)} : \left\{ \begin{array}{ccc} \mathbb{F}_2^{n(r-i+2)} & \longrightarrow & \mathbb{F}_2^{n(r-i+1)} \\ \left(\boldsymbol{k}^{(i)} \,\middle|\, \cdots \,\middle|\, \boldsymbol{k}^{(r)} \,\middle|\, \boldsymbol{x} \right) & \longmapsto & \left(\boldsymbol{k}^{(i+1)} \,\middle|\, \cdots \,\middle|\, \boldsymbol{k}^{(r)} \,\middle|\, R^{(i)}(\boldsymbol{x}) + \boldsymbol{k}^{(i)} \right). \end{array} \right.$$

Definition 5.10 (Monomial trail)**.** Let $r \in \mathbb{N}^*$, F be a Boolean circuit of depth r and $\boldsymbol{u}^{(i)} \in \mathbb{F}_2^{n^{(i)}}$ for $i \in [0, r]$. If $X^{\boldsymbol{u}^{(i-1)}} \in F^{(i)}(X)^{\boldsymbol{u}^{(i)}}$ for all i then we call the tuple $\left(\boldsymbol{u}^{(0)}, \ldots, \boldsymbol{u}^{(r)} \right)$ a monomial trail for the circuit F.

Moreover, if $\boldsymbol{u} \in \mathbb{F}_2^{n^{(0)}}$ and $\boldsymbol{v} \in \mathbb{F}_2^{n^{(r)}}$, $\ell(F, \boldsymbol{u} \to \boldsymbol{v})$ will denote the number of monomial trails between $\boldsymbol{u}$ and $\boldsymbol{v}$:

$$\ell(F, \boldsymbol{u} \to \boldsymbol{v}) \stackrel{\text{def}}{=} \left| \left\{ \left(\boldsymbol{u}^{(1)}, \cdots, \boldsymbol{u}^{(r-1)} \right) \;\middle|\; \left(\boldsymbol{u}, \boldsymbol{u}^{(1)}, \ldots, \boldsymbol{u}^{(r-1)}, \boldsymbol{v} \right) \text{ is a monomial trail} \right\} \right|.$$

Finally, we define the convention that if F is a circuit of depth 1, then for all $\boldsymbol{u}, \boldsymbol{v}$,

$$\ell(F, \boldsymbol{u} \to \boldsymbol{v}) \in \{0, 1\} \qquad \text{and} \qquad \ell(F, \boldsymbol{u} \to \boldsymbol{v}) = 1 \iff X^{\boldsymbol{u}} \in F(X)^{\boldsymbol{v}}. \tag{5.1}$$

It is then easy to check that if F is a circuit of depth $r > 1$, we have the following recurrence relation for all $\boldsymbol{u}, \boldsymbol{v}$,

$$\ell(F, \boldsymbol{u} \to \boldsymbol{v}) = \sum_{\boldsymbol{w}:\ X^{\boldsymbol{w}} \in F^{(r)}(X)^{\boldsymbol{v}}} \ell\left(\left(F^{(1)}, \ldots, F^{(r-1)} \right), \boldsymbol{u} \to \boldsymbol{w} \right). \tag{5.2}$$

Proposition 5.2. *Let F be a Boolean circuit of any depth. It holds for all $\boldsymbol{u}$, $\boldsymbol{v}$ that*

$$X^{\boldsymbol{u}} \in F(X)^{\boldsymbol{v}} \iff \ell(F, \boldsymbol{u} \to \boldsymbol{v}) \equiv 1 \bmod 2.$$

Proof. The proof will work by induction on r where r is the depth of F. The initialization case of the induction is given by Equation (5.1). Let us assume that Proposition 5.2 holds for all circuits of depth $r - 1$. We have by definition of F that

$$F^{\boldsymbol{v}}(X) = \sum_{\boldsymbol{w}:\ X^{\boldsymbol{w}} \in F^{(r)}(X)^{\boldsymbol{v}}} \left(F^{(r-1)} \circ \cdots \circ F^{(1)} \right)^{\boldsymbol{w}}(X). \tag{5.3}$$

Let $\boldsymbol{u} \in \mathbb{F}_2^{n^{(0)}}$, $\boldsymbol{v} \in \mathbb{F}_2^{n^{(r)}}$ and

$$\mathcal{S} \stackrel{\text{def}}{=} \left\{ \boldsymbol{w} \in \mathbb{F}_2^{n^{(r-1)}} \;\middle|\; \ell\left(\left(F^{(1)}, \ldots, F^{(r-1)} \right), \boldsymbol{u} \to \boldsymbol{w} \right) \text{ is odd and } X^{\boldsymbol{w}} \in F^{(r)}(X)^{\boldsymbol{v}} \right\}.$$

We know from the induction hypothesis that for all $\boldsymbol{w}$ such that $X^{\boldsymbol{w}} \in F^{(r)}(X)^{\boldsymbol{v}}$,

$$X^{\boldsymbol{u}} \in \left(F^{(r-1)} \circ \cdots \circ F^{(1)}\right)(X)^{\boldsymbol{w}} \iff \boldsymbol{w} \in \mathcal{S}.$$

We also deduce from Equation (5.2) that $|\mathcal{S}| \equiv \ell(F, \boldsymbol{u} \to \boldsymbol{v}) \mod 2$. With Equation (5.3), this implies that $X^{\boldsymbol{u}}$ appears in the ANF of $F^{\boldsymbol{v}}(X)$ if and only if $\ell(F, \boldsymbol{u} \to \boldsymbol{v})$ is odd. □

Corollary 5.1. *Let F be a Boolean circuit, $\boldsymbol{u} \in \mathbb{F}_2^{n^{(0)}}$ and $\boldsymbol{v} \in \mathbb{F}_2^{n^{(r)}}$. If $\ell(F, \boldsymbol{u} \to \boldsymbol{v}) = 0$ then $X^{\boldsymbol{u}} \notin F^{\boldsymbol{v}}(X)$.*

5.4.3 Practical circuits for monomial trails

Proposition 5.2 and Corollary 5.1 will be useful if we can indeed compute $\ell(F, \boldsymbol{u} \to \boldsymbol{v})$ or prove that it is zero. However, this computation needs the knowledge of the ANFs of all the maps of the circuit $F^{(i)}$ and all the ANFs $F^{(i)\boldsymbol{v}}$ for all $\boldsymbol{v} \in \mathbb{F}_2^{n^{(i)}}$. Depending on the maps $F^{(i)}$ and on the dimensions $n^{(i)}$, these ANFs might be too large to allow any computation in practice. In this section, we define a family of Boolean circuits, parallel-map circuits, based on parallel applications of maps with small ANFs, for which such a computation may be practical.

Lemma 5.1. *Let $n, m, d \in \mathbb{N}^*$, $n_j, m_j \in \mathbb{N}^*$ such that $\sum_{j=0}^{d-1} n_j = n$ and $\sum_{j=0}^{d-1} m_j = m$. Let $F_j : \mathbb{F}_2^{n_j} \to \mathbb{F}_2^{m_j}$ for $j \in [0, d-1]$. When a map F applies the maps F_j in parallel, i.e.*

$$F(\boldsymbol{x}_0 \mid \cdots \mid \boldsymbol{x}_{d-1}) = (F_0(\boldsymbol{x}_0) \mid \cdots \mid F_{d-1}(\boldsymbol{x}_{d-1})),$$

then for all $\boldsymbol{u} = (\boldsymbol{u}_0 \mid \cdots \mid \boldsymbol{u}_{d-1}) \in \mathbb{F}_2^n$ and $\boldsymbol{v} = (\boldsymbol{v}_0 \mid \cdots \mid \boldsymbol{v}_{d-1}) \in \mathbb{F}_2^m$,

$$X^{\boldsymbol{u}} \in F(X)^{\boldsymbol{v}} \iff \forall j \in [0, d-1],\ X^{\boldsymbol{u}_j} \in F_j(X)^{\boldsymbol{v}_j}.$$

Lemma 5.1 has a straightforward proof and it shows that when a map F applies in parallel smaller maps with small ANFs (and whose coordinate products have small ANFs), we can deduce the relations $X^{\boldsymbol{u}} \in F(X)^{\boldsymbol{v}}$ for *all* $(\boldsymbol{u}, \boldsymbol{v}) \in \mathbb{F}_2^n \times \mathbb{F}_2^m$ from the small ANFs, even if n and m are too big for enumerating elements of $\mathbb{F}_2^n \times \mathbb{F}_2^m$.

Definition 5.11 (parallel-map circuit)**.** A Boolean circuit F of depth $2r-1$ is a parallel-map circuit of depth r if it is a tuple of maps of the form

$$\left(F^{(1)}, \pi^{(1)}, F^{(2)}, \pi^{(2)}, \ldots, \pi^{(r-1)}, F^{(r)}\right)$$

where:

- each $F^{(i)} : \mathbb{F}_2^{n^{(i-1)}} \to \mathbb{F}_2^{n^{(i)}}$ applies the maps $F_j^{(i)} : \mathbb{F}_2^{n_j^{(i-1)}} \to \mathbb{F}_2^{m_j^{(i)}}$, with $\sum_j m_j^{(i)} = n^{(i)}$ in parallel;
- each $\pi^{(i)} : \mathbb{F}_2^{n^{(i)}} \to \mathbb{F}_2^{n^{(i)}}$ is a bit permutation.

Remark 5.4. For a bit permutation $\pi : \mathbb{F}_2^n \to \mathbb{F}_2^n$, we have that

$$\forall \boldsymbol{u}, \boldsymbol{v} \in \mathbb{F}_2^n, \quad X^{\boldsymbol{u}} \in \pi(X)^{\boldsymbol{v}} \iff \boldsymbol{v} = \pi(\boldsymbol{u}).$$

Any Boolean circuit can be considered as a parallel-map circuit with $d^{(i)} = 1$ for all i but defining parallel-map circuits is useful when the dimensions $n_j^{(i)}$ are small enough (typically up to 16) for practical exhaustive enumerations of $(\boldsymbol{u}, \boldsymbol{v}) \in \mathbb{F}_2^{n_j^{(i-1)}} \times \mathbb{F}_2^{n_j^{(i)}}$ such that $X^{\boldsymbol{u}} \in F_j^{(i)}(X)^{\boldsymbol{v}}$. Indeed, with Lemma 5.1, these small enumerations give the relations $X^{\boldsymbol{u}} \in F^{(i)}(X)^{\boldsymbol{v}}$ for all $(\boldsymbol{u}, \boldsymbol{v}) \in \mathbb{F}_2^{n^{(i-1)}} \times \mathbb{F}_2^{n^{(i)}}$. Therefore, computing the value $\ell(F, \boldsymbol{u} \to \boldsymbol{v})$ for all $(\boldsymbol{u}, \boldsymbol{v}) \in \mathbb{F}_2^{n^{(0)}} \times \mathbb{F}_2^{n^{(r)}}$ may become possible with Equation (5.2) for example.

Example 5.6 (`Grain-v1`)**.** Following Section 1.3.4, we have the following parallel-map circuit for the initialization of `Grain-v1`. The first map is a concatenation with a constant (see Example 5.4) and the first bit permutation switches places between the two "registers" of the LFSR and the NFSR to be consistent with the description given in Section 1.3.4.

$$F^{(1)} : \left\{ \begin{array}{ccc} \mathbb{F}_2^{80+64} & \longrightarrow & \mathbb{F}_2^{80+80} \\ (\,\boldsymbol{k} \mid \boldsymbol{x}\,) & \longmapsto & (\,\boldsymbol{k} \mid \boldsymbol{x} \mid \mathbf{1}_{16}\,), \end{array} \right. \qquad \pi^{(1)}(\,\boldsymbol{b} \mid \boldsymbol{s}\,) = (\,\boldsymbol{s} \mid \boldsymbol{b}\,).$$

Then, before computing the first bit of the output sequence, some bits have to be copied and placed to the right position.

$$F^{(2)} : \left\{ \begin{array}{ccc} \mathbb{F}_2^{80+80} & \longrightarrow & \mathbb{F}_2^{80+80+12} \\ (\,\boldsymbol{s} \mid \boldsymbol{b}\,) & \longmapsto & (\,\boldsymbol{s}_{0,\ldots,2} \mid \mathtt{copy}(s_3) \mid \boldsymbol{s}_{4,\ldots,24} \mid \mathtt{copy}(s_{25}) \mid \cdots), \end{array} \right.$$
$$\pi^{(2)}(\,\boldsymbol{s}_{0,\ldots,2} \mid s_3, s_3' \mid \cdots) = \left(\,\boldsymbol{s} \;\middle|\; \boldsymbol{b} \;\middle|\; \boldsymbol{b}'_{\mathcal{J}\cup 63} \;\middle|\; \boldsymbol{s}'_{\{3,25,46,64\}}\,\right).$$

The following function computes the first bit of the output sequence with the ANF given by Equation (1.1).

$$F^{(3)} : \left\{ \begin{array}{ccc} \mathbb{F}_2^{80+80+12} & \longrightarrow & \mathbb{F}_2^{80+80+1} \\ \left(\,\boldsymbol{s} \;\middle|\; \boldsymbol{b} \;\middle|\; \boldsymbol{b}'_{\mathcal{J}\cup\{63\}} \;\middle|\; \boldsymbol{s}'_{\{3,25,46,64\}}\,\right) & \longmapsto & (\,\boldsymbol{s} \mid \boldsymbol{b} \mid z_0\,). \end{array} \right.$$

Clocking the FSRs is given by

$$(\,\boldsymbol{s} \mid \boldsymbol{b} \mid z_0\,) \xmapsto{\pi^{(4)} \circ F^{(4)} \circ \pi^{(3)}} \left(\,\boldsymbol{s}_{1,\ldots,79} \;\middle|\; \left(\,z_0' \;\middle|\; \boldsymbol{s}'_{\{0,13,23,38,51,62\}}\,\right) \;\middle|\; \boldsymbol{b}_{1,\ldots,79} \;\middle|\; (\,s_0, z_0 \mid \boldsymbol{b}')\,\right)$$
$$\xmapsto{F^{(5)}} (\,\boldsymbol{s}_{1,\ldots,79} \mid z_0' + F(\boldsymbol{s}') \mid \boldsymbol{b}_{1,\ldots,79} \mid s_0 + z_0 + G(\boldsymbol{b}')\,),$$

where $F^{(4)}$ is composed of `copy` operations. The rest of the circuit repeats the sequence from $F^{(2)}$.

At each step i of the parallel-map circuit given in Example 5.6, Lemma 5.1 allows to deduce whether $X^{\boldsymbol{u}} \in F^{(i)}(X)^{\boldsymbol{v}}$ for all $\boldsymbol{u} \in \mathbb{F}_2^{n^{(i-1)}}$ and $\boldsymbol{v} \in \mathbb{F}_2^{n^{(i)}}$ with:

- the propagation rules for `copy` and the concatenation (given in Example 5.4),
- the ANFs of the Boolean functions F, G and with the one given by Equation (1.1).

Interestingly, all involved ANFs are small enough to make this deduction with practical computational resources even if we always have at least 2^{80+80} pairs $(\boldsymbol{u}, \boldsymbol{v})$ at each step i.

Example 5.7 (PRESENT)**.** The specification of PRESENT gives natural parallel-map circuits. For example, following Section 1.4.2 and Figure 1.15, the first round-key addition is given by

$$F^{(1)} : \begin{cases} \mathbb{F}_2^{80+64} & \longrightarrow & \mathbb{F}_2^{80+64+64} \\ (\,\boldsymbol{k} \mid \boldsymbol{x}\,) & \longmapsto & (\,\mathtt{copy}_2(k_0), \ldots, \mathtt{copy}_2(k_{63}) \mid k_{64}, \ldots, k_{79} \mid \boldsymbol{x}\,), \end{cases}$$

$$\pi^{(1)}\left(\,k_0, k_0', k_1, k_1', \ldots, k_{63}, k_{63}', k_{64}, \ldots, k_{79} \mid \boldsymbol{x}\,\right) = \left(\,k_0, \ldots, k_{79} \mid k_0', \ldots, k_{63}' \mid \boldsymbol{x}\,\right),$$

$$F^{(2)} : \begin{cases} \mathbb{F}_2^{80+64+64} & \longrightarrow & \mathbb{F}_2^{80+64} \\ \left(\boldsymbol{k} \,\middle|\, \boldsymbol{k}^{(0)} \,\middle|\, \boldsymbol{x}\right) & \longmapsto & \left(\boldsymbol{k} \,\middle|\, \boldsymbol{k}^{(0)} + \boldsymbol{x}\right). \end{cases}$$

Then the first step of the update function gives the permutation

$$\pi^{(2)}\left(\,\boldsymbol{k} \mid \boldsymbol{x}\,\right) = \left(\,\left(\,k_{i+19} \mid i \in [0, 79]\,\right) \mid \boldsymbol{x}\,\right).$$

Now comes the two other steps of the update function and the Sbox layer:

$$F^{(3)}\left(\,\boldsymbol{k} \mid \boldsymbol{x}\,\right) = (\,(\,\boldsymbol{k}_{0,\ldots,14} \mid \boldsymbol{k}_{15,\ldots,19} \oplus 1 \mid \boldsymbol{k}_{20,\ldots,75} \mid \mathtt{Sbox}\left(\boldsymbol{k}_{76,\ldots,79}\right)\,) \mid \left(\,\mathtt{Sbox}\left(\boldsymbol{x}_{0,\ldots,3}\right) \mid \cdots \mid \mathtt{Sbox}\left(\boldsymbol{x}_{60,\ldots,63}\right)\,)\right).$$

Finally, the linear layer of PRESENT is a bit permutation P:

$$\pi^{(3)}\left(\,\boldsymbol{k} \mid \boldsymbol{x}\,\right) = \left(\,\boldsymbol{k} \mid P(\boldsymbol{x})\,\right).$$

The rest of the circuit repeats the above sequence with a small change for the round-constant in $F^{(3)}$.

In Example 5.6, the most complex operations $F_j^{(i)}$ (with the notation of Definition 5.11) are Boolean functions with exactly *one* output bit. Applying Lemma 5.1 in that case only requires the ANFs of the functions $F_j^{(i)}$. Things are a bit more complex for the parallel-map circuit of PRESENT given in Example 5.7 because of the Sboxes. Indeed, Lemma 5.1 says that we need to know whether $X^{\boldsymbol{u}} \in \mathtt{Sbox}\,(X)^{\boldsymbol{v}}$ for all $(\boldsymbol{u}, \boldsymbol{v}) \in \mathbb{F}_2^4 \times \mathbb{F}_2^4$. Equivalently, we need to compute the ANF of $\mathtt{Sbox}(X)^{\boldsymbol{v}}$ for all $\boldsymbol{v} \in \mathbb{F}_2^4$. As explained in [BC16], the map

$$(\boldsymbol{u}, \boldsymbol{v}) \mapsto \begin{cases} 1 & \text{if } X^{\boldsymbol{u}} \in F\,(X)^{\boldsymbol{v}}\,, \\ 0 & \text{otherwise,} \end{cases}$$

is the counterpart of the DDT for studying monomial trails in circuits with Sboxes. We call it the *ANF table*.

5.4.4 MILP models for monomial trails

Let F be a parallel-map circuit of depth $2r - 1$. Instead of using Equation (5.2) to compute the number of monomial trails for a given input/output pair $(\boldsymbol{u}, \boldsymbol{v})$, we can see propagation rules and ANF tables as constraints that characterize the set

$$\left\{ \left(\boldsymbol{u}^{(0)}, \boldsymbol{u}^{(1)}, \cdots, \boldsymbol{u}^{(2r-1)}\right) \;\middle|\; \left(\boldsymbol{u}^{(0)}, \boldsymbol{u}^{(1)}, \cdots, \boldsymbol{u}^{(2r-1)}\right) \text{ is a monomial trail for } F \right\}. \tag{5.4}$$

If we have a tool which can enumerate the elements of this set — these elements are monomial trails — we can use it to compute $\ell\,(F, \boldsymbol{u} \to \boldsymbol{v})$ for a chosen input/output pair of exponents $(\boldsymbol{u}, \boldsymbol{v})$ by adding the constraints $\boldsymbol{u}^{(0)} = \boldsymbol{u}$ and $\boldsymbol{u}^{(2r-1)} = \boldsymbol{v}$.

MILP-based enumeration tool. In practice, the propagation rules are often modeled by $\mathbb{R}$-linear expressions which will be used as MILP constraints (see Chapter 4). This MILP model together with a MILP solver will then be able to perform that enumeration. MILP models for propagation rules are given in Appendix F of [Heb+20a] and we give below the models corresponding to the rules of Example 5.4. The MILP variable corresponding to an exponent coordinate $u_i \in \mathbb{F}_2$ is $x_{u_i} \in \{0,1\} \subset \mathbb{Z}$. This distinction is important because the addition is not the same in both cases.

$$\begin{aligned}
X^{\boldsymbol{u}} \in \texttt{xor}\,(X)^v &\iff x_{u_0} + \cdots + x_{u_{n-1}} = x_v, \\
X^u \in \texttt{xor}(X,1)^v &\iff x_u \leqslant x_v, \\
X^{\boldsymbol{u}} \in \texttt{and}\,(X)^v &\iff \forall i,\ x_{u_i} = x_v, \\
X^u \in \texttt{copy}\,(X)^{\boldsymbol{v}} &\iff \begin{cases} x_u \leqslant x_{v_0} + \cdots + x_{v_{n-1}} \\ \forall i,\ x_u \geqslant x_{v_i}, \end{cases} \\
X^{\boldsymbol{u}} \in (\ X \mid \boldsymbol{c}\ )^{(\ \boldsymbol{v} \mid \boldsymbol{v'}\ )} &\iff \begin{cases} \forall i,\ c_i = 0 \Rightarrow x_{v'_i} = 0 \\ \forall i,\ x_{u_i} = x_{v_i}. \end{cases}
\end{aligned}$$

For more complex operations like Sboxes, ANF tables can be modeled thanks to the techniques presented in Section 4.2.

Using the enumeration tool. Let F be a parallel-map circuit of depth $2r-1$. With the enumeration tool and the constraints $\boldsymbol{u}^{(0)} = \boldsymbol{u}$ and $\boldsymbol{u}^{(2r-1)} = \boldsymbol{e}_i \in \mathbb{F}_2$, we can answer the question "Does $X^{\boldsymbol{u}}$ belong to the ANF of $F_i(X)$?" by "yes" or "no". Indeed, we know thanks to Proposition 5.2 that the answer is yes if and only if this constrained enumeration returns an odd number.

Here is another useful example. With a program which outputs whether the set in Equation (5.4) has any element and the constraints $\boldsymbol{u}^{(0)} \succeq \boldsymbol{u}$, $\boldsymbol{u}^{(2r-1)} = \boldsymbol{e}_i$, we can answer the question "Does a given monomial $X^{\boldsymbol{u}}$ have multiples, i.e. $X^{\boldsymbol{v}}$ with $\boldsymbol{u} \preceq \boldsymbol{v}$, in the ANF of $F_i(X)$?" by "no" or "don't know". Indeed, Corollary 5.1 ensures that if there is no trail, the answer is no but if there exists at least one trail the answer can't be better than "don't know".

5.4.5 Superpoly recovery with monomial trails

Let F denote a Boolean function with public variables X and private variables K for which a parallel-map circuit exists. In the first application of the division property to cube attacks, Todo, Isobe, Hao and Meier [Tod+17] model the propagation of the bit-based division property (introduced in [TM16]) as a MILP problem to answer the question "Does a given monomial $X^{\boldsymbol{u}}$ have multiples in the ANF of $F(X)$"? by "no" or "don't know" efficiently. With such a tool, they can query whether the monomial $X^{\boldsymbol{e}_{\mathcal{I}}} K_\ell$ has multiples in the function

$$F(\sigma_{\mathcal{C}}(X_{\mathcal{I}}), K_0, \ldots, K_{m-1})$$

for a chosen cube $\mathcal{C} = (\mathcal{I}, \mathcal{J})$, a chosen private variable K_ℓ and where $\sigma_{\mathcal{C}}$ is defined in Definition 5.5. If the answer is "no", they have a proof that the variable K_ℓ is not involved in the superpoly $P_{\mathcal{C}}(K)$. This way they can compute a set $\mathcal{K}$ of variables possibly involved in the superpoly $P_{\mathcal{C}}(K)$. Computing the value vector of this superpoly then requires $2^{|\mathcal{I}|+|\mathcal{K}|}$ queries to the function F. Once the value vector is known, the ANF of $P_{\mathcal{C}}(K_{\mathcal{K}})$ can be efficiently computed with the fast Möbius transform.

This was the preprocessing phase of the cube attack during which several superpolies can be computed. Let N be the number of computed superpolys. For the online phase, they classically compute the secret-key-dependent constants $P_{\mathcal{C}}(\boldsymbol{k})$ for each superpoly in $2^{|\mathcal{I}|}$ queries. Assuming the superpolies are balanced and that their sets $\{\, \boldsymbol{k} \mid P_{\mathcal{C}}(\boldsymbol{k}) = 0 \,\}$ are independent when $\boldsymbol{k}$ is chosen at random, the polynomial equations are satisfied by 2^{m-N} keys. An exhaustive search among those 2^{m-N} keys finishes the attack. The total complexity is $N \cdot (2^{|\mathcal{I}|+|\mathcal{K}|} + 2^{|\mathcal{I}|}) + 2^{m-N}$, where N is the number of superpolys considered in the attack. In practice, $2^{|\mathcal{I}|+|\mathcal{K}|}$ queries is too expensive to compute the superpoly in the preprocessing phase, but this gives a bound on the complexity. Nonetheless, since the superpoly is not really computed, the balance assumption may not hold. In particular, there may be a degenerate case where the superpoly is constant. To better estimate the complexity, the authors then estimate the probability p that $P_{\mathcal{C}}(K_{\mathcal{K}})$ is balanced by studying reduced versions of the cipher for which $2^{|\mathcal{I}|+|\mathcal{K}|}$ becomes practical. Let us sum up their attack.

Cryptographer's study.

- Estimate the probability p that superpolys in the cipher are balanced by studying the superpolys on reduced-round versions of the cipher.
- Compute $N \cdot p^{-1}$ cubes and sets $\mathcal{C}, \mathcal{K}$ such that the superpoly $P_{\mathcal{C}}(K)$ only depends on variables $K_{\mathcal{K}}$ with the bit-based division property.

Preprocessing.

- For $N \cdot p^{-1}$ pairs $(\mathcal{C}, \mathcal{K})$, compute superpolys $P_{\mathcal{C}}(K)$ in $2^{|\mathcal{I}|+|\mathcal{K}|}$ queries for each superpoly and keep N balanced superpolys.

Online phase.

- Compute the N constants $P_{\mathcal{C}}(\boldsymbol{k})$ in $2^{|\mathcal{I}|}$ queries for each constant.
- Exhaustively search for the secret key $\boldsymbol{k}$ among the 2^{m-N} solutions of the polynomial system

$$P_{\mathcal{C}}(K) = P_{\mathcal{C}}(\boldsymbol{k}).$$

Assuming the N cubes and sets have the same sizes, the total complexity is then bounded by

$$N \cdot \left(\frac{1}{p} \cdot 2^{|\mathcal{I}|+|\mathcal{K}|} + 2^{|\mathcal{I}|} \right) + 2^{m-N}.$$

The authors notably applied their attack to `Trivium` and `Acorn` for which they got the parameters of Table 5.2 in their best attacks. For those attacks with parameters $N = 1$ and $p = 1$, the complexity is dominated by the final exhaustive search in 2^{m-1} queries. This family of cube attacks first proposed in [Tod+17] seems to be perfectly suited to provide a theoretical security analysis for stream ciphers and their resistance to cube attacks. Indeed, for a given number of rounds of initialization, only one cube is needed to claim a theoretical attack that calls for more rounds of initialization.

A lot of work and effort has been made since the publication of [Tod+17] to push the attack further: Wang, Todo, Li, Isobe and Meier in [Wan+18] claimed a key-recovery attack on 839-round `Trivium`, later proved a mere distinguishing attack by Ye and Tian in [YT19] by showing that

Table 5.2 – Parameters of two attacks from [Tod+17] with $p = 1$ and $N = 1$.

cipher	attacked # rounds	key size	$\lvert\mathcal{I}\rvert$	$\lvert\mathcal{K}\rvert$
`Trivium`	832/1152	80	72	5
`Acorn`	704/1792	128	64	58

the superpoly used in the claimed key-recovery attack was constant. This is an example of a degenerate case where the estimation of the probability p is not enough to claim a key recovery. In the same work, Ye and Tian managed to compute the superpoly of the attack on 832-round `Trivium` for which we gave parameters in Table 5.2. Indeed, this superpoly is much simpler than what the division-property-based analysis of [Tod+17] could tell. Wang, Hu, Guan, Zhang and Shi independently arrived to the same conclusions in [Wan+19]. They also showed that using the three-subset division property introduced in [TM16] could clear the cryptographer from the balance assumption. Finally, Hao, Leander, Meier, Todo and Wang went a step further with the introduction of the *three-subset division property without unknown subset* in [Hao+20b]. This division property variant is equivalent to monomial trails. It allowed them to have an enumeration tool for monomial trails and to consequently answer the question "Does a given monomial $X^{\boldsymbol{u}}$ belong to the ANF of $F(X)$?" by "yes" or "no" with a MILP solver. Even better, using the enumeration tool, they can ask for the enumeration of all the monomial trails for a given output exponent (usually $\boldsymbol{e}_i$ for some output coordinate i). More precisely for a Boolean function $F(X)$, this enumeration provides at the same time

1. all the possible input exponents $\boldsymbol{u}$ such that $\ell\left(F, \boldsymbol{u} \to 1\right) \geqslant 1$,

2. for each possible input $\boldsymbol{u}$, all the monomial trails starting with $\boldsymbol{u}$ and hence the exact value for $\ell\left(F, \boldsymbol{u} \to 1\right)$.

In other words, they can compute any superpoly such that all its monomial trails can be enumerated by the MILP solver. They are not restricted to superpolys that only depend on a few variables anymore but rather on superpolys that have not too many monomials. They could completely recover balanced superpolys for 840-round and 841-round `Trivium` with 78 cube variables (over 80 public variables). This selection of works using the propagation of the division property for cube attacks since [Tod+17] is not exhaustive. Moreover, it should be noted that the surrounding algorithms and strategies that use division-property propagations are a major part of these works although we chose not to develop about them.

5.5 A new technique to compute low-density ANFs

In this section, we first explore a new technique to compute the ANF of a Boolean circuit. This technique is based on the evaluation of Boolean circuits over $\mathbb{Z}$ rather than $\mathbb{F}_2$ and on an adaptation of the Möbius transform to integer coefficients. We then explain how this technique could be used for computing superpolys and mounting cube attacks. Finally we compare it with monomial trails and give first hints on how to blend the two techniques.

5.5.1 Computing ANFs of low density with $\mathbb{Z}$-compatible circuits

We define in this section a family of parallel-map circuits, $\mathbb{Z}$-compatible circuits, and we propose an algorithm to compute the ANF of a $\mathbb{Z}$-compatible circuit when it is sparse enough. This algorithm is not based on monomial trails but on the evaluation of ANFs over the ring of integers $\mathbb{Z}$.

5.5.1.1 $\mathbb{Z}$-compatible circuits

Definition 5.12 ($\mathbb{Z}$-compatible circuits)**.** A Boolean circuit F of depth $3r$ is a $\mathbb{Z}$-compatible circuit of depth r if it is a tuple of maps of the form

$$\left(C^{(1)}, \pi^{(1)}, F^{(1)}, C^{(2)}, \pi^{(2)}, F^{(2)}, \ldots, C^{(1)}, \pi^{(1)}, F^{(1)}\right)$$

where:

- each $C^{(i)} : \mathbb{F}_2^{n^{(i-1)}} \to \mathbb{F}_2^{m^{(i)}}$ only applies `copy` operations (in parallel);
- each $\pi^{(i)} : \mathbb{F}_2^{m^{(i)}} \to \mathbb{F}_2^{m^{(i)}}$ is a bit permutation;
- each $F^{(i)} : \mathbb{F}_2^{m^{(i)}} \to \mathbb{F}_2^{n^{(i)}}$ applies the maps $F_j^{(i)} : \mathbb{F}_2^{m_j^{(i)}} \to \mathbb{F}_2$ in parallel.

Comparison with parallel-map circuits. A $\mathbb{Z}$-compatible circuit is essentially a parallel-map circuit for which all maps with small ANFs $F_j^{(i)}$ have exactly one output bit except the `copy` operations. Conversely, a parallel-map circuit G can be transformed into a $\mathbb{Z}$-compatible circuit F by isolating `copy` operations and creating `copy` operations before the maps $G_j^{(i)}$ with several output bits. More precisely, let $G^{(i)} : \mathbb{F}_2^n \to \mathbb{F}_2^m$ such that $m \geqslant 2$. We assume that $G^{(i)}$ has no parallel components to ease the notation without loss of generality. Let $\mathcal{V}$ be the set of $\boldsymbol{v} \in \mathbb{F}_2^m$ such that the product $X^{\boldsymbol{v}}$ appears in the ANFs of the next step $G^{(i+1)}$. Then with the copy operations

$$C : \left\{ \begin{array}{ccc} \mathbb{F}_2^n & \longrightarrow & \mathbb{F}_2^{|\mathcal{V}| \cdot n} \\ \boldsymbol{x} & \longmapsto & (\, \boldsymbol{x} \mid \cdots \mid \boldsymbol{x} \,) \end{array} \right.$$

applied just before, we can replace $G^{(i)}$ by

$$F^{(i)} : \left\{ \begin{array}{ccc} \mathbb{F}_2^{|\mathcal{V}| \cdot n} & \longrightarrow & \mathbb{F}_2^{|\mathcal{V}|} \\ (\, \boldsymbol{x}_{\boldsymbol{e}_0} \mid \cdots \mid \boldsymbol{x}_{\boldsymbol{v}} \mid \cdots) & \longmapsto & \left(G_0^{(i)}(\boldsymbol{x}) \;\middle|\; \cdots \;\middle|\; G^{(i)}(\boldsymbol{x})^{\boldsymbol{v}} \;\middle|\; \cdots \right) \end{array} \right.$$

and the monomials $X^{\boldsymbol{v}}$, $\boldsymbol{v} \in \mathcal{V}$ in the ANFs of $G^{(i+1)}$ by the corresponding output bit of $F^{(i)}$. The resulting $\mathbb{Z}$-compatible circuit F evaluates the same map as the original parallel-map circuit G but F cannot be better than G for computing monomial trails and for using Proposition 5.2.

Evaluation over $\mathbb{Z}$. Let F be a $\mathbb{Z}$-compatible circuit of depth r. Evaluating F means applying the ANFs of the successive operations $F_j^{(i)}$ to an input vector $\boldsymbol{x} \in \mathbb{F}_2^{n^{(0)}}$. Since the ANFs are only composed of additions and multiplications, we can evaluate the circuit F for an input vector $\boldsymbol{x}^{\mathbb{Z}} \in \mathbb{Z}^{n^{(0)}}$ by replacing the operations $(+, \times)$ of $\mathbb{F}_2$ by the operations $(+, \times)$ of the ring $\mathbb{Z}$. We then get a map $F^{\mathbb{Z}} : \mathbb{Z}^{n^{(0)}} \to \mathbb{Z}^{n^{(r)}}$.

We highlight that the output $F^{\mathbb{Z}}(\boldsymbol{x})$ depends on the circuit F in the sense that two circuits evaluating the same Boolean map over $\mathbb{F}_2$ can give different output values when evaluated over $\mathbb{Z}$. For example, the circuit $F = \left(F^{(1)}, F^{(2)}\right)$, with $F^{(1)}(X_1, X_2) = (X_1 + X_2, 1 + X_1)$ and $F^{(2)}(Y_1, Y_2) = Y_1Y_2$, evaluates the same map as $G = \left(G^{(1)}\right)$, with $G^{(1)}(X_1, X_2) = X_2 + X_1X_2$. However, when evaluated over $\mathbb{Z}$, $F^{\mathbb{Z}}(1, 1) = 4$ and $G^{\mathbb{Z}}(1, 1) = 2$.

Since we will essentially query $F^{\mathbb{Z}}$ on vectors with 0 or 1 values, we allow ourselves the notation $F^{\mathbb{Z}}(\boldsymbol{x})$ for $\boldsymbol{x} \in \mathbb{F}_2^n$. It is clear that $F^{\mathbb{Z}}(\boldsymbol{x}) \equiv F(\boldsymbol{x}) \bmod 2$.

Definition 5.13 ($\mathbb{Z}$-value vector)**.** Let P be a $\mathbb{Z}$-compatible circuit. With the above notation, we define the $\mathbb{Z}$-value vector of P by

$$v_{P^{\mathbb{Z}}} \stackrel{\text{def}}{=} \left(P^{\mathbb{Z}}(\boldsymbol{x}) \;\middle|\; \boldsymbol{x} \in \mathbb{F}_2^n \right).$$

Proposition 5.3 ($\mathbb{Z}$-Möbius transform)**.** *The map*

$$\Phi : \left\{ \begin{array}{rcl} \mathbb{Z}^{2^n} & \longrightarrow & \mathbb{Z}^{2^n} \\ \boldsymbol{a} & \longmapsto & \boldsymbol{b} \text{ with } b_{\boldsymbol{u}} = (-1)^{\mathrm{wt}(\boldsymbol{u})} \cdot \sum_{\boldsymbol{x} \preceq \boldsymbol{u}} (-1)^{\mathrm{wt}(\boldsymbol{x})} a_{\boldsymbol{x}} \text{ for all } \boldsymbol{u} \in \mathbb{F}_2^n \end{array} \right.$$

is bijective. We call it the $\mathbb{Z}$-Möbius transform*. Its inverse is*

$$\Phi^{-1} : \left\{ \begin{array}{rcl} \mathbb{Z}^{2^n} & \longrightarrow & \mathbb{Z}^{2^n} \\ \boldsymbol{b} & \longmapsto & \boldsymbol{a} \text{ with } a_{\boldsymbol{u}} = \sum_{\boldsymbol{x} \preceq \boldsymbol{u}} b_{\boldsymbol{x}} \text{ for all } \boldsymbol{u} \in \mathbb{F}_2^n. \end{array} \right.$$

Proof. Indeed, let $\boldsymbol{a} \in \mathbb{Z}^{2^n}$, $\boldsymbol{b} = \Phi(\boldsymbol{a})$ and $\boldsymbol{u} \in \mathbb{F}_2^n$. Then

$$\sum_{\boldsymbol{x} \preceq \boldsymbol{u}} b_{\boldsymbol{x}} = \sum_{\boldsymbol{x} \preceq \boldsymbol{u}} (-1)^{\mathrm{wt}(\boldsymbol{x})} \sum_{\boldsymbol{y} \preceq \boldsymbol{x}} (-1)^{\mathrm{wt}(\boldsymbol{y})} a_{\boldsymbol{y}} = \sum_{\boldsymbol{y} \preceq \boldsymbol{u}} a_{\boldsymbol{y}} \cdot \left(\sum_{\boldsymbol{x} \preceq \boldsymbol{u} + \boldsymbol{y}} (-1)^{\mathrm{wt}(\boldsymbol{x})} \right) = a_{\boldsymbol{u}}.$$

□

Definition 5.14 ($\mathbb{Z}$-ANF)**.** Let P be a $\mathbb{Z}$-compatible circuit. We define the $\mathbb{Z}$-ANF of P by

$$\left(p^{\mathbb{Z}}_{\boldsymbol{u}} \;\middle|\; \boldsymbol{u} \in \mathbb{F}_2^n \right) \stackrel{\text{def}}{=} \Phi\left(v_{P^{\mathbb{Z}}}\right).$$

If the $\mathbb{Z}$-ANF of P has only non-negative elements, then so does the $\mathbb{Z}$-value vector. However, the converse does not hold. For example, the $\mathbb{Z}$-value vector $(0, 1, 1, 1)$ corresponds to the $\mathbb{Z}$-ANF $(0, 1, 1, -1)$. We will then say that P is non-negative when its $\mathbb{Z}$-ANF only has non-negative elements. Since we defined the evaluation of a $\mathbb{Z}$-compatible circuit over $\mathbb{Z}$ with positive constants and additions, the $\mathbb{Z}$-ANF of a $\mathbb{Z}$-compatible circuit will always be non-negative.

It is easy to check that for all $\boldsymbol{u} \in \mathbb{F}_2$, $p^{\mathbb{Z}}_{\boldsymbol{u}} \equiv p_{\boldsymbol{u}} \bmod 2$. Hence, if we can compute the $\mathbb{Z}$-ANF of P, we consequently can compute the ANF of P. Moreover, the $\mathbb{Z}$-Möbius transform can be performed by a fast algorithm very similar to Algorithm 11 in $n2^{n-1}$ operations over $\mathbb{Z}$. Of course in practice, n is too large and we want a more clever algorithm when the ANF of P is rather sparse. We next propose such an algorithm that benefits from using the ring $\mathbb{Z}$ under the assumption that the $\mathbb{Z}$-ANF is sparse.

5.5.1.2 The algorithm

Let P be a $\mathbb{Z}$-compatible circuit of depth r such that $n^{(r)} = 1$ and let $n \stackrel{\text{def}}{=} n^{(0)}$. Our goal is to build an algorithm which computes the non-zero monomials in the ANF of P. This algorithm will be referred to as Algorithm 14 for which we give the pseudo code on Page 158. We first explain the ideas behind Algorithm 14.

Evaluating Boolean circuits over $\mathbb{Z}$ is interesting because there is the property of positivity on $\mathbb{Z}$: for an arbitrary set $\mathcal{U}$,

$$\sum_{\boldsymbol{u}\in\mathcal{U}} p^{\mathbb{Z}}_{\boldsymbol{u}} = 0 \text{ and } \forall \boldsymbol{u} \in \mathcal{U},\ p^{\mathbb{Z}}_{\boldsymbol{u}} \geqslant 0 \quad \Rightarrow \quad p^{\mathbb{Z}}_{\boldsymbol{u}} = 0 \text{ for all } \boldsymbol{u} \in \mathcal{U}.$$

The strategy will then be to compute values v equal to a sum of coefficients $p^{\mathbb{Z}}_{\boldsymbol{u}}$ — i.e. $v = \sum_{\boldsymbol{u}\in\mathcal{U}} p^{\mathbb{Z}}_{\boldsymbol{u}}$ — to discard entire sets $\mathcal{U}$ of $\boldsymbol{u}$ when the value v is zero. For example, we have that

$$P^{\mathbb{Z}}(1,\ldots,1) = \sum_{\boldsymbol{u}\in\mathbb{F}_2^n} p^{\mathbb{Z}}_{\boldsymbol{u}}.$$

If $P^{\mathbb{Z}}(1,\ldots,1) = 0$, then we can conclude with only one evaluation that $P^{\mathbb{Z}} = 0$ and $P = 0$. Similarly, if $P^{\mathbb{Z}}(1,\ldots,1,0) = 0$, we know that we can write $P^{\mathbb{Z}}(X) = X_{n-1} \cdot Q^{\mathbb{Z}}(X)$ because

$$P^{\mathbb{Z}}(1,\ldots,1,0) = \sum_{\boldsymbol{u}:\ u_{n-1}=0} p^{\mathbb{Z}}_{\boldsymbol{u}}.$$

The general formula we are going to use in the algorithm is, for any $\boldsymbol{u} \in \mathbb{F}_2^\ell$,

$$P^{\mathbb{Z}}(\mathbf{1}_{n-\ell}|\boldsymbol{u}) = \sum_{\boldsymbol{v}'\preceq(\mathbf{1}_{n-\ell}|\boldsymbol{u})} p^{\mathbb{Z}}_{\boldsymbol{v}'} = \sum_{\boldsymbol{v}\preceq\boldsymbol{u}} \sum_{\boldsymbol{w}\in\mathbb{F}_2^{n-\ell}} p^{\mathbb{Z}}_{(\boldsymbol{w}|\boldsymbol{v})}.$$

If we define $S^{(\ell)}_{\boldsymbol{u}} \stackrel{\text{def}}{=} \sum_{\boldsymbol{w}\in\mathbb{F}_2^{n-\ell}} p^{\mathbb{Z}}_{(\boldsymbol{w}|\boldsymbol{u})}$ — and consistently $S^{(n)}_{\boldsymbol{u}} = p^{\mathbb{Z}}_{\boldsymbol{u}}$ — then we have the following formula that will be extensively used in the algorithm:

$$S^{(\ell)}_{\boldsymbol{u}} = P^{\mathbb{Z}}(\mathbf{1}_{n-\ell}|\boldsymbol{u}) - \sum_{\boldsymbol{v}\prec\boldsymbol{u}} S^{(\ell)}_{\boldsymbol{v}}. \tag{5.5}$$

Besides, $S^{(\ell)}_{\boldsymbol{u}} > 0$ if and only if there exists $\boldsymbol{w} \in \mathbb{F}_2^{n-\ell}$ such that $p^{\mathbb{Z}}_{(\boldsymbol{w}|\boldsymbol{u})} > 0$. Consequently, for all $\ell < n$, $\ell' > \ell$, $\boldsymbol{u} \in \mathbb{F}_2^\ell$ and $\boldsymbol{w} \in \mathbb{F}_2^{\ell'-\ell}$, we have that

$$S^{(\ell)}_{\boldsymbol{u}} = 0 \Rightarrow S^{(\ell')}_{(\boldsymbol{w}\ |\ \boldsymbol{u})} = 0.$$

The idea of Algorithm 14 is to compute the sums $S^{(\ell)}_{\boldsymbol{u}}$ with Equation (5.5) without computing sums that are already known as being 0. The top of the tree computed by the algorithm is illustrated in Figure 5.1. Algorithm 14 starts with the computation of $S^{(0)} = P^{\mathbb{Z}}(\mathbf{1}_n)$, the root of the tree in Figure 5.1. The algorithm then computes the other nodes in a breadth-first way. Indeed, according to Equation (5.5), the computation of $S^{(\ell)}_{\boldsymbol{u}}$ needs one evaluation of $P^{\mathbb{Z}}$ and the sums $S^{(\ell)}_{\boldsymbol{v}}$ for $\boldsymbol{v} \prec \boldsymbol{u}$. In other words, to compute the value of a node, we need some values from nodes on the left at the same level in the tree.

Of course this tree has $2^{n+1} - 1$ nodes and can quickly become too big to be computed. But remember that if a node has the value zero, then all the nodes of its sub-tree are also zero and do

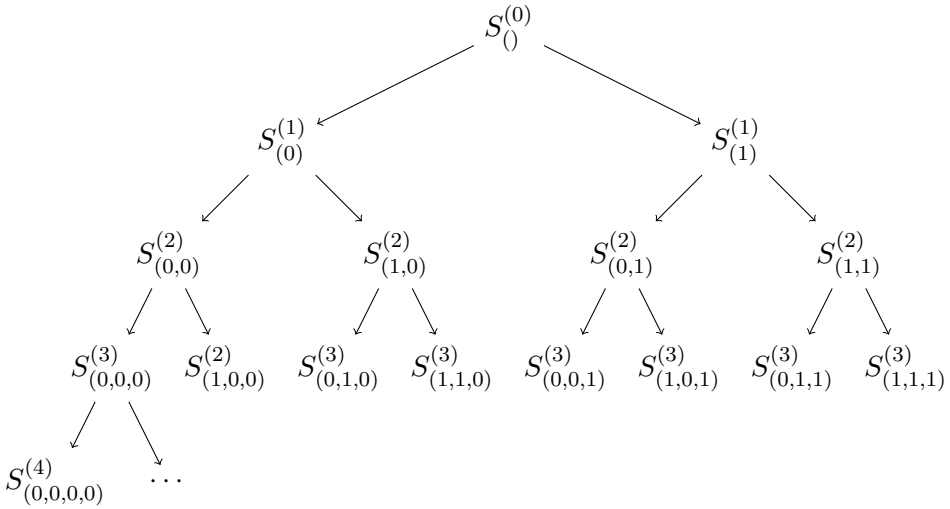

Figure 5.1 – Top of the tree computed by Algorithm 14. The nodes carry the values $S^{(\ell)}_{\boldsymbol{u}} = \sum_{\boldsymbol{w}\in\mathbb{F}_2^{n-\ell}} p^{\mathbb{Z}}_{(\boldsymbol{w}|\boldsymbol{u})}$ and the $\mathbb{Z}$-ANF of $P^{\mathbb{Z}}$ is then given by the leafs $S^{(n)}_{\boldsymbol{u}} = p^{\mathbb{Z}}_{\boldsymbol{u}}$.

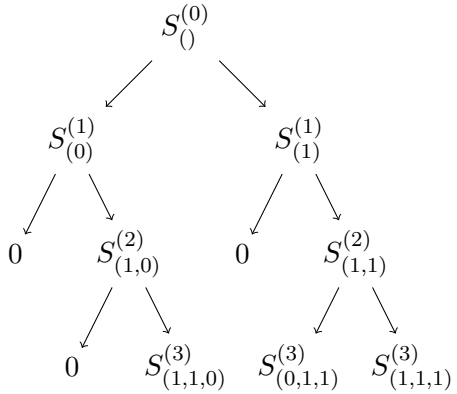

Figure 5.2 – Top of the tree computed by Algorithm 14 when $S^{(2)}_{(0,0)} = S^{(2)}_{(0,1)} = S^{(3)}_{(0,1,0)} = 0$

not need to be computed. For example, if it is found that $S^{(2)}_{(0,0)}$, $S^{(2)}_{(0,1)}$ and $S^{(3)}_{(0,1,0)}$ are zero, the tree computed by Algorithm 14 looks like the one in Figure 5.2.

Now that we understand what we need to compute a new node $S^{(\ell)}_{\boldsymbol{u}}$, we need to know *how* we can apply Equation (5.5). It is obvious that we first need to compute $P^{\mathbb{Z}}(\,\mathbf{1} \mid \boldsymbol{u}\,)$ with the given circuit but we also have to compute $\sum_{\boldsymbol{v}\prec\boldsymbol{u}} S^{(\ell)}_{\boldsymbol{v}}$. We just explained that the breadth-first computation of the tree made all the values of this sum available at that point: they are either non-zero, in which case they have been explicitely computed, either zero, in which case they belong to a non-computed zero sub-tree. Hence, to compute this sum, we just have to fetch the values with a depth-first search in the tree restricted to values $\boldsymbol{v} \prec \boldsymbol{u}$ and ignoring the zero sub-trees.

A small improvement to Algorithm 14 is to use a fast $\mathbb{Z}$-Möbius transform to directly compute a

Algorithm 14 Compute the $\mathbb{Z}$-ANF of a $\mathbb{Z}$-compatible circuit when it is sparse.

Binary tree. In this algorithm, a node N in a binary tree structure is a tuple $(N_{\text{label}}, N_{\text{left}}, N_{\text{right}})$ where $N_{\text{label}} \in \mathbb{N}$ is a label, N_{left} is another node called the left child of N and N_{right} is another node called the right child of N. The empty node is denoted by `nil`.

Lists. A list is a data structure with two methods: `push` adds an element at the end of the list and `pop` outputs and deletes an element in the list. If the list is a FIFO (First-In-First-Out), `pop` outputs the first element in the list, shifting all other elements in the list toward the first position. If the list is a LIFO (Last-In-First-Out), `pop` outputs the last element in the list. The empty list is denoted by `nil`.

1: **procedure** MAIN(P) ▷ Main procedure of the algorithm.
2: *input*
3: P: a $\mathbb{Z}$-compatible circuit with input size n and output size 1.
4: *output*
5: The $\mathbb{Z}$-ANF of P under the form $\left\{ \left(\boldsymbol{u}, p^{\mathbb{Z}}_{\boldsymbol{u}}\right) \in \mathbb{F}_2^n \times \mathbb{N} \;\middle|\; p^{\mathbb{Z}}_{\boldsymbol{u}} \neq 0 \right\}$.
6: `zanf` $\leftarrow$ `nil`.
7: $S^{(0)}_{()} \leftarrow P^{\mathbb{Z}}(\mathbf{1}_n)$.
8: `root` $\leftarrow \left(S^{(0)}_{()}, \texttt{nil}, \texttt{nil}\right)$.
9: `fifo` $\leftarrow$ `nil`; `fifo.push(root)`.
10: **while** `fifo` $\neq$ `nil` **do**
11: Let $N =$ `fifo.pop()`; let $\left(S^{(\ell)}_{\boldsymbol{u}}, \texttt{nil}, \texttt{nil}\right) = N$.
12: **if** $\ell < n$ **then**
13: `left-eval` $\leftarrow P^{\mathbb{Z}}(\,\mathbf{1}_{n-\ell-1} \mid 0 \mid \boldsymbol{u}\,)$.
14: `left-sum` $\leftarrow$ COMPUTE-SUM$((\,0 \mid \boldsymbol{u}\,))$.
15: $S^{(\ell+1)}_{(\,0 \mid \boldsymbol{u}\,)} \leftarrow$ `left-eval` $-$ `left-sum`. ▷ Equation (5.5)
16: $S^{(\ell+1)}_{(\,1 \mid \boldsymbol{u}\,)} \leftarrow S^{(\ell)}_{\boldsymbol{u}} - S^{(\ell+1)}_{(\,0 \mid \boldsymbol{u}\,)}$.
17: **if** $S^{(\ell+1)}_{(\,0 \mid \boldsymbol{u}\,)} \neq 0$ **then**
18: $N_{\text{left}} \leftarrow \left(S^{(\ell+1)}_{(\,0 \mid \boldsymbol{u}\,)}, \texttt{nil}, \texttt{nil}\right)$.
19: `fifo.push` (N_{left})
20: **if** $S^{(\ell+1)}_{(\,1 \mid \boldsymbol{u}\,)} \neq 0$ **then**
21: $N_{\text{right}} \leftarrow \left(S^{(\ell+1)}_{(\,1 \mid \boldsymbol{u}\,)}, \texttt{nil}, \texttt{nil}\right)$.
22: `fifo.push` (N_{right})
23: **else**
24: `zanf.push` $\left(\left(\boldsymbol{u}, S^{(n)}_{\boldsymbol{u}}\right)\right)$.
25: **return** `zanf`.

procedure COMPUTE-SUM(`root`, $\ell, \boldsymbol{u}$) ▷ Auxiliary procedure for Algorithm 14.
  input
    `root`: root of the tree.
    ℓ, $\boldsymbol{u} \in \mathbb{F}_2^\ell$: the vector $\boldsymbol{u}$ defines a path of depth ℓ in the tree.
  output
    The value $\sum_{\boldsymbol{v} \prec \boldsymbol{u}} S^{(\ell)}\boldsymbol{v}$.

  `sum` $\leftarrow 0 \in \mathbb{N}$.
  `lifo` $\leftarrow$ `nil`; `lifo.push` (`root`).
  while `lifo` $\neq$ `nil` **do**
    Let $N =$ `lifo.pop`(); let $\left(S_{\boldsymbol{v}}^{(i)}, N_{\text{left}}, N_{\text{right}}\right) = N$.
    if $N_{\text{left}} = N_{\text{right}} =$ `nil` **then**
      if $i = \ell$ **then** ▷ We only add $S_{\boldsymbol{v}}^{(\ell)}$ if the depth i is the same as the depth ℓ.
        `sum` $\leftarrow$ `sum` $+ S_{\boldsymbol{v}}^{(\ell)}$.
    else
      if $u_i = 1$ and $N_{\text{right}} \neq$ `nil` **then**
        `lifo.push` (N_{right}).
      `lifo.push` (N_{left}).
  return `sum`.

starting level of the tree $\ell > 0$. Indeed, the values $S_{\boldsymbol{u}}^{(\ell)},\ \boldsymbol{u} \in \mathbb{F}_2^\ell$ form the $\mathbb{Z}$-ANF of

$$P^{\mathbb{Z}}\left(\mathbf{1}_{n-\ell} \;\middle|\; X_{[n-\ell,n-1]}\right) = \sum_{\boldsymbol{u} \in \mathbb{F}_2^\ell} S_{\boldsymbol{u}}^{(\ell)} \cdot X_{[n-\ell,n-1]}^{\boldsymbol{u}}.$$

5.5.1.3 Complexity

Let $D \in \mathbb{N}$ be an upper bound on the number of non-zero coefficients of the $\mathbb{Z}$-ANF. Then at each level ℓ of the tree from Figure 5.1, there are at most D nodes since a given non-zero coefficient $p_{\boldsymbol{u}}^{\mathbb{Z}}$ appears in exactly one $S_{\boldsymbol{u}'}^{(\ell)}$ such that $\boldsymbol{u} = (\,\boldsymbol{w} \mid \boldsymbol{u}'\,)$. For each of the nD nodes, one evaluation of $P^{\mathbb{Z}}$ is needed. For each node at depth ℓ and position $d \in [1, D]$, the computation of the sum in Equation (5.5) with the depth-first search needs to visit $d-1$ leaves (at that point in the algorithm) at depth ℓ, hence a cost bounded by $\ell \cdot d$ comparisons (to navigate the tree) and d additions in $\mathbb{Z}$. By summing over d and ℓ, we get $O(n^2D^2)$ comparisons and $O(nD^2)$ additions. For superpoly recovery, the time complexity will be dominated by the evaluations of $P^{\mathbb{Z}}$ anyway.

To the best of our knowledge, this is the first time that such an upper bound on the complexity of computing an ANF as a function of its density (or number of non-zero coefficients) and a number of circuit evaluations. However, it is not clear how far D is from the number of non-zero coefficients in the $\mathbb{F}_2$-ANF. This will depend on how much the circuit wrongly diffuses monomials with an even coefficient: a low-depth circuit built with complex ANFs will keep this wrong diffusion low whereas a high-depth circuit built with basic operations is expected to give a significant gap between the two notions of density.

In practice, the integers computed by the evaluation of $P^{\mathbb{Z}}$ can be huge and we prefer performing these operations not in $\mathbb{Z}$ but rather $\mathbb{Z}/2^r\mathbb{Z}$ for a suitable power of two (64 is good for modern

Algorithm 15 Compute the superpoly of the cube $\mathcal{C}$ in $F^{\mathbb{Z}}$ if its $\mathbb{Z}$-ANF is sparse.

input
    F: the $\mathbb{Z}$-compatible circuit.
    $\mathcal{C} = (\mathcal{I}, \mathcal{J})$: the cube of which we want to recover the superpoly.
output
    The $\mathbb{Z}$-ANF of $P_{\mathcal{C}}^{\mathbb{Z}}(K)$.
return MAIN $\left(\boldsymbol{k} \in \mathbb{F}_2^m \mapsto (-1)^{|\mathcal{I}|} \cdot \sum_{\boldsymbol{x} \preceq \boldsymbol{e}_{\mathcal{I}}} (-1)^{\mathrm{wt}(\boldsymbol{x})} F^{\mathbb{Z}}(\boldsymbol{e}_{\mathcal{J}} + \boldsymbol{x}, \boldsymbol{k})\right)$.

processors). This comes with a small loss of precision: there is a risk that 2^r is considered to be 0 but the probability is only 2^{-r} if $P^{\mathbb{Z}}$ behaves randomly. We just have to choose r such that $nD \cdot 2^{-r}$ is negligeable.

5.5.2 Superpoly recovery with Möbius transforms

Let F be a $\mathbb{Z}$-compatible circuit with n public variables $X_0, \ldots, X_{n-1}$, m private variables $K_0, \ldots, K_{m-1}$ and only one output bit. In this section, we propose a method to find and compute superpolys in $F(X, K)$ which only works on cubes of practical size. $F^{\mathbb{Z}}$ is the function we get when we evaluate the circuit F over $\mathbb{Z}$ rather than $\mathbb{F}_2$.

5.5.2.1 Recovery on a given cube

We first explicit how to recover the superpoly of a given cube $\mathcal{C} = (\mathcal{I}, \mathcal{J})$. We want to use Algorithm 14 for the superpoly $P_{\mathcal{C}}(K)$ but we do not have a $\mathbb{Z}$-compatible circuit for this Boolean function. Nonetheless, what Algorithm 14 really needs is an access to queries of the form $\boldsymbol{k} \mapsto P_{\mathcal{C}}^{\mathbb{Z}}(\boldsymbol{k})$, and Proposition 5.3 gives the formula that allows to compute $P_{\mathcal{C}}^{\mathbb{Z}}(\boldsymbol{k})$ with the values $F^{\mathbb{Z}}(\boldsymbol{x}, \boldsymbol{k})$, $\boldsymbol{x} \in \mathcal{C}$. More precisely, we have that

$$P_{\mathcal{C}}^{\mathbb{Z}}(K) = (-1)^{|\mathcal{I}|} \cdot \sum_{\boldsymbol{x} \preceq \boldsymbol{e}_{\mathcal{I}}} (-1)^{\mathrm{wt}(\boldsymbol{x})} F^{\mathbb{Z}}(\boldsymbol{e}_{\mathcal{J}} + \boldsymbol{x}, K). \tag{5.6}$$

Therefore, we can compute $\boldsymbol{k} \mapsto P_{\mathcal{C}}^{\mathbb{Z}}(\boldsymbol{k})$ for all $\boldsymbol{k} \in \mathbb{F}_2^m$ with $2^{|\mathcal{I}|}$ evaluations of F over $\mathbb{Z}$. The complexity of the recovery will then be dominated by $m \cdot D \cdot 2^{|\mathcal{I}|}$ evaluations of $F^{\mathbb{Z}}$ where D is the number of non-zero monomials in the superpoly $P_{\mathcal{C}}^{\mathbb{Z}}(K)$. We highlight that Equation (5.6) does not artificially add any monomial with an even coefficient to $P_{\mathcal{C}}^{\mathbb{Z}}$ — which is a risk when performing evaluations over $\mathbb{Z}$ in general — and that D is really the density of the polynomial

$$P_{\mathcal{C}}^{\mathbb{Z}}(K) = \sum_{\boldsymbol{w} \preceq \boldsymbol{e}_{\mathcal{J}}} P_{\boldsymbol{w}+\boldsymbol{e}_{\mathcal{I}}}^{\mathbb{Z}}(K)$$

when $P_{\boldsymbol{u}}^{\mathbb{Z}}(K)$ is defined through the $\mathbb{Z}$-ANF of $F^{\mathbb{Z}}$ by

$$\forall (\boldsymbol{x}, \boldsymbol{k}) \in \mathbb{F}_2^{n+m},\ F^{\mathbb{Z}}(\boldsymbol{x}, \boldsymbol{k}) = \sum_{\boldsymbol{u} \in \mathbb{F}_2^n} P_{\boldsymbol{u}}^{\mathbb{Z}}(\boldsymbol{k}) \cdot \boldsymbol{x}^{\boldsymbol{u}}.$$

We sum up this method in Algorithm 15.

5.5.2.2 Finding cubes

Now we have to find cubes that might carry a sparse superpoly. To explain our method, we fix $\mathcal{J} = \emptyset$ since a different choice whould come from an analysis similar to the one of Section 5.3.1.4 and our method does not give any improvement in that respect. Our method simply consists in defining a "search-window" for cube variables and testing the sparsity of superpoly ANFs with the Möbius transform for all cubes in the search-window.

In the following, we write the $\mathbb{F}_2$-ANF of F as

$$F(X,K) = \sum_{\boldsymbol{u}\in\mathbb{F}_2^n} P_{\mathrm{supp}(\boldsymbol{u})}(K)\cdot X^{\boldsymbol{u}}, \qquad P_{\mathrm{supp}(\boldsymbol{u})}(K) = \sum_{\boldsymbol{v}\in\mathbb{F}_2^m} p_{\mathrm{supp}(\boldsymbol{u}),\boldsymbol{v}}\cdot K^{\boldsymbol{v}}.$$

Partial superpoly ANF. The goal is to find a cube $\mathcal{I}$ such that the superpoly

$$P_{\mathcal{I}}(K) = \sum_{\boldsymbol{v}\in\mathbb{F}_2^m} p_{\mathcal{I},\boldsymbol{v}}\cdot K^{\boldsymbol{v}}.$$

has a small number of non-zero coefficients $p_{\mathcal{I},\boldsymbol{v}}$. Let $m' < m$ and $D \in \mathbb{N}$. We have that

$$P_{\mathcal{I}}\left(K_{[0,m'-1]} \;\middle|\; \mathbf{1}_{m-m'}\right) = \sum_{\boldsymbol{v}\in\mathbb{F}_2^{m'}} \left(\sum_{\boldsymbol{w}\in\mathbb{F}_2^{m-m'}} p_{\mathcal{I},(\boldsymbol{v}|\boldsymbol{w})}\right)\cdot K_{[0,m'-1]}^{\boldsymbol{v}}.$$

Therefore, if $P_{\mathcal{I}}\left(K_{0,\ldots,m'-1} \mid \mathbf{1}_{m-m'}\right)$ has D non-zero coefficients, then $P_{\mathcal{I}}(K)$ has at least D non-zero coefficients. This means that from the ANF of $P_{\mathcal{I}}\left(K_{0,\ldots,m'-1} \mid \mathbf{1}_{m-m'}\right)$, with size $2^{m'}$ bits, we can discard the cube $\mathcal{I}$ if this ANF has too many non-zero coefficients. Counting the number of non-zero coefficients in the ANF of $P_{\mathcal{I}}\left(K_{[0,m'-1]} \;\middle|\; \mathbf{1}_{m-m'}\right)$ is similar to a one-sided tester for the sparsity of the superpoly $P_{\mathcal{I}}(K)$.

Search-window for cube variables. The search-window is defined by two sets $\mathcal{I}_{\mathrm{sup}}$ and $\mathcal{I}_{\mathrm{inf}} \subset \mathcal{I}_{\mathrm{sup}}$ as we will search for potential cube variables $\mathcal{I}$ between those sets: $\mathcal{I}_{\mathrm{inf}} \subseteq \mathcal{I} \subseteq \mathcal{I}_{\mathrm{sup}}$. An analysis similar to the one of Section 5.3.1.4 can give ideas on how to choose the search-window. We also choose a set of private variables $\mathcal{K}$. For an easier notation, let us assume without loss of generality that we chose $\mathcal{I}_{\mathrm{inf}} = [0, n_{\mathrm{inf}} - 1]$, $\mathcal{I}_{\mathrm{sup}} = [0, n_{\mathrm{sup}} - 1]$ and $\mathcal{K} = [0, m' - 1]$.

Computing $\mathbb{F}_2$-ANFs. Finally, for all $\mathcal{I}$ in the search-window, we compute the $\mathbb{F}_2$-ANF of the Boolean function $P_{\mathcal{I}}\left(K_{0,\ldots,m'-1} \mid \mathbf{1}_{m-m'}\right)$. This is equivalent to computing the ANF of

$$G(X_{[n_{\mathrm{inf}},n_{\mathrm{sup}}-1]}, K_{[0,m'-1]}) \stackrel{\mathrm{def}}{=} \sum_{\boldsymbol{y}\in\mathbb{F}_2^{n_{\mathrm{inf}}}} F\left(\left(\boldsymbol{y} \;\middle|\; X_{[n_{\mathrm{inf}},n_{\mathrm{sup}}-1]} \;\middle|\; \mathbf{0}\right), \left(K_{[0,m'-1]} \;\middle|\; \mathbf{1}\right)\right). \tag{5.7}$$

Indeed, the ANF of G is

$$G(X,K) = \sum_{\mathcal{I}:\ \mathcal{I}_{\mathrm{inf}}\subseteq\mathcal{I}\subseteq\mathcal{I}_{\mathrm{sup}}} P_{\mathcal{I}}\left(K_{[0,m'-1]} \;\middle|\; \mathbf{1}\right)\cdot X^{e_{\mathcal{I}}},$$

Note that we explained a similar idea in Section 5.2.2.2 in a raw setting with $\mathcal{I}_{\mathrm{sup}} = [1, n]$.

Algorithm 16 Search for cubes with a sparse superpoly.

input
Query access to F: $(\boldsymbol{x}, \boldsymbol{k}) \mapsto F(\boldsymbol{x}, \boldsymbol{k})$.
$\mathcal{I}_{\text{inf}}$, $\mathcal{I}_{\text{sup}}$: search-window for cube variables.
m': number of private variables; determines the sparsity threshold $2^{m'-2}$, over which a superpoly ANF is considered too dense.
output
A list of cubes with potentially sparse superpolys.

Compute the value vector of $G\left(X_{\mathcal{I}_{\text{sup}} \setminus \mathcal{I}_{\text{inf}}}, K_{0,m'-1}\right)$ with queries to F and Equation (5.7).

Compute the ANF vector of G from the value vector with the Möbius transform.

For all $\mathcal{I} \subseteq \mathcal{I}_{\text{sup}} \setminus \mathcal{I}_{\text{inf}}$, add $\mathcal{I} \cup \mathcal{I}_{\text{inf}}$ to the output set if the ANF-vector of $P_{\mathcal{I} \cup \mathcal{I}_{\text{inf}}}\left(K_{[0,m'-1]} \;\middle|\; \mathbf{1}\right)$, which is a sub-vector of the ANF-vector of G, has weight smaller than $2^{m'-1}$.

Sparsity threshold and complexity. To find a cube $\mathcal{C}$ which carries a sparse superpoly, we go over those sets $\mathcal{I}$ and look at the ANF of $P_{\mathcal{I}}\left(K_{0,\ldots,m'-1} \mid \mathbf{1}\right)$. If this ANF has strictly less than $2^{m'-1}$ monomials, there is a bias towards the entire superpoly being sparse. We personnally often used $2^{m'-2}$ as a sparsity threshold, with $m' = D + 2$ and D the maximum weight of ANF-vectors we tolerate in the superpolys we want to recover. Such parameters ensure we do not miss sparse superpolys during our search. We sum up this method in Algorithm 16.

This computation needs $2^{n_{\text{sup}}+m'}$ evaluations of F and a memory of $2^{n_{\text{inf}}+m'}$ bits. The fast Möbius transform will need $(n_{\text{inf}} + m') \cdot 2^{n_{\text{inf}}+m'-1}$ operations.

Application to `Trivium`. We applied Algorithm 16 to 11 output bits of `Trivium` after 800 initialization rounds with the parameters $m' = 15$, $n_{\text{inf}} = 7$, $n_{\text{sup}} = 27$ and well-chosen potential cube variables $\mathcal{I}_{\text{sup}}$. We found approximately 600 candidate cubes out of $11 \cdot 2^{27-7}$.

5.5.3 Link with monomial trails

5.5.3.1 Equality only for depth-1 circuits.

Le F be a $\mathbb{Z}$-compatible circuit of depth 1 such that $\pi^{(1)} \circ C^{(1)}(\boldsymbol{x}) = (\,\boldsymbol{x} \mid \cdots \mid \boldsymbol{x}\,)$. Then, if $a_{\boldsymbol{u},\boldsymbol{v}}$ is the coefficient of $X^{\boldsymbol{u}}$ in the $\mathbb{Z}$-ANF of $F^{\mathbb{Z}}(X)$, we have that

$$\ell\left(F, \boldsymbol{u} \to \boldsymbol{v}\right) = a_{\boldsymbol{u},\boldsymbol{v}}. \tag{5.8}$$

Indeed, for all $\boldsymbol{x} \in \mathbb{F}_2^{n^{(0)}}$,

$$F^{\mathbb{Z}}(\boldsymbol{x})^{\boldsymbol{v}} = \prod_{i \in \text{supp}(\boldsymbol{v})} \left(\sum_{\boldsymbol{u}:\ X^{\boldsymbol{u}} \in F_i^{(1)}(X)} \boldsymbol{x}^{\boldsymbol{u}} \right) = \sum_{\substack{\left(\boldsymbol{u}'_i \;\middle|\; i \in \text{supp}(\boldsymbol{v})\right):\\ \forall i,\ X^{\boldsymbol{u}'_i} \in F_i^{(1)}(X)}} \prod_{i \in \text{supp}(\boldsymbol{w})} \boldsymbol{x}^{\boldsymbol{u}'_i},$$

hence for all $\boldsymbol{u}, \boldsymbol{v}$,

$$a_{\boldsymbol{u},\boldsymbol{v}} = \left| \left\{ \left(\boldsymbol{u}'_i \mid i \in \operatorname{supp}(\boldsymbol{v}) \right) \;\middle|\; \forall i,\ X^{\boldsymbol{u}'_i} \in F_i^{(1)}(X) \text{ and } \bigvee_i \boldsymbol{u}'_i = \boldsymbol{u} \right\} \right|.$$

Besides,

$$\ell(F, \boldsymbol{u} \to \boldsymbol{v}) = \left| \left\{ \boldsymbol{u}' \in \left(\mathbb{F}_2^{n^{(0)}}\right)^{n^{(1)}} \;\middle|\; \begin{array}{l} \forall i \in \operatorname{supp}(\boldsymbol{v}),\ X^{\boldsymbol{u}'_i} \in F_i^{(1)}(X) \\ \forall i \notin \operatorname{supp}(\boldsymbol{v}),\ \boldsymbol{u}'_i = \boldsymbol{0} \\ \text{and } \bigvee_i \boldsymbol{u}'_i = \boldsymbol{u} \end{array} \right\} \right|,$$

hence Equation (5.8). However, the relation of Equation (5.8) does not generalize to $\mathbb{Z}$-compatible circuits with higher depths. Indeed, the coefficients of $F^{\mathbb{Z}}$ grow much faster than numbers of monomial trails because of the multiplication over $\mathbb{Z}$.

Consider for example the following $\mathbb{Z}$-compatible circuit of depth 2.

$$\begin{pmatrix} X_1 \\ X_2 \end{pmatrix} \xmapsto{\pi \circ C} \begin{pmatrix} X'_1 \\ X'_2 \\ X'_3 \\ X'_4 \end{pmatrix} \xmapsto{F^{(1)}} \begin{pmatrix} 1 + X'_1 X'_2 \\ X'_3 X'_4 \end{pmatrix} \qquad \begin{pmatrix} Y_1 \\ Y_2 \end{pmatrix} \xmapsto{\pi \circ C} \begin{pmatrix} Y'_1 \\ Y'_2 \\ Y'_3 \\ Y'_4 \end{pmatrix} \xmapsto{F^{(2)}} \begin{pmatrix} 1 + Y'_1 + Y'_2 \\ Y'_3 Y'_4 \end{pmatrix}.$$

Then for all $\boldsymbol{x} \in \mathbb{F}_2^2$,

$$F^{\mathbb{Z}}(\boldsymbol{x})^{(1,1)} = 3x_1 + 3x_2 + 14x_1x_2,$$

and for all $\boldsymbol{u}$,

$$\ell(F, \boldsymbol{u} \to (1,1)) = 3 \cdot \ell\left(\left(\pi \circ C, F^{(1)}\right), \boldsymbol{u} \to (1,1)\right),$$

and in particular $\ell(F, (1,1) \to (1,1)) = 6 \neq 14$.

5.5.3.2 Semi-evaluated circuits.

Let F be a Boolean circuit $\left(F^{(1)}, \ldots, F^{(r)}\right)$ of depth r and such that $F^{(1)}$ operates on two different sets of variables $X_0, \ldots, X_{n-1}$ and $K_0, \ldots, K_{m-1}$. Let $\boldsymbol{k} \in \mathbb{F}_2^m$. We build a new circuit $G_{\boldsymbol{k}}$ from F such that $\forall \boldsymbol{x} \in \mathbb{F}_2^n$, $G_{\boldsymbol{k}}(\boldsymbol{x}) = F(\boldsymbol{x}, \boldsymbol{k})$ with a natural recursive approach. The coordinate functions of $G_{\boldsymbol{k}}^{(1)}$ will have the ANFs of the coordinate functions of $F^{(1)}(X, \boldsymbol{k})$. We suppose $G_{\boldsymbol{k}}^{(1)}, \ldots, G_{\boldsymbol{k}}^{(r-1)}$ already built and we denote by $\mathcal{I}$ the set of coordinates of $G_{\boldsymbol{k}}^{(r-1)}$ with constant ANFs. Let

$$\sigma : \left\{ \begin{array}{ccc} \mathbb{F}_2^n & \longrightarrow & \mathbb{F}_2^n \\ \boldsymbol{x} & \longmapsto & (\,(\,x_i \mid i \notin \mathcal{I}\,) \mid (\,x_i \mid i \in \mathcal{I}\,)\,) \end{array} \right.,$$

then we define

$$G_{\boldsymbol{k}}^{(r)} \stackrel{\text{def}}{=} F^{(r)} \circ \sigma^{-1}\left(X_0, \ldots, X_{n^{(r-1)}-|\mathcal{I}|-1} \;\middle|\; \left(G_{\boldsymbol{k},i}^{(r-1)} \;\middle|\; i \in \mathcal{I}\right)\right).$$

With such a construction, we have $G_{\boldsymbol{k}}(X) = F(X, \boldsymbol{k})$. Then, considering F has exactly one output bit and writing $F(X, K) = \sum_{\boldsymbol{u} \in \mathbb{F}_2^n} P_{\boldsymbol{u}}(K) X^{\boldsymbol{u}}$, we have with Proposition 5.2 that, for all $\boldsymbol{u} \in \mathbb{F}_2^n$ and $\boldsymbol{k} \in \mathbb{F}_2^m$,

$$P_{\boldsymbol{u}}(\boldsymbol{k}) = 1 \iff \ell(G_{\boldsymbol{k}}, \boldsymbol{u} \to 1) \equiv 1 \bmod 2.$$

In other words, the vector

$$(\ell\,(G_{\boldsymbol{k}}, \boldsymbol{u} \to 1) \mid \boldsymbol{k} \in \mathbb{F}_2^m\,) \tag{5.9}$$

is a consistent $\mathbb{Z}$-value vector for the superpoly $P_{\boldsymbol{u}}(K)$.

To recover the ANF of this superpoly, it would be tempting to use Algorithm 14 with the query access $\boldsymbol{k} \mapsto P_{\boldsymbol{u}}^{\mathbb{Z}}(\boldsymbol{k})$ given by the computation of the quantity $\ell\,(G_{\boldsymbol{k}}, \boldsymbol{u} \to 1)$ with the techniques discussed in Section 5.4.4. Indeed, when the method from [Hao+20b], discussed in Section 5.4.5 and based on the computation of

$$\{\, \boldsymbol{u}_K \mid \ell\,(F, (\boldsymbol{u}_X, \boldsymbol{u}_K) \to 1) \neq 0 \,\}$$

for a given cube $\mathrm{Prec}\,(\boldsymbol{u}_X)$, is impractical for the enumeration tool, the same enumeration tool could be used several times on the simpler circuits $G_{\boldsymbol{k}}$ by the queries of Algorithm 14. The problem is that the $\mathbb{Z}$-value vector of Equation (5.9) was not obtained through the evaluation of a $\mathbb{Z}$-compatible circuit. Then the corresponding $\mathbb{Z}$-ANF could have negative coefficients and we could encounter the situation where $S_{\boldsymbol{w}}^{(\ell)} < S_{(\,0 \mid \boldsymbol{w}\,)}^{(\ell+1)}$ for some $\ell \in \mathbb{N}$ and $\boldsymbol{w} \in \mathbb{F}_2^\ell$. Therefore, Algorithm 14 is not correct in this situation.

It is not clear for the moment whether Algorithm 14 could be adapted to a non-negative $\mathbb{Z}$-value vector with possibly negative $\mathbb{Z}$-ANF coefficients. Meanwhile, the enumeration of monomial trails in a semi-evaluated circuit $G_{\boldsymbol{k}}$ is a potential tool to evaluate superpolys on big-size cubes for which the Möbius transform is impractical. This could be interesting to mount cube testers or dynamic cube attacks as those attacks mostly require the evaluation of superpolys.

5.6 Conclusion

In this chapter, we gave an overview of the different techniques used to analyze ANFs and their applications to cube attacks. We have seen that their are two families of techniques for ANF analysis: those based on the computation of Möbius transforms and those based on the enumeration of monomial trails.

Techniques based on Möbius transform computations are limited to the computational power available to the cryptanalyst, and thus fail at providing strong and convincing security arguments against algebraic attacks beyond the practical range. Indeed, computing an ANF with the Möbius transform requires a time and data complexity exponential in the number of involved variables whatever the number of non-zero coefficients in the ANF. We then proposed a new technique based on Möbius transform computations and evaluations of circuits over $\mathbb{Z}$ to challenge this natural limitation. In particular, for cube attacks, our technique seems to improve the recovery of sparse-ANF superpolys but it is still limited to practical-size cubes because of Möbius transform computations. If it might be interesting to investigate this technique further and to find applications, there remains a major theoretical uncertainty: the gap between the density of a $\mathbb{Z}$-ANF and the density of the corresponding $\mathbb{F}_2$-ANF. Indeed, this gap highly depends on the circuit evaluated over $\mathbb{Z}$.

Techniques based on monomial trails — or division property — have allowed more precise analysis for big-size cubes. Indeed, the paradigm of monomial trails is completely different from Möbius transform computations as it relies on tracing the existence of monomials through the rounds of a cipher, or through the gates of a circuit. If monomial trails have allowed cryptographers to provide much better ANF-analysis than Möbius transform-based techniques so far, the enumeration of monomial trails remains a heavy computation highly dependent on the circuit under study.

Moreover, similarly to our new Möbius transform-based technique, the unknown gap between the complexity of the enumeration of monomial trails and the density of the ANF brings uncertainty when the enumeration exceeds practical computational resources.

Finally, the enumeration of monomial trails in semi-evaluated circuits could help to surpass the current limitations of monomial trails in two directions. First, it could allow to mount property-testing-based cube attacks since they allow to evaluate superpolys of big-size cubes. Second, with our views on ANFs with integer coefficients, it could help to exhibit a new superpoly-recovery algorithm for big-size cubes.

Chapter 6

Study of a McEliece trapdoor function with rank metric

Contributions brought forward in this chapter were published in DCC 2020 and are a joint work with Alain Couvreur [CC20].

6.1 Introduction

To instantiate a McEliece encryption scheme (see Section 1.6.3), one needs a family of codes with random-looking generator matrices and an efficient decoding algorithm. While the original proposal due to McEliece himself [McE78] relies on classical Goppa codes endowed with the Hamming metric, one can actually consider codes endowed with any other metric. The use of $\mathbb{F}_{q^m}$–linear rank metric codes, first suggested by Gabidulin et al. [GPT91] is of particular interest, since the $\mathbb{F}_{q^m}$–linearity permits a very "compact" representation of the code and hence permits to design a public-key encryption scheme with rather short keys compared to the original McEliece proposal.

Compared to the Hamming-metric world, only a few families of codes with efficient decoding algorithms are known in rank-metric. Basically, the McEliece scheme has been instantiated with two general families of rank metric codes, namely Gabidulin codes [Del78; Gab85] and LRPC codes [Gab+13].

In [Loi17], Loidreau proposed the use of codes which can in some sense be regarded as an intermediary version between Gabidulin codes and LRPC codes. These codes are obtained by right multiplying a Gabidulin code with an invertible matrix whose entries are in $\mathbb{F}_{q^m}$ and span an $\mathbb{F}_q$–subspace of small dimension λ. This approach can be regarded as a "rank metric" counterpart of the BBCRS scheme [Bal+16] in Hamming metric.

In the present chapter, we explain why Loidreau's scheme is weak when $\lambda = 2$ and the dimension of the public code $\mathscr{C}_{\mathrm{pub}}$ is bigger than half the length n: $\dim \mathscr{C}_{\mathrm{pub}} \geqslant n/2$. We finally describe a polynomial-time key-recovery attack in this situation.

6.1.1 Rank metric codes

In this article m, n denote positive integers and q a prime power. A *code of dimension k* is an $\mathbb{F}_{q^m}$–subspace of $\mathbb{F}_{q^m}^n$ whose dimension as an $\mathbb{F}_{q^m}$–vector space is k. Given a vector $\boldsymbol{x} \in \mathbb{F}_{q^m}^n$, the *rank weight* or *rank* of $\boldsymbol{x}$, denoted as $|\boldsymbol{x}|_{\mathrm{R}}$ is the dimension of the $\mathbb{F}_q$–vector subspace of $\mathbb{F}_{q^m}$ spanned

by the entries of $\boldsymbol{x}$. The *support* of a vector $\boldsymbol{x} \in \mathbb{F}_{q^m}^n$, denoted $\mathrm{supp}_{\mathrm{R}}(\boldsymbol{x})$ is the $\mathbb{F}_q$–vector space spanned by the entries of $\boldsymbol{x}$. Hence the rank of $\boldsymbol{x}$ is nothing but the dimension of its support. The *rank distance* or *distance* of two vectors $\boldsymbol{x}, \boldsymbol{y} \in \mathbb{F}_{q^m}^n$ is defined as

$$\mathrm{d}_{\mathrm{R}}(\boldsymbol{x}, \boldsymbol{y}) \stackrel{\text{def}}{=} |\boldsymbol{x} - \boldsymbol{y}|_{\mathrm{R}}.$$

Given a linear code $\mathscr{C} \subseteq \mathbb{F}_{q^m}^n$, the *minimum distance* of $\mathscr{C}$ is defined as

$$\mathrm{d}_{\min}(\mathscr{C}) \stackrel{\text{def}}{=} \min_{\boldsymbol{x} \in \mathscr{C} \setminus \{0\}} |\boldsymbol{x}|_{\mathrm{R}}.$$

Finally, given a code $\mathscr{C} \subseteq \mathbb{F}_{q^m}^n$, the *dual* code of $\mathscr{C}$, denoted by $\mathscr{C}^\perp$, is the orthogonal of $\mathscr{C}$ with respect to the canonical inner product in $\mathbb{F}_{q^m}$:

$$\begin{cases} \mathbb{F}_{q^m}^n \times \mathbb{F}_{q^m}^n & \longrightarrow & \mathbb{F}_{q^m} \\ (\boldsymbol{x}, \boldsymbol{y}) & \longmapsto & \langle \boldsymbol{x} \mid \boldsymbol{y} \rangle = \sum_{i=1}^n x_i y_i. \end{cases}$$

6.1.2 q–polynomials and Gabidulin codes

A *q–polynomial* or a *linearized polynomial* is an $\mathbb{F}_{q^m}$–linear combination of monomials

$$X, X^q, X^{q^2}, \ldots, X^{q^s}, \ldots$$

Such a polynomial induces a function $\mathbb{F}_{q^m} \to \mathbb{F}_{q^m}$ which is $\mathbb{F}_q$–linear. The *q–degree* of a q-polynomial P, denoted by $\deg_q P$ is the integer s such that the degree of P is q^s. In short:

$$P = \sum_{i=0}^{\deg_q P} p_i X^{q^i},\ p_i \in \mathbb{F}_{q^m},\ p_{\deg_q P} \neq 0.$$

The space of q-polynomials is denoted by $\mathcal{L}$ and, given a positive integer s, the space of q-polynomials of degree less than s is denoted by

$$\mathcal{L}_{<s} \stackrel{\text{def}}{=} \left\{ P \in \mathcal{L} \;\middle|\; \deg_q P < s \right\}.$$

Given positive integers k, n with $k \leqslant n \leqslant m$ and an n-tuple $\boldsymbol{a} = (a_1, \ldots, a_n)$ of $\mathbb{F}_q$-linearly independent elements of $\mathbb{F}_{q^m}$, the *Gabidulin code* $\mathcal{G}_k(\boldsymbol{a})$ is defined as

$$\mathcal{G}_k(\boldsymbol{a}) \stackrel{\text{def}}{=} \{ (f(a_1), \ldots, f(a_n)) \mid f \in \mathcal{L}_{<k} \}.$$

This code has a generator matrix of the form:

$$\begin{pmatrix} a_1 & \ldots & a_n \\ a_1^q & \ldots & a_n^q \\ \vdots & & \vdots \\ a_1^{q^{k-1}} & \ldots & a_n^{q^{k-1}} \end{pmatrix}.$$

Such codes are known to have minimum distance $n - k + 1$ and to benefit from a decoding algorithm correcting up to half the minimum distance (see [Loi06]).

Note that the vector $\boldsymbol{a}$ is not unique as shown by the following lemma which will be useful for our attack.

Lemma 6.1 ([Ber03, Theorem 2])**.** *Let $\alpha \in \mathbb{F}_{q^m}^*$. Then $\mathcal{G}_k(\boldsymbol{a}) = \mathcal{G}_k(\alpha \boldsymbol{a})$.*

Proof. Let $P \in \mathbb{F}_{q^m}[X]$ be a q–polynomial of q–degree $< k$ and $Q \in \mathbb{F}_{q^m}[X]$ be the q–polynomial of the same degree defined by $Q(X) = P(\alpha^{-1}X)$. Then the codeword $(P(a_1), \ldots, P(a_n)) \in \mathcal{G}_k(\boldsymbol{a})$ is equal to the codeword $(Q(\alpha a_1), \ldots, Q(\alpha a_n)) \in \mathcal{G}_k(\alpha \boldsymbol{a})$. This proves that $\mathcal{G}_k(\boldsymbol{g}) \subseteq \mathcal{G}_k(\alpha \boldsymbol{g})$ and the converse inclusion is proved in a similar fashion. □

6.1.3 The component-wise Frobenius map

In what follows, we will frequently apply the component-wise Frobenius map or its iterates to vectors or codes. Hence, we introduce the following notation. Given a vector $\boldsymbol{v} \in \mathbb{F}_{q^m}^n$ and a non-negative integer s, we denote by $\boldsymbol{v}^{[s]}$ the vector:

$$\boldsymbol{v}^{[s]} \stackrel{\text{def}}{=} \left(v_1^{q^s},\ \ldots,\ v_n^{q^s}\right).$$

Similarly, given a code $\mathscr{C} \subseteq \mathbb{F}_{q^m}^n$ and a positive integer s, the code $\mathscr{C}^{[s]}$ denotes the code

$$\mathscr{C}^{[s]} \stackrel{\text{def}}{=} \left\{ \boldsymbol{c}^{[s]} \;\middle|\; \boldsymbol{c} \in \mathscr{C} \right\}.$$

$\mathscr{C}^{[s]}$ is an $\mathbb{F}_{q^m}$–linear code of the same dimension as $\mathscr{C}$ since $\boldsymbol{x} \mapsto \boldsymbol{x}^{[s]}$ is an $\mathbb{F}_q$–linear bijection.

6.2 Distinguishing Gabidulin codes

In [Ove08], Overbeck proposes a general framework to break cryptosystems based on Gabidulin codes. Since the work presented in this chapter was inspired by this framework, we briefly explain his fundamental ideas in Section 6.2.1 and we redirect the interested reader to [Ove08] for a more in-depth study.

6.2.1 Overbeck's distinguisher

The core of Overbeck's attack is that a simple operation allows us to distinguish Gabidulin codes from random ones. Indeed, given a random code $\mathscr{C} \subseteq \mathbb{F}_{q^m}^n$ of dimension $k < n/2$, the expected dimension of the code $\mathscr{C} + \mathscr{C}^{[1]}$ equals $2k$ and, equivalently $\mathscr{C} \cap \mathscr{C}^{[1]}$ is likely to be equal to 0. More generally, we have the following statement.

Proposition 6.1. *If $\mathscr{C}_{\text{rand}}$ is a code of length n and dimension k chosen uniformly at random, then for a non-negative integer a and for a positive integer $s < k$, we have*

$$\mathbb{P}\left(\dim_{\mathbb{F}_{q^m}} \mathscr{C}_{\text{rand}} + \mathscr{C}_{\text{rand}}{}^{[1]} + \cdots + \mathscr{C}_{\text{rand}}{}^{[s]} \leqslant \min\left(n, (s+1)k\right) - a\right) = O\left(q^{-ma}\right).$$

Proof. See Section 6.2.2. □

On the other hand, for a Gabidulin code, the behaviour with respect to such operations is completely different, as explained in the following statement.

Proposition 6.2. *Let $\boldsymbol{a} \in \mathbb{F}_{q^m}^n$ be a word of rank n, $k \leqslant n$ and s be an integer. Then,*

$$\mathcal{G}_k(\boldsymbol{a}) \cap \mathcal{G}_k(\boldsymbol{a})^{[1]} = \mathcal{G}_{k-1}\left(\boldsymbol{a}^{[1]}\right); \tag{6.1}$$

$$\mathcal{G}_k(\boldsymbol{a}) + \cdots + \mathcal{G}_k(\boldsymbol{a})^{[s]} = \mathcal{G}_{k+s}(\boldsymbol{a}). \tag{6.2}$$

Proof. We first prove Equation (6.2). Note that for all i, $\mathcal{G}_k(\boldsymbol{a})^{[i]} = \mathcal{G}_k\left(\boldsymbol{a}^{[i]}\right)$ is spanned by $\left\{ \boldsymbol{a}^{[i+j]} \;\middle|\; j \in [0, k-1] \right\}$. Therefore, $\mathcal{G}_k(\boldsymbol{a}) + \cdots + \mathcal{G}_k(\boldsymbol{a})^{[s]}$ is spanned by

$$\bigcup_{i=0}^{s} \left\{ \boldsymbol{a}^{[i+j]} \;\middle|\; j \in [0, k-1] \right\} = \left\{ \boldsymbol{a}^{[i]} \;\middle|\; i \in [0, k+s-1] \right\}$$

and hence is equal to $\mathcal{G}_{k+s}(\boldsymbol{a})$.

For Equation (6.1), we have that

$$\mathcal{G}_{k-1}\left(\boldsymbol{a}^{[1]}\right) \subseteq \mathcal{G}_k(\boldsymbol{a}) \cap \mathcal{G}_k(\boldsymbol{a})^{[1]}$$

because the generators $\boldsymbol{a}^{[i]}$ for $i \in [1, k-1]$ of $\mathcal{G}_{k-1}\left(\boldsymbol{a}^{[1]}\right)$ are in $\mathcal{G}_k(\boldsymbol{a}) \cap \mathcal{G}_k(\boldsymbol{a})^{[1]}$. Conversely, the two codes have the same dimension because, with Equation (6.2) and $s = 1$, we have that

$$2k - \dim \mathcal{G}_k(\boldsymbol{a}) \cap \mathcal{G}_k(\boldsymbol{a})^{[1]} = \dim \mathcal{G}_k(\boldsymbol{a}) + \mathcal{G}_k(\boldsymbol{a})^{[1]} = \dim \mathcal{G}_{k+1}(\boldsymbol{a}) = k+1.$$

□

Example 6.1. We illustrate in this example the difference of behaviour between Gabidulin codes and random codes. If $\mathscr{G}$ is a Gabidulin code of length n and dimension $k < n/2$, then $\dim \mathscr{G} + \mathscr{G}^{[1]} = k+1$. For a random code $\mathscr{C}_{\text{rand}}$ of the same length and dimension,

$$\lim_{m \to +\infty} \mathbb{P}\left(\dim_{\mathbb{F}_{q^m}} \mathscr{C}_{\text{rand}} + \mathscr{C}_{\text{rand}}{}^{[1]} = 2k \right) = 1.$$

6.2.2 Proof of Proposition 6.1

Since we could not find any proof for Proposition 6.1 in the literature, we propose one here for completeness. We begin with a classical combinatorial result on subspaces of $\mathbb{F}_{q^m}$.

Preliminaries on Gaussian binomial coefficients

Notation 6.1. In what follows, we denote by $\begin{bmatrix} a \\ b \end{bmatrix}_{q^m}$ the Gaussian binomial coefficient representing the number of subspaces of dimension b of a vector space of dimension a over $\mathbb{F}_{q^m}$.

Lemma 6.2. *There exists a positive constant C such that for any pair of positive integers n, k such that $n \geqslant k$, we have*

$$q^{k(n-k)} \leqslant \begin{bmatrix} n \\ k \end{bmatrix}_q \leqslant C \cdot q^{k(n-k)}.$$

Proof. By definition of Gaussian binomials, we have

$$\begin{bmatrix} n \\ k \end{bmatrix}_q = \prod_{t=0}^{k-1} \frac{q^n - q^t}{q^k - q^t} = q^{k(n-k)} \prod_{t=0}^{k-1} \frac{1 - q^{t-n}}{1 - q^{t-k}}.$$

Since $n \geqslant k$, we get

$$\prod_{t=0}^{k-1} \frac{1 - q^{t-n}}{1 - q^{t-k}} \geqslant 1,$$

which yields the left-hand inequality. To get the other equality, we need to bound from above the product:

$$\prod_{t=0}^{k-1} \frac{1-q^{t-n}}{1-q^{t-k}} \leqslant \prod_{t=0}^{k-1} \frac{1}{1-q^{t-k}} = \prod_{j=0}^{k-1} \frac{1}{1-\frac{1}{q^{j+1}}},$$

where the last equality is obtained by applying the change of variables $j = k-1-t$. Set

$$a_k \stackrel{\text{def}}{=} \prod_{j=0}^{k-1} \frac{1}{1-\frac{1}{q^{j+1}}}.$$

The sequence a_k is increasing and converges. Indeed,

$$\log(a_k) = \sum_{j=0}^{k-1} -\log\left(1-\frac{1}{q^{j+1}}\right)$$

and the series with general term $-\log(1-1/q^{j+1})$ converges. As a conclusion, the right-hand inequality is obtained by taking

$$C \stackrel{\text{def}}{=} \prod_{j=0}^{\infty} \frac{1}{1-\frac{1}{q^{j+1}}}.$$

□

Remark 6.1. A finer analysis would permit to prove that $C \leqslant \left(\frac{q}{q-1}\right)^{\frac{q}{q-1}}$. In particular, since $q \geqslant 2$, we have that $C \leqslant 4$.

Lemma 6.3. *Let $\mathscr{C}_{\text{rand}}$ be an $\mathbb{F}_{q^m}$–linear code of length n and dimension k chosen uniformly at random and $\mathscr{A}$ be a subspace of $\mathbb{F}_{q^m}^n$ of dimension $t \leqslant k$. Then*

$$\mathbb{P}\left(\mathscr{A} \subseteq \mathscr{C}_{\text{rand}}\right) \leqslant C \cdot q^{-mt(n-k)},$$

where C is the constant of Lemma 6.2.

Proof. We have

$$\mathbb{P}\left(\mathscr{A} \subseteq \mathscr{C}_{\text{rand}}\right) = \begin{bmatrix} n-t \\ k-t \end{bmatrix}_{q^m} \cdot \begin{bmatrix} n \\ k \end{bmatrix}_{q^m}^{-1}.$$

Using Lemma 6.2 we get the upper bound,

$$\mathbb{P}\left(\mathscr{A} \subseteq \mathscr{C}_{\text{rand}}\right) \leqslant C \cdot q^{m(k-t)(n-k)} \cdot q^{-mk(n-k)} = C \cdot q^{-mt(n-k)}.$$

□

Expected dimension of $\mathscr{C}_{\text{rand}} + \mathscr{C}_{\text{rand}}{}^{[1]} + \cdots + \mathscr{C}_{\text{rand}}{}^{[s]}$

Let $\mathscr{C}_{\text{rand}}$ be a subspace of $\mathbb{F}_{q^m}^n$ chosen uniformly at random among its subspaces of dimension k. From $\mathscr{C}_{\text{rand}}$ we build the map

$$\Psi : \left\{ \begin{array}{ccc} \mathscr{C}_{\text{rand}}{}^{s+1} & \longrightarrow & \mathbb{F}_{q^m}^n \\ (\boldsymbol{c}_0, \ldots, \boldsymbol{c}_s) & \longmapsto & \boldsymbol{c}_0 + \boldsymbol{c}_1{}^{[1]} + \cdots + \boldsymbol{c}_s{}^{[s]}. \end{array} \right.$$

The image of this map is $\mathscr{C}_{\text{rand}} + \mathscr{C}_{\text{rand}}^{[1]} + \cdots + \mathscr{C}_{\text{rand}}^{[s]}$ and hence the dimension of $\mathscr{C}_{\text{rand}} + \mathscr{C}_{\text{rand}}^{[1]} + \cdots + \mathscr{C}_{\text{rand}}^{[s]}$ is related to the dimension of the kernel of Ψ. Therefore, our approach will consist in estimating $\mathbb{E}(|\ker \Psi|)$. We have

$$\mathbb{E}(|\ker \Psi|) = \sum_{\substack{\boldsymbol{x}_0,\ \ldots,\ \boldsymbol{x}_s\ \in \mathbb{F}_{q^m}^n \text{ s.t.} \\ \boldsymbol{x}_0 + \boldsymbol{x}_1^{[1]} + \cdots + \boldsymbol{x}_s^{[s]} = 0}} \mathbb{P}(\boldsymbol{x}_0, \ldots, \boldsymbol{x}_s \in \mathscr{C}_{\text{rand}}). \tag{6.3}$$

For any $0 \leqslant t \leqslant s+1$, we introduce the set

$$\mathcal{S}_t \stackrel{\text{def}}{=} \left\{ (\boldsymbol{x}_0, \ldots, \boldsymbol{x}_s) \in \left(\mathbb{F}_{q^m}^n\right)^s \;\middle|\; \begin{array}{rcl} \boldsymbol{x}_0 + \boldsymbol{x}_1^{[1]} + \cdots + \boldsymbol{x}_s^{[s]} & = & 0 \\ \dim_{\mathbb{F}_{q^m}} \langle \boldsymbol{x}_0, \ldots, \boldsymbol{x}_s \rangle & = & t \end{array} \right\}.$$

Thanks to Equation (6.3) and Lemma 6.3, we can write that

$$\mathbb{E}(|\ker \Psi|) \leqslant C \cdot \sum_{t=0}^{s+1} q^{-mt(n-k)} |\mathcal{S}_t|. \tag{6.4}$$

Lemma 6.4. *Let $1 \leqslant t \leqslant k-1$. Then $|\mathcal{S}_t| \leqslant C \cdot q^{(mt+n)(s+1-t)+mn(t-1)}$.*

Proof. Let $(\boldsymbol{x}_0, \ldots, \boldsymbol{x}_s) \in \mathcal{S}_t$. Since the $\boldsymbol{x}_i$'s span a space of dimension t, there exists a unique $(s+1-t) \times (s+1)$ full-rank matrix $\boldsymbol{M}$ in reduced echelon form with entries in $\mathbb{F}_{q^m}$ such that

$$\boldsymbol{M} \cdot \begin{pmatrix} \boldsymbol{x}_0 \\ \hline \vdots \\ \hline \boldsymbol{x}_s \end{pmatrix} = 0. \tag{6.5}$$

Let us count the number of possible $(s+1)$–tuples $(\boldsymbol{x}_0, \ldots, \boldsymbol{x}_s)$ satisfying Equation (6.5) and such that $\boldsymbol{x}_0 + \boldsymbol{x}_1^{[1]} + \cdots + \boldsymbol{x}_s^{[s]} = 0$. For any $1 \leqslant i \leqslant n$, we have

$$x_{0,i} + x_{1,i}^q + \cdots + x_{s,i}^{q^s} = 0 \tag{6.6}$$

$$\text{and} \qquad \boldsymbol{M} \cdot \begin{pmatrix} x_{0,i} \\ \vdots \\ x_{s,i} \end{pmatrix} = 0 \tag{6.7}$$

Let us label the columns of $\boldsymbol{M}$ from 0 to s. Let $\mathcal{P} \subseteq [0, s]$ be the set of indices of columns which are pivots for $\boldsymbol{M}$ and $\mathcal{P}^c$ its complementary. We denote by a the smallest element of $\mathcal{P}^c$. Notice that

$$|\mathcal{P}^c| = t \qquad \text{and} \qquad a \leqslant s+1-t. \tag{6.8}$$

In Equation (6.6), we can eliminate any $x_{j,i}$ where $j \in \mathcal{P}$ using Equation (6.7). By this manner we get an expression depending only on the $x_{j,i}$'s for $j \in \mathcal{P}^c$. If we fix the value of the $x_{r,i}$'s for any $r \in \mathcal{P}^c \setminus \{a\}$, we obtain an equation of the form

$$Q(x_{a,i}) = 0,$$

where Q is a q–polynomial of q–degree at most a. There are then at most q^a possible values for $x_{a,i}$ for any choice of the elements $x_{r,i}$ with $r \in \mathcal{P}^c \setminus \{i\}$. Using Equation (6.8) we deduce that there are at most

$$q^{m(t-1)+a} \leqslant q^{m(t-1)+s+1-t}$$

possible choices for the tuple $(x_{0,i}, \ldots, x_{s,i})$. Consequently, Equation (6.5) has at most $q^{mn(t-1)+n(s+1-t)}$ solutions.

Finally, since the full-rank $(s+1-t) \times (s+1)$ matrices in row echelon form are in one–to–one correspondence with t–dimensional subspaces of $\mathbb{F}_{q^m}^{s+1}$ there are $\begin{bmatrix} s+1 \\ t \end{bmatrix}_{q^m}$ possible choices for $\boldsymbol{M}$. Using Lemma 6.2, we deduce the result. □

Combining Equation (6.4) and Lemma 6.4, we get

$$\begin{aligned}\mathbb{E}\left(|\ker \Psi|\right) &\leqslant C^2 \cdot \sum_{t=0}^{s+1} q^{-mt(n-k)+(mt+n)(s+1-t)+mn(t-1)} \\ &\leqslant C^2 \cdot q^{m(k(s+1)-n)} \cdot \sum_{t=0}^{s+1} q^{(s+1-t)(mt+n-mk)}.\end{aligned}$$

By assumption (in the statement of Proposition 6.1), we have $s < k$. Next, since $n \leqslant m$, we see that the exponents in the above sum are all less than or equal to 0. More precisely,

$$\begin{aligned}\sum_{t=0}^{s+1} q^{(s+1-t)(mt+n-mk)} &\leqslant 2 + \sum_{t=0}^{s-1} q^{(s+1-t)(mt+n-mk)} \\ &\leqslant 2 + \sum_{t=0}^{s-1} q^{-m(s+1-t)} \\ &\leqslant 2 + \sum_{i=0}^{s-1} q^{-m(i+2)} \leqslant 2 + \frac{q^{-2m}}{1-q^{-m}}.\end{aligned}$$

Consequently, we have the following result.

Corollary 6.1. *There is a positive constant C' such that*

$$\mathbb{E}\left(|\ker \Psi|\right) \leqslant C' \cdot q^{m(k(s+1)-n)}.$$

End of the proof

To conclude the proof of Proposition 6.1, we have from the rank theorem that

$$\begin{aligned}\dim_{\mathbb{F}_{q^m}} \mathscr{C}_{\text{rand}} + \mathscr{C}_{\text{rand}}{}^{[1]} + \cdots + \mathscr{C}_{\text{rand}}{}^{[s]} \leqslant \min\{k(s+1), n\} - a \\ \Rightarrow \dim_{\mathbb{F}_{q^m}} \ker \Psi \geqslant \max\{0, k(s+1)-n\} + a.\end{aligned}$$

Using Markov inequality together with Corollary 6.1, we get

$$\mathbb{P}\left(|\ker \Psi| \geqslant q^{m(\max\{0,k(s+1)-n\}+a)}\right) \leqslant C' \cdot \frac{q^{m(k(s+1)-n)}}{q^{m(\max\{0,k(s+1)-n\}+a)}} \leqslant C' \cdot q^{-ma}.$$

6.3 Loidreau's scheme

In order to mask the structure of Gabidulin codes and to resist Overbeck's attack, Loidreau suggested in [Loi17] the following construction. Denote by $\boldsymbol{G}$ a random generator matrix of a Gabidulin code $\mathcal{G}_k(\boldsymbol{b})$. Fix an integer $\lambda \leqslant m$ and an $\mathbb{F}_q$–vector subspace $\mathcal{V}$ of $\mathbb{F}_{q^m}$ of dimension λ. Let $\boldsymbol{P} \in \mathrm{GL}_n(\mathbb{F}_{q^m})$ whose entries are all in $\mathcal{V}$. Then, let

$$\boldsymbol{G}_{\mathrm{pub}} \stackrel{\mathrm{def}}{=} \boldsymbol{G}\boldsymbol{P}^{-1}.$$

We have the following encryption scheme.

Public key. The pair $(\boldsymbol{G}_{\mathrm{pub}}, t)$ where $t \stackrel{\mathrm{def}}{=} \left\lfloor \frac{n-k}{2\lambda} \right\rfloor$.

Private key. The pair $(\boldsymbol{b}, \boldsymbol{P})$.

Encryption. Given a plain text $\boldsymbol{m} \in \mathbb{F}_{q^m}^k$, choose a uniformly random vector $\boldsymbol{e} \in \mathbb{F}_{q^m}^n$ of rank weight t. The cipher text is

$$\boldsymbol{c} \stackrel{\mathrm{def}}{=} \boldsymbol{m}\boldsymbol{G}_{\mathrm{pub}} + \boldsymbol{e}.$$

Decryption: Compute $\boldsymbol{c}\boldsymbol{P} = \boldsymbol{m}\boldsymbol{G} + \boldsymbol{e}\boldsymbol{P}$. Since the entries of $\boldsymbol{P}$ are all in $\mathcal{V}$, then the entries of $\boldsymbol{e} \cdot \boldsymbol{P}$ are in the product space

$$\mathrm{supp}_{\mathrm{R}}(\boldsymbol{e}) \cdot \mathcal{V} \stackrel{\mathrm{def}}{=} \mathrm{vect}_{\mathbb{F}_q}\left\{u \cdot v \mid u \in \mathrm{supp}_{\mathrm{R}}(\boldsymbol{e}),\ v \in \mathcal{V}\right\}.$$

The dimension of this space is bounded from above by $t\lambda \leqslant \frac{n-k}{2}$. Therefore, using a classical decoding algorithm for Gabidulin codes, one can recover $\boldsymbol{m}$.

Remark 6.2. Loidreau's scheme can be considered as a generalization of the GPT cryptosystem proposed in [GO01], for which the parameter λ would be equal to 1. This cryptosystem was broken by Overbeck in [Ove08] thanks to the distinguishers presented in Section 6.2.1. The hope of Loidreau in [Loi17] is that higher values of λ, while increasing the key size, could counter this kind of attacks.

6.4 A distinguisher against Loidreau's scheme

6.4.1 Context

The goal of this section is to establish a distinguisher for Loidreau's cryptosystem instantiated with $\lambda = 2$ and a public code $\mathscr{C}_{\mathrm{pub}}$ of dimension $k \geqslant \frac{n}{2}$. Similar to Overbeck's attack, this distinguisher relies on Propositions 6.1 and 6.2. As for the attacks against the BBCRS system [Cou+14; Cou+15], it is more convenient to work on the dual of the public code (defined in Section 6.1.1) because of the following lemmas.

Lemma 6.5 ([Gab85, Theorem 7])**.** *Let $\boldsymbol{b} \in \mathbb{F}_{q^m}^n$ of rank n. The code $\mathcal{G}_k(\boldsymbol{b})^{\perp}$ is a Gabidulin code $\mathcal{G}_{n-k}(\boldsymbol{a})$ for some $\boldsymbol{a} \in \mathbb{F}_{q^m}^n$ of rank n.*

Lemma 6.6. *Any full-rank generator matrix $\boldsymbol{H}_{\mathrm{pub}}$ of ${\mathscr{C}_{\mathrm{pub}}}^{\perp}$ can be decomposed as*

$$\boldsymbol{H}_{\mathrm{pub}} = \boldsymbol{H}_{\mathrm{sec}}\boldsymbol{P}^T$$

where $\boldsymbol{H}_{\mathrm{sec}}$ is a parity-check matrix of the Gabidulin code $\mathcal{G}_k(\boldsymbol{b})$.

Proof. Let $\boldsymbol{H}_{\text{pub}}$ be a full–rank generator matrix of ${\mathscr{C}_{\text{pub}}}^{\perp}$. Since $\mathscr{C}_{\text{pub}} = \mathscr{C}_{\text{sec}} P^{-1}$, there exists a full-rank generator matrix $\boldsymbol{G}_{\text{sec}}$ of $\mathscr{C}_{\text{sec}}$ such that

$$\boldsymbol{H}_{\text{pub}} \left(\boldsymbol{G}_{\text{sec}} P^{-1}\right)^T = 0 \qquad \text{or equivalently} \qquad \boldsymbol{H}_{\text{pub}} \left(P^{-1}\right)^T \boldsymbol{G}_{\text{sec}}^T = 0.$$

Which means that $\boldsymbol{H}_{\text{pub}} \left(P^{-1}\right)^T$ is a parity-check matrix of $\mathscr{C}_{\text{sec}} = \mathcal{G}_k(\boldsymbol{b})$. □

The convenient aspect of the previous lemma is that the matrix $\boldsymbol{P}$ has its entries in a small vector space, while its inverse does not.

6.4.2 The case $\lambda = 2$

We suppose in this section that the vector space $\mathcal{V} \subseteq \mathbb{F}_{q^m}^n$ in which the matrix $\boldsymbol{P}$ has all its entries has dimension 2:

$$\lambda = \dim_{\mathbb{F}_q} \mathcal{V} = 2.$$

Note that, w.l.o.g, one can suppose that $1 \in \mathcal{V}$. Indeed, if $\mathcal{V}$ is spanned over $\mathbb{F}_q$ by $\alpha, \beta \in \mathbb{F}_{q^m} \setminus \{0\}$, then one can replace $\boldsymbol{H}_{\text{sec}}$ by $\boldsymbol{H}'_{\text{sec}} = \alpha \boldsymbol{H}_{\text{sec}}$ which spans the *same code*, $\boldsymbol{P}' = \alpha^{-1}\boldsymbol{P}$ has entries in $\mathcal{V}' = \text{vect}_{\mathbb{F}_q}(1, \alpha^{-1}\beta)$, and $\boldsymbol{H}_{\text{pub}} = \boldsymbol{H}'_{\text{sec}} \boldsymbol{P}'^T$.

Thus, from now on, we suppose that $\mathcal{V} = \text{vect}_{\mathbb{F}_q}(1, \gamma)$ for some $\gamma \in \mathbb{F}_{q^m} \setminus \mathbb{F}_q$. Consequently, $\boldsymbol{P}^T$ can be decomposed as

$$\boldsymbol{P}^T = \boldsymbol{P}_0 + \gamma \boldsymbol{P}_1,$$

where $\boldsymbol{P}_0, \boldsymbol{P}_1$ are square matrices with entries in $\mathbb{F}_q$. We have seen that ${\mathscr{C}_{\text{sec}}}^{\perp} = \mathcal{G}_{n-k}(\boldsymbol{a})$ for some $\boldsymbol{a} \in \mathbb{F}_{q^m}^n$ with $|\boldsymbol{a}|_{\text{R}} = n$. We define

$$\boldsymbol{g} \stackrel{\text{def}}{=} \boldsymbol{a}\boldsymbol{P}_0 \qquad \text{and} \qquad \boldsymbol{h} \stackrel{\text{def}}{=} \boldsymbol{a}\boldsymbol{P}_1.$$

Lemma 6.7. *The code ${\mathscr{C}_{\text{pub}}}^{\perp}$ is spanned by*

$$\boldsymbol{g} + \gamma \boldsymbol{h},\ \boldsymbol{g}^{[1]} + \gamma \boldsymbol{h}^{[1]},\ \ldots,\ \boldsymbol{g}^{[n-k-1]} + \gamma \boldsymbol{h}^{[n-k-1]}.$$

Proof. For any $\boldsymbol{c} \in {\mathscr{C}_{\text{pub}}}^{\perp}$ there exists $P \in \mathcal{L}_{<n-k}$ such that

$$\boldsymbol{c} = P(\boldsymbol{a})\boldsymbol{P}^T = P(\boldsymbol{a})\boldsymbol{P}_0 + \gamma P(\boldsymbol{a})\boldsymbol{P}_1 = P(\boldsymbol{g}) + \gamma P(\boldsymbol{h}),$$

which yields the result. □

We can now state a crucial result.

Theorem 6.1. *The dual of the public code satisfies:*

$$\dim_{\mathbb{F}_{q^m}} {\mathscr{C}_{\text{pub}}}^{\perp} + {\mathscr{C}_{\text{pub}}}^{\perp[1]} + {\mathscr{C}_{\text{pub}}}^{\perp[2]} \leqslant 2 \dim_{\mathbb{F}_{q^m}} {\mathscr{C}_{\text{pub}}}^{\perp} + 2.$$

Proof. Thanks to Lemma 6.7, we prove that ${\mathscr{C}_{\text{pub}}}^{\perp} + {\mathscr{C}_{\text{pub}}}^{\perp[1]}$ is spanned by

$$\boldsymbol{g} + \gamma \boldsymbol{h},\ \ldots,\ \boldsymbol{g}^{[n-k-1]} + \gamma \boldsymbol{h}^{[n-k-1]} \qquad \text{and} \qquad \boldsymbol{g}^{[1]} + \gamma^q \boldsymbol{h}^{[1]},\ \ldots,\ \boldsymbol{g}^{[n-k]} + \gamma^q \boldsymbol{h}^{[n-k]}.$$

Equivalently, ${\mathscr{C}_{\mathrm{pub}}}^{\perp} + {\mathscr{C}_{\mathrm{pub}}}^{\perp^{[1]}}$ is spanned by:

$$\boldsymbol{g} + \gamma \boldsymbol{h} \qquad \text{and} \qquad \boldsymbol{g}^{[1]},\ \boldsymbol{h}^{[1]}, \qquad \ldots, \qquad \boldsymbol{g}^{[n-k-1]},\ \boldsymbol{h}^{[n-k-1]} \qquad \text{and} \qquad \boldsymbol{g}^{[n-k]} + \gamma^q \boldsymbol{h}^{[n-k]}.$$

Finally, a similar reasoning allows us to show that ${\mathscr{C}_{\mathrm{pub}}}^{\perp} + {\mathscr{C}_{\mathrm{pub}}}^{\perp^{[1]}} + {\mathscr{C}_{\mathrm{pub}}}^{\perp^{[2]}}$ is spanned by

$$\boldsymbol{g} + \gamma \boldsymbol{h} \qquad \text{and} \qquad \boldsymbol{g}^{[1]},\ \boldsymbol{h}^{[1]}, \qquad \ldots, \qquad \boldsymbol{g}^{[n-k]},\ \boldsymbol{h}^{[n-k]} \qquad \text{and} \qquad \boldsymbol{g}^{[n-k+1]} + \gamma^{q^2} \boldsymbol{h}^{[n-k+1]},$$

and hence has dimension at most $2(n-k)+2$. □

As a conclusion, thanks to Proposition 6.1, we deduce that ${\mathscr{C}_{\mathrm{pub}}}^{\perp}$ is distinguishable in polynomial time from a random code as soon as $2k - 2 > n$ and $k < n - 2$.

6.4.3 The case $\lambda > 2$

In the general case, the approach consists in computing

$${\mathscr{C}_{\mathrm{pub}}}^{\perp} + {\mathscr{C}_{\mathrm{pub}}}^{\perp^{[1]}} + \cdots + {\mathscr{C}_{\mathrm{pub}}}^{\perp^{[\lambda]}}.$$

A similar reasoning permits to prove that the dimension of this space is bounded from above by $\lambda \dim {\mathscr{C}_{\mathrm{pub}}}^{\perp} + \lambda$. Thanks to Proposition 6.1, such a code is distinguishable from a random one if

$$\lambda \dim {\mathscr{C}_{\mathrm{pub}}}^{\perp} + \lambda < \min\left\{n, (\lambda+1) \dim {\mathscr{C}_{\mathrm{pub}}}^{\perp}\right\}.$$

That is to say

$$\lambda(n-k+1) < n \qquad \text{and} \qquad \lambda(n-k+1) < (\lambda+1)(n-k).$$

In summary, the public code is distinguishable from a random one if its dimension k satisfies:

$$n\left(1 - \frac{1}{\lambda}\right) + 1 < k < n - \lambda.$$

6.5 The attack

In this section, we derive an attack from the distinguisher defined in Section 6.4. In what follows, we suppose that $\lambda = 2$ and the public code has rate larger than $1/2$ so that the distinguisher introduced in Section 6.4 works on it. Recall that ${\mathscr{C}_{\mathrm{pub}}}^{\perp} = \mathcal{G}_{n-k}(\boldsymbol{a})\,\boldsymbol{P}$ for some $\boldsymbol{a} \in \mathbb{F}_{q^m}^n$ whose entries are $\mathbb{F}_q$–independent and $\boldsymbol{P}$ is of the form $\boldsymbol{P}_0 + \gamma \boldsymbol{P}_1$ for $\boldsymbol{P}_0, \boldsymbol{P}_1 \in \mathcal{M}_n(\mathbb{F}_q)$ and $\gamma \in \mathbb{F}_{q^m} \setminus \mathbb{F}_q$. Finally, recall that

$$\boldsymbol{g} \stackrel{\text{def}}{=} \boldsymbol{a}\boldsymbol{P}_0 \qquad \text{and} \qquad \boldsymbol{h} \stackrel{\text{def}}{=} \boldsymbol{a}\boldsymbol{P}_1.$$

In addition, we make the following assumptions:

(1) $\mathcal{G}_{n-k+2}(\boldsymbol{g}) \cap \mathcal{G}_{n-k+2}(\boldsymbol{h}) = \{0\}$, $|\boldsymbol{g}|_{\mathrm{R}} \geqslant n-k+2$ and $|\boldsymbol{h}|_{\mathrm{R}} \geqslant n-k+2$;

(2) $m > 2$ and γ is not contained in any subfield of $\mathbb{F}_{q^m}$.

Comments on the assumptions. According to our experiments using `Magma` [BCP97], Assumption (1) holds with a high probability. For instance, we ran 1000 tests for parameters: $q = 2, m = n = 30$ and $k = 17$. The average rank of $\boldsymbol{g}, \boldsymbol{h}$ is 29.1 and their minimum rank is 27. In addition, for these 1000 tests, the intersection $\mathcal{G}_{n-k+2}(\boldsymbol{g}) \cap \mathcal{G}_{n-k+2}(\boldsymbol{h})$ was always $\{0\}$. Finally, Assumption (2) is reasonable in order to prevent against possible attacks based on an exhaustive search for γ.

The aim of the attack is to recover the triple $(\gamma, \boldsymbol{g}, \boldsymbol{h})$, or more precisely, to recover a triple $(\gamma', \boldsymbol{g}', \boldsymbol{h}')$ such that

$$\mathscr{C}_{\text{pub}}{}^{\perp} = \text{vect}\left\{\boldsymbol{g}'^{[i]} + \gamma' \boldsymbol{h}'^{[i]} \;\middle|\; i \in [0, n-k-1]\right\}. \tag{6.9}$$

Actually, the triple $(\gamma, \boldsymbol{g}, \boldsymbol{h})$ is far from being unique and any other triple satisfying Equation (6.9) allows us to decrypt messages (see further Section 6.5.3). Let us describe an action of $\mathbf{PGL}(2, \mathbb{F}_q)$ on such triples.

Proposition 6.3. *Let $a, b, c, d \in \mathbb{F}_q$ such that $ad - bc \neq 0$ and $\delta \in \mathbb{F}_{q^m}$ such that $\gamma = \frac{a\delta+b}{c\delta+d}$. Then, the triple $(\delta, d\boldsymbol{g} + b\boldsymbol{h}, c\boldsymbol{g} + a\boldsymbol{h})$ satisfies Equation (6.9).*

Proof. It suffices to observe that for any $i \geqslant 0$,

$$\boldsymbol{g}^{[i]} + \frac{a\delta + b}{c\delta + d}\boldsymbol{h}^{[i]} = \frac{1}{c\delta + d}\left((d\boldsymbol{g} + b\boldsymbol{h})^{[i]} + \delta\,(c\boldsymbol{g} + a\boldsymbol{h})^{[i]}\right).$$

□

6.5.1 Step 1: using the distinguisher to compute some subcodes

As shown in the proof of Theorem 6.1, $\mathscr{C}_{\text{pub}}{}^{\perp} + \mathscr{C}_{\text{pub}}{}^{\perp[1]}$ is spanned by:

$$\boldsymbol{g} + \gamma\boldsymbol{h} \qquad \text{and} \qquad \boldsymbol{g}^{[1]}, \boldsymbol{h}^{[1]}, \qquad \ldots, \qquad \boldsymbol{g}^{[r]}, \boldsymbol{h}^{[r]} \qquad \text{and} \qquad \boldsymbol{g}^{[r+1]} + \gamma^q \boldsymbol{h}^{[r+1]},$$

where $r \stackrel{\text{def}}{=} n - k - 1$.

Lemma 6.8. *Under Assumption (1), we have that*

$$\left(\mathscr{C}_{\text{pub}}{}^{\perp} + \mathscr{C}_{\text{pub}}{}^{\perp[1]}\right) \cap \left(\mathscr{C}_{\text{pub}}{}^{\perp[1]} + \mathscr{C}_{\text{pub}}{}^{\perp[2]}\right)$$

is spanned by:

$$\boldsymbol{g}^{[1]} + \gamma^q \boldsymbol{h}^{[1]} \qquad \textit{and} \qquad \boldsymbol{g}^{[2]}, \boldsymbol{h}^{[2]}, \qquad \ldots, \qquad \boldsymbol{g}^{[r]}, \boldsymbol{h}^{[r]} \qquad \textit{and} \qquad \boldsymbol{g}^{[r+1]} + \gamma^q \boldsymbol{h}^{[r+1]}.$$

Proof. Indeed, if a vector $\boldsymbol{x}$ belongs to this intersection, we have that

$$\begin{aligned}\boldsymbol{x} &= x_{0,0}(\boldsymbol{g} + \gamma\boldsymbol{h}) + \sum_{i=1}^{r}\left(x_{0,i}\boldsymbol{g}^{[i]} + y_{0,i}\boldsymbol{h}^{[i]}\right) + x_{0,r+1}\left(\boldsymbol{g}^{[r+1]} + \gamma^q\boldsymbol{h}^{[r+1]}\right)\\ &= x_{1,0}\left(\boldsymbol{g}^{[1]} + \gamma^q\boldsymbol{h}^{[1]}\right) + \sum_{i=1}^{r}\left(x_{1,i}\boldsymbol{g}^{[i+1]} + y_{1,i}\boldsymbol{h}^{[i+1]}\right) + x_{1,r+1}\left(\boldsymbol{g}^{[r+2]} + \gamma^{q^2}\boldsymbol{h}^{[r+2]}\right).\end{aligned}$$

Hence

$$\begin{aligned} x_{0,0}\boldsymbol{g} + (x_{0,1} - x_{1,0})\boldsymbol{g}^{[1]} + \sum_{i=2}^{r}(x_{0,i} - x_{1,i-1})\boldsymbol{g}^{[i]} + (x_{0,r+1} - x_{1,r})\boldsymbol{g}^{[r+1]} - x_{1,r+1}\boldsymbol{g}^{[r+2]} \\ = -x_{0,0}\gamma\boldsymbol{h} + (\gamma^q x_{1,0} - y_{0,1})\boldsymbol{h}^{[1]} + \sum_{i=2}^{r}(y_{1,i-1} - y_{0,i})\boldsymbol{h}^{[i]} + (y_{1,r} - x_{0,r+1}\gamma^q)\boldsymbol{h}^{[r+1]} \\ + x_{1,r+1}\gamma^{q^2}\boldsymbol{h}^{[r+2]}. \end{aligned}$$

Since by Assumption (1) $\mathcal{G}_{r+3}(\boldsymbol{g}) \cap \mathcal{G}_{r+3}(\boldsymbol{h}) = \{0\}$, both sides of this equation are zero and in particular, $x_{0,0} = x_{1,r+1} = 0$ and $x_{0,1} = x_{1,0} = \gamma^{-q} y_{0,1}$. This finally gives that

$$\boldsymbol{x} \in \operatorname{vect}\left(\boldsymbol{g}^{[1]} + \gamma^q \boldsymbol{h}^{[1]},\ \boldsymbol{g}^{[2]}, \boldsymbol{h}^{[2]},\ \ldots,\ \boldsymbol{g}^{[r]}, \boldsymbol{h}^{[r]},\ \boldsymbol{g}^{[r+1]} + \gamma^q \boldsymbol{h}^{[r+1]}\right).$$

□

Then, a proof by induction shows that iterating intersections

$$\left(\mathscr{C}_{\text{pub}}{}^{\perp} + \mathscr{C}_{\text{pub}}{}^{\perp^{[1]}}\right) \cap \left(\mathscr{C}_{\text{pub}}{}^{\perp^{[1]}} + \mathscr{C}_{\text{pub}}{}^{\perp^{[2]}}\right) \cap \cdots \cap \left(\mathscr{C}_{\text{pub}}{}^{\perp^{[r]}} + \mathscr{C}_{\text{pub}}{}^{\perp^{[r+1]}}\right),$$

yields the code spanned by

$$\boldsymbol{g}^{[r]} + \gamma^{q^r}\boldsymbol{h}^{[r]} \quad \text{and} \quad \boldsymbol{g}^{[r+1]} + \gamma^q \boldsymbol{h}^{[r+1]}.$$

Applying the inverse of the r–th Frobenius, we get the code spanned by

$$\boldsymbol{g} + \gamma\boldsymbol{h} \quad \text{and} \quad \boldsymbol{g}^{[1]} + \gamma^{q^{1-r}}\boldsymbol{h}^{[1]}.$$

Next, one can compute

$$\mathscr{C}_{\text{pub}}{}^{\perp} \cap \operatorname{vect}\left(\boldsymbol{g} + \gamma\boldsymbol{h},\ \boldsymbol{g}^{[1]} + \gamma^{q^{1-r}}\boldsymbol{h}^{[1]}\right) = \operatorname{vect}(\boldsymbol{g} + \gamma\boldsymbol{h}), \tag{6.10}$$

Indeed, if $\boldsymbol{x} \in \mathscr{C}_{\text{pub}}{}^{\perp} \cap \operatorname{vect}\left(\boldsymbol{g} + \gamma\boldsymbol{h},\ \boldsymbol{g}^{[1]} + \gamma^{q^{1-r}}\boldsymbol{h}^{[1]}\right)$, we can write for some x and y,

$$\boldsymbol{x} = \sum_{i=0}^{r} x_i \left(\boldsymbol{g}^{[i]} + \gamma\boldsymbol{h}^{[i]}\right) = y_0(\boldsymbol{g} + \gamma\boldsymbol{h}) + y_1\left(\boldsymbol{g}^{[1]} + \gamma^{q^{1-r}}\boldsymbol{h}^{[1]}\right).$$

Assumption (1) gives that $x_1 = y_1$ and $x_1\gamma = y_1\gamma^{q^{1-r}}$. Finally with Assumption (2), we deduce $y_1 = 0$ and hence $\boldsymbol{x} \in \operatorname{vect}(\boldsymbol{g} + \gamma\boldsymbol{h})$.

Now, we can compute

$$\begin{aligned} &\operatorname{vect}\left(\boldsymbol{g} + \gamma\boldsymbol{h},\ \boldsymbol{g}^{[1]} + \gamma^{q^{1-r}}\boldsymbol{h}^{[1]}\right) + \operatorname{vect}(\boldsymbol{g} + \gamma\boldsymbol{h})^{[1]} \\ &= \operatorname{vect}\left(\boldsymbol{g} + \gamma\boldsymbol{h},\ \boldsymbol{g}^{[1]} + \gamma^{q^{1-r}}\boldsymbol{h}^{[1]},\ \boldsymbol{g}^{[1]} + \gamma^q\boldsymbol{h}^{[1]}\right) \\ &= \operatorname{vect}\left(\boldsymbol{g} + \gamma\boldsymbol{h},\ \boldsymbol{g}^{[1]},\ \boldsymbol{h}^{[1]}\right). \end{aligned}$$

Similarly, we compute the intersection with ${\mathscr{C}_{\text{pub}}}^{\perp^{[-1]}} \stackrel{\text{def}}{=} {\mathscr{C}_{\text{pub}}}^{\perp^{[m-1]}}$ and get with Assumptions (1) and (2)

$$ {\mathscr{C}_{\text{pub}}}^{\perp^{[-1]}} \cap \text{vect}\left(\boldsymbol{g}+\gamma\boldsymbol{h},\ \boldsymbol{g}^{[1]},\ \boldsymbol{h}^{[1]}\right) = \text{vect}\left(\boldsymbol{g}^{[1]}+\gamma^{q^{m-1}}\boldsymbol{h}^{[1]}\right). \tag{6.11} $$

Applying the inverse Frobenius to the last code, we get $\text{vect}\left(\boldsymbol{g}+\gamma^{q^{m-2}}\boldsymbol{h}\right)$. Since, from Equation (6.10), we also know $\text{vect}\left(\boldsymbol{g}+\gamma\boldsymbol{h}\right)$, one can compute

$$ \begin{aligned} \text{vect}\left(\boldsymbol{g}+\gamma\boldsymbol{h}\right) + \text{vect}\left(\boldsymbol{g}^{[1]}+\gamma^{q^{m-1}}\boldsymbol{h}^{[1]}\right)^{[-1]} &= \text{vect}\left(\boldsymbol{g}+\gamma\boldsymbol{h},\ \boldsymbol{g}+\gamma^{q^{m-2}}\boldsymbol{h}\right) \\ &= \text{vect}\left(\boldsymbol{g},\boldsymbol{h}\right). \end{aligned} \tag{6.12} $$

Next, for any $i \in [0,r]$, one can compute

$$ {\mathscr{C}_{\text{pub}}}^{\perp} \cap \text{vect}\left(\boldsymbol{g},\boldsymbol{h}\right)^{[i]} = \text{vect}\left(\boldsymbol{g}^{[i]}+\gamma\boldsymbol{h}^{[i]}\right). $$

By applying the i–th inverse Frobenius to the previous result, we obtain the space generated by $\boldsymbol{g}+\gamma^{q^{-i}}\boldsymbol{h}$ for any $i \in [0,r]$. In summary, we know the spaces

$$ \text{vect}\left(\boldsymbol{g}+\gamma\boldsymbol{h}\right), \qquad \text{vect}\left(\boldsymbol{g}+\gamma^{q^{-1}}\boldsymbol{h}\right), \qquad \ldots, \qquad \text{vect}\left(\boldsymbol{g}+\gamma^{q^{-r}}\boldsymbol{h}\right). $$

In addition, from Lemma 6.1, the vector $\boldsymbol{a}$ and hence the pair $(\boldsymbol{g},\boldsymbol{h})$ is determined up to some multiplicative constant. Therefore, one can choose an arbitrary element of $\text{vect}\left(\boldsymbol{g}+\gamma\boldsymbol{h}\right)$ and suppose that this element is $\boldsymbol{g}+\gamma\boldsymbol{h}$.

6.5.2 Step 2: Finding γ

In summary, the vector $\boldsymbol{g}+\gamma\boldsymbol{h}$ and the spaces $\text{vect}\left(\boldsymbol{g}+\gamma^{q^i}\boldsymbol{h}\right)$ for any $i \in [-1,-r]$ are known. To compute γ, we will use the following lemma.

Lemma 6.9. *For $i,j \in [1,r]$, $i \neq j$, there exists a unique pair*

$$ (\boldsymbol{u}_{ij},\boldsymbol{v}_{ij}) \in \text{vect}\left(\boldsymbol{g}+\gamma^{q^{-i}}\boldsymbol{h}\right) \times \text{vect}\left(\boldsymbol{g}+\gamma^{q^{-j}}\boldsymbol{h}\right) $$

such that $\boldsymbol{u}_{ij}+\boldsymbol{v}_{ij} = \boldsymbol{g}+\gamma\boldsymbol{h}$.

Proof. It suffices to observe that $\text{vect}\left(\boldsymbol{g},\boldsymbol{h}\right) = \text{vect}\left(\boldsymbol{g}+\gamma^{q^{-i}}\boldsymbol{h}\right) + \text{vect}\left(\boldsymbol{g}+\gamma^{q^{-j}}\boldsymbol{h}\right)$ and $\text{vect}\left(\boldsymbol{g}+\gamma^{q^{-i}}\boldsymbol{h}\right) \cap \text{vect}\left(\boldsymbol{g}+\gamma^{q^{-j}}\boldsymbol{h}\right) = \{0\}$. □

The pairs of vectors $(\boldsymbol{u}_{ij},\boldsymbol{v}_{ij})$ can be easily computed. Thus, from now on, we suppose we know them. In addition, despite γ, $\boldsymbol{g}+\gamma^{q^{-i}}\boldsymbol{h}$ and $\boldsymbol{g}+\gamma^{q^{-j}}\boldsymbol{h}$ being unknown, a computation allows us to show that $\boldsymbol{u}_{ij},\boldsymbol{v}_{ij}$ have the following expressions.

$$ \boldsymbol{u}_{ij} = \frac{\gamma^{q^{-j}}-\gamma}{\gamma^{q^{-j}}-\gamma^{q^{-i}}} \cdot (\boldsymbol{g}+\gamma^{q^{-i}}\boldsymbol{h}) \qquad \text{and} \qquad \boldsymbol{v}_{ij} = \frac{\gamma-\gamma^{q^{-i}}}{\gamma^{q^{-j}}-\gamma^{q^{-i}}} \cdot (\boldsymbol{g}+\gamma^{q^{-j}}\boldsymbol{h}). \tag{6.13} $$

Consider the vectors $\boldsymbol{u}_{12}$ and $\boldsymbol{u}_{13}$. They are collinear since, from Equation (6.13), they are both multiples of $\boldsymbol{g} + \gamma^{q^{-1}}\boldsymbol{h}$. Therefore, one can compute the scalar α such that $\boldsymbol{u}_{12} = \alpha \cdot \boldsymbol{u}_{13}$. From Equation (6.13) we deduce that γ satisfies the following relation.

$$\frac{\gamma^{q^{-2}} - \gamma}{\gamma^{q^{-2}} - \gamma^{q^{-1}}} = \alpha \cdot \frac{\gamma^{q^{-3}} - \gamma}{\gamma^{q^{-3}} - \gamma^{q^{-1}}}. \tag{6.14}$$

Or equivalently, γ is a root of the polynomial

$$Q_\gamma(X) \stackrel{\text{def}}{=} (X^q - X^{q^3})(X - X^{q^2}) - \alpha^{q^3}(X - X^{q^3})(X^q - X^{q^2}).$$

Moreover $(X^q - X)^{q+1}$ divides Q_γ. Indeed, $X^q - X$ divides $X^{q^2} - X$ and $X^{q^3} - X$ then $(X^q - X)^q$ divides $X^{q^3} - X^q$ and $X^{q^2} - X^q$.

We now set

$$P_\gamma(X) \stackrel{\text{def}}{=} \frac{Q_\gamma}{(X^q - X)^{q+1}}.$$

Since the element γ we look for is not in $\mathbb{F}_q$, it is not a root of $X^q - X$ and hence is a root of P_γ. Next, the forthcoming Proposition 6.4 provides the description of the other roots. We first need a technical lemma.

Lemma 6.10. *Let $a, b, c, d \in \mathbb{F}_q$, let i, j be two non-negative integers and set $A(X) = X^{q^i} - X^{q^j} \in \mathbb{F}_q[X]$. Then,*

$$A\left(\frac{aX + b}{cX + d}\right) = \frac{ad - bc}{(cX + d)^{q^i + q^j}} \cdot A(X).$$

Proposition 6.4. *The set of roots of P_γ equals the orbit of γ under the action of $\mathbf{PGL}(2, \mathbb{F}_q)$. Equivalently, any root of P_γ is of the form $\frac{a\gamma+b}{c\gamma+d}$ for $a, b, c, d \in \mathbb{F}_q$ such that $ad - bc \neq 0$.*

Proof. First, notice that $\deg Q_\gamma = q^3 + q^2$ and hence

$$\deg P_\gamma = \deg Q_\gamma - q(q+1) = q^3 - q = |\mathbf{PGL}(2, \mathbb{F}_q)|.$$

Second, for any $\begin{pmatrix} a & b \\ c & d \end{pmatrix} \in \mathbf{PGL}(2, \mathbb{F}_q)$, Lemma 6.10 entails

$$P_\gamma\left(\frac{aX + b}{cX + d}\right) = \frac{1}{(cX + d)^{q^3 - q}} \cdot P_\gamma(X).$$

Since $\gamma \in \mathbb{F}_{q^m} \setminus \mathbb{F}_q$, then $c\gamma + d \neq 0$ and hence, $P_\gamma\left(\frac{a\gamma+b}{c\gamma+d}\right) = 0$.

We proved that any element in the orbit of γ under $\mathbf{PGL}(2, \mathbb{F}_q)$ is a root of P_γ. To conclude, we need to prove that the orbit of γ under $\mathbf{PGL}(2, \mathbb{F}_q)$ has cardinality $\deg P_\gamma = q^3 - q$ which means that the stabiliser of γ with respect to this group action is trivial. Indeed, suppose that

$$\gamma = \frac{a\gamma + b}{c\gamma + d}, \quad \text{for some} \quad \begin{pmatrix} a & b \\ c & d \end{pmatrix} \in \mathbf{PGL}(2, \mathbb{F}_q) \setminus \left\{ \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \right\}.$$

Then γ is a root of the polynomial

$$X(cX + d) - (aX + b) = cX^2 + (d - a)X + b \in \mathbb{F}_q[X].$$

This polynomial is nonzero. Indeed, if it was, we would have $b = c = 0$ and $a = d$ which means

$$\begin{pmatrix} a & b \\ c & d \end{pmatrix} \sim \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \quad \text{in} \quad \mathbf{PGL}\left(2, \mathbb{F}_q\right).$$

Next, this nonzero polynomial cancelling γ has degree at most 2, while from Assumption (2) the minimal polynomial of γ over $\mathbb{F}_q$ has degree $m > 2$. □

Thanks to Propositions 6.3 and 6.4, we deduce that choosing an arbitrary root γ' of P_γ provides a candidate for γ and there remains to compute $\boldsymbol{g}', \boldsymbol{h}'$ providing our triple. Since $\gamma = \frac{a\gamma'+b}{c\gamma'+d}$ for some $a, b, c, d \in \mathbb{F}_q$, then, $\boldsymbol{g} + \gamma\boldsymbol{h} = \boldsymbol{g}' + \gamma'\boldsymbol{h}'$ where

$$\boldsymbol{g}' \stackrel{\text{def}}{=} \frac{1}{(c\gamma' + d)} \cdot (d\boldsymbol{g} + b\boldsymbol{h}) \qquad \text{and} \qquad \boldsymbol{h}' \stackrel{\text{def}}{=} \frac{1}{(c\gamma' + d)} \cdot (c\boldsymbol{g} + a\boldsymbol{h}).$$

Considering Equation (6.13) and using Lemma 6.10, we get

$$\boldsymbol{u}_{12} = \frac{\gamma'^{q^{-2}} - \gamma'}{\gamma'^{q^{-2}} - \gamma'^{q^{-1}}} \cdot (\boldsymbol{g}' + \gamma'^{q^{-1}}\boldsymbol{h}').$$

Consequently, we know γ' and the vectors $\boldsymbol{g}' + \gamma'\boldsymbol{h}' = \boldsymbol{g} + \gamma\boldsymbol{h}$ and $\boldsymbol{u}_{12}$. Thus, we can also compute

$$\boldsymbol{g}' + \gamma'^{q^{-1}}\boldsymbol{h}' = \frac{\gamma'^{q^{-2}} - \gamma'^{q^{-1}}}{\gamma'^{-2} - \gamma'}\boldsymbol{u}_{12}.$$

Knowing γ', $\boldsymbol{g}' + \gamma'\boldsymbol{h}'$ and $\boldsymbol{g}' + \gamma'^{q^{-1}}\boldsymbol{h}'$ allows us to recover $(\boldsymbol{g}', \boldsymbol{h}')$.

6.5.3 End of the attack

Choose an arbitrary support vector $\boldsymbol{a}' \in \mathbb{F}_{q^m}^n$ of rank n. Let $\boldsymbol{Q}_0$ (resp. $\boldsymbol{Q}_1$) be the unique $n \times n$ matrix with entries in $\mathbb{F}_q$ such that $\boldsymbol{a}'\boldsymbol{Q}_0 = \boldsymbol{g}'$ (resp. $\boldsymbol{a}'\boldsymbol{Q}_1 = \boldsymbol{h}'$). Then, by setting $\boldsymbol{Q} \stackrel{\text{def}}{=} \boldsymbol{Q}_0^T + \gamma'\boldsymbol{Q}_1^T$, we have

$$\mathscr{C}_{\text{pub}}{}^{\perp} = \mathcal{G}_{n-k}\left(\boldsymbol{a}'\right) \cdot \boldsymbol{Q}^T$$

and the matrix $\boldsymbol{Q}$ is nonsingular. Indeed, it sends the full rank vector $\boldsymbol{a}'$ onto $\boldsymbol{g}'+\gamma'\boldsymbol{h}' = \boldsymbol{g}+\gamma\boldsymbol{h} = \boldsymbol{a}\boldsymbol{P}$. Therefore, using Lemma 6.6 we get another representation of the public code as

$$\mathscr{C}_{\text{pub}} = \mathcal{G}_k\left(\boldsymbol{b}'\right) \cdot \boldsymbol{Q}^{-1},$$

where $\boldsymbol{b}' \in \mathbb{F}_{q^m}^n$ is so that $\mathcal{G}_k\left(\boldsymbol{b}'\right) = \mathcal{G}_{n-k}\left(\boldsymbol{a}'\right)^{\perp}$. Then, any ciphertext is of the form $\boldsymbol{c} + \boldsymbol{e}$ with $\boldsymbol{c} \in \mathscr{C}_{\text{pub}}$ and $\boldsymbol{e} \in \mathbb{F}_{q^m}^n$ of rank less than or equal to $\frac{n-k}{4}$. Then $\boldsymbol{e}\boldsymbol{Q}$ has rank weight at most $\frac{n-k}{2}$ and hence, the vector $(\boldsymbol{c}+\boldsymbol{e})\boldsymbol{Q}$ can be decoded as a corrupted codeword of $\mathcal{G}_k\left(\boldsymbol{b}'\right)$ in order to recover $\boldsymbol{e}$ and deduce the plaintext.

6.5.4 Complexity of the attack

Let us conclude with a short complexity analysis of the attack. Let ω be the exponent of the complexity of linear algebra operations. Additions, multiplications will be considered as elementary operations in $\mathbb{F}_{q^m}$ that we will count. The evaluation of the Frobenius map costs $O(\log q)$ operations.

Table 6.1 – Timings for our implementation. For any parameter, the attack has been run 100 times and the last column gives the average timing.

q	m	n	k	Average time
2	50	50	32	0.6 s
2	80	70	41	1.2 s
2	120	110	65	9.5 s

Step 1. The computation of the dual public code ${\mathscr{C}_{\text{pub}}}^{\perp}$ costs $O(n^\omega)$ operations. Next, the computation of ${\mathscr{C}_{\text{pub}}}^{\perp^{[1]}}$ costs $O\left(n^2 \log q\right)$ operations in $\mathbb{F}_{q^m}$ and the iterative computation of ${\mathscr{C}_{\text{pub}}}^{\perp^{[i]}}$ for $i \in [1, n-k+1]$ costs $O\left(n^3 \log q\right)$ operations in $\mathbb{F}_{q^m}$. The computation of ${\mathscr{C}_{\text{pub}}}^{\perp} + {\mathscr{C}_{\text{pub}}}^{\perp^{[1]}}$ boils down to a Gaussian elimination and hence costs $O(n^\omega)$ operations in $\mathbb{F}_{q^m}$. Since we compute $O(n)$ intersections of spaces, the overall cost of the computation of

$$\left({\mathscr{C}_{\text{pub}}}^{\perp} + {\mathscr{C}_{\text{pub}}}^{\perp^{[1]}}\right) \cap \left({\mathscr{C}_{\text{pub}}}^{\perp^{[1]}} + {\mathscr{C}_{\text{pub}}}^{\perp^{[2]}}\right) \cap \cdots \cap \left({\mathscr{C}_{\text{pub}}}^{\perp^{[r]}} + {\mathscr{C}_{\text{pub}}}^{\perp^{[r+1]}}\right),$$

is of $O\left(n^{\omega+1}\right)$ operations in $\mathbb{F}_{q^m}$. As a conclusion, the overall cost of the first step is $O\left(n^3 \log q + n^{\omega+1}\right)$.

Step 2. The computation of a pair $(\boldsymbol{u}_{ij}, \boldsymbol{v}_{ij})$ represents the resolution of a linear system with 2 unknowns and n equations, which costs $O(n)$ operations. This computation should be performed $O(n)$ times, yielding an overall cost of $O\left(n^2\right)$ operations in $\mathbb{F}_{q^m}$, which is negligible compared to the previous step.

Next, the computation of a root of P_γ can be computed using the Cantor–Zassenhaus algorithm whose complexity is in $\widetilde{O}\Big(\left(\deg P_\gamma\right)^2 m \log q\Big)$ operations in $\mathbb{F}_{q^m}$ (see for instance [Bos+17, Théorème 19.20]), where $\widetilde{O}(\cdot)$ means that the factors in $\log\left(\deg P_\gamma\right)$ are neglected. Furthermore, since $\deg P_\gamma = q^3 - q$ we get a complexity of $\widetilde{O}\left(mq^6\right)$ for the calculation of γ'.

The remainder of the attack consists in a finite number of linear systems solving, i.e. a cost of $O(n^\omega)$, which is negligible compared to Step 1.

Summary. This yields an overall cost of $O\left(n^3 \log q + n^{\omega+1}\right) + \widetilde{O}\left(mq^6\right)$. Classically, one chooses q small and n close to m, for instance $q = 2$ and $m = O(n)$. In this situation, we get an overall cost of $O\left(n^{\omega+1}\right)$.

6.5.5 Implementation

The attack has been implemented using `Magma` and permits to recover a 4–tuple

$$(\boldsymbol{a}', \gamma'\boldsymbol{Q}_0, \boldsymbol{Q}_1) \text{ such that } {\mathscr{C}_{\text{pub}}}^{\perp} = \mathcal{G}_{n-k}\left(\boldsymbol{a}'\right) \cdot \left(\boldsymbol{Q}_0 + \gamma'\boldsymbol{Q}_1\right)^T.$$

The attack ran on a personal machine[1] and succeeded in a few seconds for parameters of cryptographic size as illustrated by Table 6.1. Our implementation is only a proof of concept and could be significantly optimized.

[1] Processor: Intel® Core™ i5-8250U CPU @ 1.60GHz.

6.6 Conclusion

We provided a distinguisher *à la Overbeck* for the public keys of Loidreau's scheme when $\lambda = 2$ and the public code has rate $R_{\text{pub}} \geqslant \frac{1}{2}$. From this distinguisher, we are able to derive a polynomial-time key-recovery attack. For small values of q, the attack runs in $O(n^{\omega+1})$ operations in $\mathbb{F}_{q^m}$ and a `Magma` implementation succeeds to recover the hidden structure of the public key in a few seconds.

For larger values of λ, the distinguisher holds when the dimension of the code satisfies

$$n\left(1 - \frac{1}{\lambda}\right) + 1 < k < n - \lambda \tag{6.15}$$

but generalizing the key recovery should require more work. Indeed, the fact that the distinguisher holds should allow to easily generalize the first step (Section 6.5.1) to larger values of λ but generalizing the second step (Section 6.5.2) should require a harder analysis with multivariate polynomials. Besides, the parameters λ, k, n proposed by Loidreau in [Loi17] avoid the weak case of Equation (6.15) anyway. Here, an analogy with Hamming metric and the original McEliece encryption scheme can be drawn. Indeed, similarly to Loidreau's public keys, high-rate alternant codes have been proved to be distinguishable from random codes in polynomial time [Fau+10].

Conclusions

In an effort to gain some measure of hindsight over my contributions, the first chapter of this document explained the context and the relevance of cryptanalysis in general, for both symmetric and asymmetric primitives. In particular, it shared the view of cryptographers on computational security and it drew a parallel between studying symmetric-key primitives like block-ciphers and studying public-key primitives like one-way trapdoor functions. Indeed, even if this thesis was mostly dedicated to the cryptanalysis of symmetric-key primitives, I dabbled in the public-key setting with an analysis of a rank-metric code-based encryption scheme in Chapter 6.

My thesis started with the analysis of recent distinguishers on the most used block-cipher, the AES. This work showed that these distinguishers were not likely to be improved, which is a rather unfavourable perspective for cryptanalysis enthusiasts. However, we provided a valuable tool for designers: a systematic method for testing the resistance of a general SPN cipher against these distinguishers.

Providing valuable tools would become the heart of another contribution on MILP modelings. Block-cipher design and cryptanalysis, especially with the quest for lightweight primitives, seem to be more and more subject to computational power and programming skills. In this context, MILP modelings allow cryptographers to enhance their cryptanalysis capabilities without increasing their programming effort unreasonably. Therefore, the modeling methods presented in Chapter 4 are more a contribution to the cryptographer's workflow than a contribution to cryptanalysis itself. Naturally, our modeling methods may be improved but, as we saw with the MILP-aided search for impossible differentials, the algorithmic thinking about how and when to use MILP models will retain the most significant part of the cryptographer's work.

Independently, I tried to approach algebraic attacks creatively to overcome the limitations of the Möbius transform over $\mathbb{F}_2$. This attempt eventually reached the concept of $\mathbb{Z}$-ANF, and an elegant algorithm to compute sparse ANFs when an evaluation circuit is known. However, the prior detection of possibly-sparse superpolys with the Möbius transform proved to be a non-trivial task, which kept me from exhibiting any valuable application of $\mathbb{Z}$-ANFs. Either way, the development of monomial trails for ANF analysis, to which Chapter 5 aims at contributing with a consistent presentation, may well surpass the $\mathbb{Z}$-ANF. Notably, enumerating monomial trails in semi-evaluated circuits might revive property-testing-based cube attacks.

Bibliography

[Abd+17] Ahmed Abdelkhalek, Yu Sasaki, Yosuke Todo, Mohamed Tolba, and Amr M. Youssef. "MILP Modeling for (Large) S-boxes to Optimize Probability of Differential Characteristics". In: *IACR Trans. Symm. Cryptol.* 2017.4 (2017), pp. 99–129. ISSN: 2519-173X (cit. on pp. 15, 70, 99, 101, 103, 110, 116, 130).

[Alo+03] Noga Alon, Tali Kaufman, Michael Krivelevich, Simon Litsyn, and Dana Ron. "Testing Low-Degree Polynomials over GF(2)". In: *Approximation, Randomization, and Combinatorial Optimization.. Algorithms and Techniques*. Ed. by Sanjeev Arora, Klaus Jansen, José D. P. Rolim, and Amit Sahai. Berlin, Heidelberg: Springer Berlin Heidelberg, 2003, pp. 188–199 (cit. on pp. 141, 142).

[AR99] Yonatan Aumann and Michael O. Rabin. "Information Theoretically Secure Communication in the Limited Storage Space Model". In: *CRYPTO'99*. Ed. by Michael J. Wiener. Vol. 1666. LNCS. Springer, Heidelberg, Aug. 1999, pp. 65–79 (cit. on p. 28).

[Aum+09] Jean-Philippe Aumasson, Itai Dinur, Willi Meier, and Adi Shamir. "Cube Testers and Key Recovery Attacks on Reduced-Round MD6 and Trivium". In: *FSE 2009*. Ed. by Orr Dunkelman. Vol. 5665. LNCS. Springer, Heidelberg, Feb. 2009, pp. 1–22 (cit. on pp. 17, 132, 141–143).

[BA08] Behnam Bahrak and Mohammad Reza Aref. "Impossible differential attack on seven-round AES-128". In: *IET Inf. Secur.* 2.2 (2008), pp. 28–32 (cit. on pp. 71, 73).

[Bal+16] Marco Baldi, Marco Bianchi, Franco Chiaraluce, Joachim Rosenthal, and Davide Schipani. "Enhanced Public Key Security for the McEliece Cryptosystem". In: *J. Cryptology* 29.1 (2016), pp. 1–27 (cit. on p. 167).

[Ban+15] Subhadeep Banik, Andrey Bogdanov, Takanori Isobe, Kyoji Shibutani, Harunaga Hiwatari, Toru Akishita, and Francesco Regazzoni. "Midori: A Block Cipher for Low Energy". In: *ASIACRYPT 2015, Part II*. Ed. by Tetsu Iwata and Jung Hee Cheon. Vol. 9453. LNCS. Springer, Heidelberg, 2015, pp. 411–436 (cit. on pp. 14, 74, 88, 89).

[BB06] Elad Barkan and Eli Biham. "Conditional Estimators: An Effective Attack on A5/1". In: *SAC 2005*. Ed. by Bart Preneel and Stafford Tavares. Vol. 3897. LNCS. Springer, Heidelberg, Aug. 2006, pp. 1–19 (cit. on p. 36).

[BBS99] Eli Biham, Alex Biryukov, and Adi Shamir. "Cryptanalysis of Skipjack Reduced to 31 Rounds Using Impossible Differentials". In: *EUROCRYPT'99*. Ed. by Jacques Stern. Vol. 1592. LNCS. Springer, Heidelberg, May 1999, pp. 12–23 (cit. on pp. 13, 71).

[BC16] Christina Boura and Anne Canteaut. "Another View of the Division Property". In: *CRYPTO 2016, Part I*. Ed. by Matthew Robshaw and Jonathan Katz. Vol. 9814. LNCS. Springer, Heidelberg, Aug. 2016, pp. 654–682 (cit. on pp. 145, 150).

[BC20] Christina Boura and Daniel Coggia. "Efficient MILP Modelings for Sboxes and Linear Layers of SPN ciphers". In: *IACR Transactions on Symmetric Cryptology* 2020.3 (2020), pp. 327–361 (cit. on pp. 14, 95).

[BCC19] Christina Boura, Anne Canteaut, and Daniel Coggia. "A General Proof Framework for Recent AES Distinguishers". In: *IACR Trans. Symm. Cryptol.* 2019.1 (2019), pp. 170–191. ISSN: 2519-173X (cit. on pp. 13, 73).

[BCP97] Wieb Bosma, John Cannon, and Catherine Playoust. "The Magma Algebra System I: The User Language". In: *J. Symbolic Comput.* 24.3/4 (1997), pp. 235–265 (cit. on p. 177).

[BD00] Eli Biham and Orr Dunkelman. "Cryptanalysis of the A5/1 GSM Stream Cipher". In: *INDOCRYPT 2000*. Ed. by Bimal K. Roy and Eiji Okamoto. Vol. 1977. LNCS. Springer, Heidelberg, Dec. 2000, pp. 43–51 (cit. on p. 36).

[Bei+16] Christof Beierle, Jérémy Jean, Stefan Kölbl, Gregor Leander, Amir Moradi, Thomas Peyrin, Yu Sasaki, Pascal Sasdrich, and Siang Meng Sim. "The SKINNY Family of Block Ciphers and Its Low-Latency Variant MANTIS". In: *CRYPTO 2016, Part II*. Ed. by Matthew Robshaw and Jonathan Katz. Vol. 9815. LNCS. Springer, Heidelberg, Aug. 2016, pp. 123–153 (cit. on pp. 14, 70, 74, 88, 91, 99, 121).

[Bel+97] Mihir Bellare, Anand Desai, Eric Jokipii, and Phillip Rogaway. "A Concrete Security Treatment of Symmetric Encryption". In: *38th FOCS*. IEEE Computer Society Press, Oct. 1997, pp. 394–403 (cit. on pp. 34, 47, 48).

[Ber03] Thierry P. Berger. "Isometries for rank distance and permutation group of Gabidulin codes". In: *IEEE Trans. Inform. Theory* 49.11 (2003), pp. 3016–3019 (cit. on p. 169).

[Ber08] Daniel J. Bernstein. "The Salsa20 Family of Stream Ciphers". In: *New Stream Cipher Designs: The eSTREAM Finalists*. Ed. by Matthew Robshaw and Olivier Billet. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, pp. 84–97. ISBN: 978-3-540-68351-3 (cit. on p. 36).

[BGT11] Céline Blondeau, Benoît Gérard, and Jean-Pierre Tillich. "Accurate estimates of the data complexity and success probability for various cryptanalyses". In: *Des. Codes Cryptogr.* 59.1-3 (2011), pp. 3–34 (cit. on p. 64).

[BGW99] Marc Briceno, Ian Goldberg, and David Wagner. *A pedagogical implementation of A5/1*. `https://mtlin.org/article/a51.html`. 1999 (cit. on p. 36).

[Bil+13] Begül Bilgin, Andrey Bogdanov, Miroslav Knežević, Florian Mendel, and Qingju Wang. "Fides: Lightweight Authenticated Cipher with Side-Channel Resistance for Constrained Hardware". In: *CHES 2013*. Ed. by Guido Bertoni and Jean-Sébastien Coron. Vol. 8086. LNCS. Springer, Heidelberg, Aug. 2013, pp. 142–158 (cit. on p. 119).

[BL] Daniel J. Bernstein and Tanja Lange, eds. *eBACS: ECRYPT Benchmarking of Cryptographic Systems*. `https://bench.cr.yp.to` (cit. on p. 49).

[BL16] Karthikeyan Bhargavan and Gaëtan Leurent. "On the Practical (In-)Security of 64-bit Block Ciphers: Collision Attacks on HTTP over TLS and OpenVPN". In: *ACM CCS 2016*. Ed. by Edgar R. Weippl, Stefan Katzenbeisser, Christopher Kruegel, Andrew C. Myers, and Shai Halevi. ACM Press, Oct. 2016, pp. 456–467 (cit. on p. 48).

[Bla06] John Black. "The Ideal-Cipher Model, Revisited: An Uninstantiable Blockcipher-Based Hash Function". In: *FSE 2006*. Ed. by Matthew J. B. Robshaw. Vol. 4047. LNCS. Springer, Heidelberg, Mar. 2006, pp. 328–340 (cit. on p. 30).

[BLR93] Manuel Blum, Michael Luby, and Ronitt Rubinfeld. "Self-testing/correcting with applications to numerical problems". In: *Journal of Computer and System Sciences* 47.3 (1993), pp. 549 –595. ISSN: 0022-0000 (cit. on pp. 139, 142).

[BM17] Elaine Barker and Nicky Mouha. *Recommandation for the Triple Data Encryption Algorithm (TDEA) Block Cipher*. Tech. rep. SP 800-67 Rev.2. National Institute of Standards and Technology (NIST),U.S. Department of Commerce, Nov. 2017 (cit. on p. 42).

[BMT78] Elwyn Berlekamp, Robert McEliece, and Henk van Tilborg. "On the inherent intractability of certain coding problems". In: *IEEE Trans. Inform. Theory* 24.3 (May 1978), pp. 384–386 (cit. on p. 57).

[BN00] Mihir Bellare and Chanathip Namprempre. "Authenticated Encryption: Relations among notions and analysis of the generic composition paradigm". In: *ASIACRYPT 2000*. Ed. by Tatsuaki Okamoto. Vol. 1976. LNCS. Springer, Heidelberg, Dec. 2000, pp. 531–545 (cit. on pp. 53, 54).

[Bog+07] Andrey Bogdanov, Lars R. Knudsen, Gregor Leander, Christof Paar, Axel Poschmann, Matthew J. B. Robshaw, Yannick Seurin, and C. Vikkelsoe. "PRESENT: An Ultra-Lightweight Block Cipher". In: *CHES 2007*. Ed. by Pascal Paillier and Ingrid Verbauwhede. Vol. 4727. LNCS. Springer, Heidelberg, Sept. 2007, pp. 450–466 (cit. on pp. 43, 49).

[Bor+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 Yalçin. "PRINCE - A Low-Latency Block Cipher for Pervasive Computing Applications - Extended Abstract". In: *ASIACRYPT 2012*. Ed. by Xiaoyun Wang and Kazue Sako. Vol. 7658. LNCS. Springer, Heidelberg, Dec. 2012, pp. 208–225 (cit. on pp. 88, 93).

[Bos+17] Alin Bostan, Frédéric Chyzak, Marc Giusti, Romain Lebreton, Grégoire Lecerf, Bruno Salvy, and Éric Schost. *Algorithmes Efficaces en Calcul Formel*. French. 686 pages. Imprimé par CreateSpace. Aussi disponible en version électronique. Palaiseau: Frédéric Chyzak (auto-édit.), Sept. 2017. ISBN: 979-10-699-0947-2 (cit. on p. 182).

[Bou+10] Charles Bouillaguet, Hsieh-Chung Chen, Chen-Mou Cheng, Tung Chou, Ruben Niederhagen, Adi Shamir, and Bo-Yin Yang. "Fast Exhaustive Search for Polynomial Systems in $\mathbb{F}_2$". In: *CHES 2010*. Ed. by Stefan Mangard and François-Xavier Standaert. Vol. 6225. LNCS. Springer, Heidelberg, Aug. 2010, pp. 203–218 (cit. on p. 138).

[Bou+18] Christina Boura, Virginie Lallemand, María Naya-Plasencia, and Valentin Suder. "Making the Impossible Possible". In: *Journal of Cryptology* 31.1 (Jan. 2018), pp. 101–133 (cit. on p. 71).

[Bou+20] Fabrice Boudot, Pierrick Gaudry, Aurore Guillevic, Nadia Heninger, Emmanuel Thomé, and Paul Zimmermann. "Comparing the Difficulty of Factorization and Discrete Logarithm: A 240-Digit Experiment". In: *CRYPTO 2020, Part II*. Ed. by Daniele Micciancio and Thomas Ristenpart. Vol. 12171. LNCS. Springer, Heidelberg, Aug. 2020, pp. 62–91 (cit. on p. 56).

[BR04] Mihir Bellare and Phillip Rogaway. *Code-Based Game-Playing Proofs and the Security of Triple Encryption*. Cryptology ePrint Archive, Report 2004/331. `https://eprint.iacr.org/2004/331`. 2004 (cit. on pp. 46, 52).

[BR93] Mihir Bellare and Phillip Rogaway. "Random Oracles are Practical: A Paradigm for Designing Efficient Protocols". In: *ACM CCS 93*. Ed. by Dorothy E. Denning, Raymond Pyle, Ravi Ganesan, Ravi S. Sandhu, and Victoria Ashby. ACM Press, Nov. 1993, pp. 62–73 (cit. on p. 29).

[BR95] Mihir Bellare and Phillip Rogaway. "Optimal Asymmetric Encryption". In: *EUROCRYPT'94*. Ed. by Alfredo De Santis. Vol. 950. LNCS. Springer, Heidelberg, May 1995, pp. 92–111 (cit. on p. 60).

[Bra+84] Robert King Brayton, Alberto L. Sangiovanni-Vincentelli, Curtis T. Mc-Mullen, and Gary D. Hachtel. "Logic Minimization Algorithms for VLSI Synthesis". In: *Kluwer Academic Publishers* (1984) (cit. on p. 99).

[Bro+10] K.A. Browning, J.F. Dillon, M.T. McQuistan, and A.J. Wolfe. "An APN permutation in dimension six". In: *Finite Fields: theory and applications* 42 (2010), pp. 33–42 (cit. on p. 119).

[BS91] Eli Biham and Adi Shamir. "Differential Cryptanalysis of DES-like Cryptosystems". In: *Journal of Cryptology* 4.1 (Jan. 1991), pp. 3–72 (cit. on pp. 61, 64).

[BSW01] Alex Biryukov, Adi Shamir, and David Wagner. "Real Time Cryptanalysis of A5/1 on a PC". In: *FSE 2000*. Ed. by Bruce Schneier. Vol. 1978. LNCS. Springer, Heidelberg, Apr. 2001, pp. 1–18 (cit. on p. 36).

[BW99] Alex Biryukov and David Wagner. "Slide Attacks". In: *FSE'99*. Ed. by Lars R. Knudsen. Vol. 1636. LNCS. Springer, Heidelberg, Mar. 1999, pp. 245–259 (cit. on p. 41).

[Can+18] Anne Canteaut, Sergiu Carpov, Caroline Fontaine, Tancrède Lepoint, María Naya-Plasencia, Pascal Paillier, and Renaud Sirdey. "Stream Ciphers: A Practical Solution for Efficient Homomorphic-Ciphertext Compression". In: *Journal of Cryptology* 31.3 (July 2018), pp. 885–916 (cit. on p. 36).

[Can+19] Anne Canteaut, Sébastien Duval, Gaëtan Leurent, María Naya-Plasencia, Léo Perrin, Thomas Pornin, and André Schrottenloher. *Saturnin: a suite of lightweight symmetric algorithms for post-quantum security*. Candidate to the Lightweight Cryptography NIST Competition. 2019 (cit. on p. 126).

[CC20] Daniel Coggia and Alain Couvreur. "On the security of a Loidreau rank metric code based encryption scheme". In: *Des. Codes Cryptogr.* 88.9 (2020), pp. 1941–1957 (cit. on pp. 18, 167).

[Cha+12] Shu jen Chang, Ray Perlner, William Burr, Meltem Sönmez Turan, John Kelsey, Souradyuti Paul, and Lawrence Bassham. *Third-Round Report of the SHA-3 Cryptographic Hash Algorithm Competition*. Tech. rep. NISTIR 7896. National Institute of Standards and Technology (NIST),U.S. Department of Commerce, Nov. 2012 (cit. on p. 51).

[CM03] Nicolas Courtois and Willi Meier. "Algebraic Attacks on Stream Ciphers with Linear Feedback". In: *EUROCRYPT 2003*. Ed. by Eli Biham. Vol. 2656. LNCS. Springer, Heidelberg, May 2003, pp. 345–359 (cit. on p. 131).

[Cou+14] Alain Couvreur, Philippe Gaborit, Valérie Gauthier-Umaña, Ayoub Otmani, and Jean-Pierre Tillich. "Distinguisher-based attacks on public-key cryptosystems using Reed-Solomon codes". In: *Des. Codes Cryptogr.* 73.2 (2014), pp. 641–666 (cit. on p. 174).

[Cou+15] Alain Couvreur, Ayoub Otmani, Jean-Pierre Tillich, and Valérie Gauthier-Umaña. "A Polynomial-Time Attack on the BBCRS Scheme". In: *Public-Key Cryptography - PKC 2015*. Ed. by J. Katz. Vol. 9020. LNCS. Springer, 2015, pp. 175–193 (cit. on p. 174).

[CP02] Nicolas Courtois and Josef Pieprzyk. "Cryptanalysis of Block Ciphers with Overdefined Systems of Equations". In: *ASIACRYPT 2002*. Ed. by Yuliang Zheng. Vol. 2501. LNCS. Springer, Heidelberg, Dec. 2002, pp. 267–287 (cit. on p. 131).

[Dae95] Joan Daemen. "Cipher and Hash Function Design. Strategies based on linear and differential cryptanalysis". Theses. Katholieke Universiteit Leuven, 1995 (cit. on p. 67).

[DCP08] Christophe De Cannière and Bart Preneel. "Trivium". In: *New Stream Cipher Designs: The eSTREAM Finalists*. Ed. by Matthew Robshaw and Olivier Billet. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, pp. 244–266. ISBN: 978-3-540-68351-3 (cit. on p. 36).

[Del78] Philippe Delsarte. "Bilinear Forms over a Finite Field, with Applications to Coding Theory". In: *J. Comb. Theory, Ser. A* 25.3 (1978), pp. 226–241 (cit. on p. 167).

[DH77] W. Diffie and M. E. Hellman. "Special Feature Exhaustive Cryptanalysis of the NBS Data Encryption Standard". In: *Computer* 10.6 (1977), pp. 74–84 (cit. on p. 42).

[Dil09] John Dillon. *APN polynomials: An Update*. Invited talk at Fq9, the 9th International Conference on Finite Fields and Applications, Dublin, July 13-17. 2009 (cit. on p. 119).

[Din+11] Itai Dinur, Tim Güneysu, Christof Paar, Adi Shamir, and Ralf Zimmermann. "An Experimentally Verified Attack on Full Grain-128 Using Dedicated Reconfigurable Hardware". In: *ASIACRYPT 2011*. Ed. by Dong Hoon Lee and Xiaoyun Wang. Vol. 7073. LNCS. Springer, Heidelberg, Dec. 2011, pp. 327–343 (cit. on p. 144).

[Din+15] Itai Dinur, Yunwen Liu, Willi Meier, and Qingju Wang. "Optimized Interpolation Attacks on LowMC". In: *ASIACRYPT 2015, Part II*. Ed. by Tetsu Iwata and Jung Hee Cheon. Vol. 9453. LNCS. Springer, Heidelberg, 2015, pp. 535–560 (cit. on p. 132).

[Dob+19] Christoph Dobraunig, Maria Eichlseder, Florian Mendel, and Martin Schläffer. *Ascon v1.2*. Submission to Round 1 of the NIST Lightweight Cryptography project. 2019 (cit. on p. 119).

[Don20] Jason A. Donenfeld. *WireGuard: Next Generation Kernel Network Tunnel*. `https://www.wireguard.com/papers/wireguard.pdf`. 2020 (cit. on p. 51).

[DR01a] Joan Daemen and Vincent Rijmen. *The Design of Rijndael AES The Advanced Encryption Standard*. 1st ed. Information Security and Cryptography. Springer, Berlin, Heidelberg, Nov. 2001. ISBN: 978-3-540-42580-9 (cit. on pp. 42, 67).

[DR01b] Joan Daemen and Vincent Rijmen. "The Wide Trail Design Strategy". In: *8th IMA International Conference on Cryptography and Coding*. Ed. by Bahram Honary. Vol. 2260. LNCS. Springer, Heidelberg, Dec. 2001, pp. 222–238 (cit. on pp. 42, 67).

[DS08] Hüseyin Demirci and Ali Aydin Selçuk. "A Meet-in-the-Middle Attack on 8-Round AES". In: *FSE 2008*. Ed. by Kaisa Nyberg. Vol. 5086. LNCS. Springer, Heidelberg, Feb. 2008, pp. 116–126 (cit. on p. 73).

[DS09] Itai Dinur and Adi Shamir. "Cube Attacks on Tweakable Black Box Polynomials". In: *EUROCRYPT 2009*. Ed. by Antoine Joux. Vol. 5479. LNCS. Springer, Heidelberg, Apr. 2009, pp. 278–299 (cit. on pp. 17, 132, 138, 140, 141).

[DS11] Itai Dinur and Adi Shamir. "Breaking Grain-128 with Dynamic Cube Attacks". In: *FSE 2011*. Ed. by Antoine Joux. Vol. 6733. LNCS. Springer, Heidelberg, Feb. 2011, pp. 167–187 (cit. on pp. 17, 132, 144).

[Dwo01] Morris Dworkin. *Recommandation for Block Cipher Modes of Operation*. Tech. rep. SP 800-38A. National Institute of Standards and Technology (NIST),U.S. Department of Commerce, Dec. 2001 (cit. on p. 46).

[Dwo04] Morris Dworkin. *Recommandation for Block Cipher Modes of Operation: The CCM Mode for Authentication and Confidentiality*. Tech. rep. SP 800-38C. National Institute of Standards and Technology (NIST),U.S. Department of Commerce, May 2004 (cit. on p. 54).

[Dwo05] Morris Dworkin. *Recommandation for Block Cipher Modes of Operation: The CMAC Mode for Authentication*. Tech. rep. SP 800-38B. National Institute of Standards and Technology (NIST),U.S. Department of Commerce, Dec. 2005 (cit. on p. 52).

[Dwo07] Morris Dworkin. *Recommandation for Block Cipher Modes of Operation: The Galois/Counter mode (GCM) and GMAC*. Tech. rep. SP 800-38D. National Institute of Standards and Technology (NIST),U.S. Department of Commerce, Nov. 2007 (cit. on p. 54).

[EJ03] P. Ekdahl and T. Johansson. "Another attack on A5/1". In: *IEEE Transactions on Information Theory* 49.1 (2003), pp. 284–289 (cit. on p. 36).

[Fau+10] Jean-Charles Faugère, Valérie Gauthier, Ayoub Otmani, Ludovic Perret, and Jean-Pierre Tillich. *A Distinguisher for High Rate McEliece Cryptosystems*. IACR Cryptology ePrint Archive, Report2010/331. http://eprint.iacr.org/. 2010 (cit. on p. 183).

[Fau02] Jean Charles Faugère. "A New Efficient Algorithm for Computing Gröbner Bases without Reduction to Zero (F5)". In: *Proceedings of the 2002 International Symposium on Symbolic and Algebraic Computation*. ISSAC '02. Lille, France: Association for Computing Machinery, 2002, 7583. ISBN: 1581134843 (cit. on p. 131).

[Fau99] Jean-Charles Faugére. "A new efficient algorithm for computing Gröbner bases (F4)". In: *Journal of Pure and Applied Algebra* 139.1 (1999), pp. 61–88. ISSN: 0022-4049 (cit. on p. 131).

[Fer+01] Niels Ferguson, John Kelsey, Stefan Lucks, Bruce Schneier, Michael Stay, David Wagner, and Doug Whiting. "Improved Cryptanalysis of Rijndael". In: *FSE 2000*. Ed. by Bruce Schneier. Vol. 1978. LNCS. Springer, Heidelberg, Apr. 2001, pp. 213–230 (cit. on p. 73).

[For06] 3GPP Task Force. *Specification of the 3GPP Confidentiality and Integrity Algorithms UEA2 & UIA2. Document 2: SNOW 3G Specification, version 1.1*. `https://www.gsma.com/aboutus/wp-content/uploads/2014/12/snow3gspec.pdf`. 2006 (cit. on p. 36).

[Fou98] Electronic Frontier Foundation. *Cracking DES: Secrets of Encryption Research, Wiretap Politics & Chip Design*. Electronic Frontier Foundation, 1998. ISBN: 978-1-5659-2520-5 (cit. on p. 42).

[FV14] Pierre-Alain Fouque and Thomas Vannet. "Improving Key Recovery to 784 and 799 Rounds of Trivium Using Optimized Cube Attacks". In: *FSE 2013*. Ed. by Shiho Moriai. Vol. 8424. LNCS. Springer, Heidelberg, Mar. 2014, pp. 502–517 (cit. on p. 141).

[Gab+13] Philippe Gaborit, Gaétan Murat, Olivier Ruatta, and Gilles Zémor. "Low Rank Parity Check codes and their application to cryptography". In: *Proceedings of the Workshop on Coding and Cryptography WCC'2013*. Bergen, Norway, 2013 (cit. on p. 167).

[Gab85] Ernest M. Gabidulin. "Theory of codes with maximum rank distance". In: *Problemy Peredachi Informatsii* 21.1 (1985), pp. 3–16 (cit. on pp. 167, 174).

[GBM02] Jovan Dj. Golic, Vittorio Bagini, and Guglielmo Morgari. "Linear Cryptanalysis of Bluetooth Stream Cipher". In: *EUROCRYPT 2002*. Ed. by Lars R. Knudsen. Vol. 2332. LNCS. Springer, Heidelberg, 2002, pp. 238–255 (cit. on p. 37).

[GM00] Henri Gilbert and Marine Minier. "A Collision Attack on 7 Rounds of Rijndael". In: *3rd AES Candidate Conference*. New York, USA: National Institute of Standards and Technology, 2000, pp. 230–241 (cit. on p. 73).

[GNL12] Zheng Gong, Svetla Nikova, and Yee Wei Law. "KLEIN: A New Family of Lightweight Block Ciphers". In: *RFIDSec 2011*. Ed. by Ari Juels and Christof Paar. Vol. 7055. LNCS. Springer, Heidelberg, June 2012, pp. 1–18 (cit. on pp. 14, 74, 88, 90).

[GO01] Ernst M. Gabidulin and Alexei V. Ourivski. "Modified GPT PKC with Right Scrambler". In: *Electron. Notes Discrete Math.* 6 (2001), pp. 168–177 (cit. on p. 174).

[GO20] LLC Gurobi Optimization. *Gurobi Optimizer Reference Manual*. 2020 (cit. on pp. 97, 121).

[GPT91] Ernst M. Gabidulin, A. V. Paramonov, and O. V. Tretjakov. "Ideals over a non-commutative ring and their applications to cryptography". In: *Advances in Cryptology - EUROCRYPT'91*. LNCS 547. Brighton, Apr. 1991, pp. 482–489 (cit. on p. 167).

[Gra18] Lorenzo Grassi. "Mixture Differential Cryptanalysis: a New Approach to Distinguishers and Attacks on round-reduced AES". In: *IACR Trans. Symm. Cryptol.* 2018.2 (2018), pp. 133–160. ISSN: 2519-173X (cit. on pp. 14, 71, 73, 74, 76, 77, 81, 87).

[GRR16] Lorenzo Grassi, Christian Rechberger, and Sondre Rønjom. "Subspace Trail Cryptanalysis and its Applications to AES". In: *IACR Trans. Symm. Cryptol.* 2016.2 (2016), pp. 192–225. ISSN: 2519-173X (cit. on pp. 71, 74–76, 81).

[GRR17] Lorenzo Grassi, Christian Rechberger, and Sondre Rønjom. "A New Structural-Differential Property of 5-Round AES". In: *EUROCRYPT 2017, Part II*. Ed. by Jean-Sébastien Coron and Jesper Buus Nielsen. Vol. 10211. LNCS. Springer, Heidelberg, 2017, pp. 289–317 (cit. on pp. 13, 71, 73–77, 81).

[Guo+11] Jian Guo, Thomas Peyrin, Axel Poschmann, and Matthew J. B. Robshaw. "The LED Block Cipher". In: *CHES 2011*. Ed. by Bart Preneel and Tsuyoshi Takagi. Vol. 6917. LNCS. Springer, Heidelberg, 2011, pp. 326–341 (cit. on pp. 14, 74, 88, 89).

[Hao+20a] Yonglin Hao, Lin Jiao, Chaoyun Li, Willi Meier, Yosuke Todo, and Qingju Wang. "Links between Division Property and Other Cube Attack Variants". In: *IACR Trans. Symm. Cryptol.* 2020.1 (2020), pp. 363–395. ISSN: 2519-173X (cit. on p. 144).

[Hao+20b] Yonglin Hao, Gregor Leander, Willi Meier, Yosuke Todo, and Qingju Wang. "Modeling for Three-Subset Division Property Without Unknown Subset - Improved Cube Attacks Against Trivium and Grain-128AEAD". In: *EUROCRYPT 2020, Part I*. Ed. by Anne Canteaut and Yuval Ishai. Vol. 12105. LNCS. Springer, Heidelberg, May 2020, pp. 466–495 (cit. on pp. 145, 153, 164).

[Heb+20a] Phil Hebborn, Baptiste Lambin, Gregor Leander, and Yosuke Todo. *Lower Bounds on the Degree of Block Ciphers*. Cryptology ePrint Archive, Report 2020/1051. `https://eprint.iacr.org/2020/1051`. 2020 (cit. on p. 151).

[Heb+20b] Phil Hebborn, Baptiste Lambin, Gregor Leander, and Yosuke Todo. "Lower Bounds on the Degree of Block Ciphers". In: *ASIACRYPT 2020, Part I*. Ed. by Shiho Moriai and Huaxiong Wang. Vol. 12491. LNCS. Springer, Heidelberg, Dec. 2020, pp. 537–566 (cit. on p. 145).

[Hel+08] Martin Hell, Thomas Johansson, Alexander Maximov, and Willi Meier. "The Grain Family of Stream Ciphers". In: *New Stream Cipher Designs: The eSTREAM Finalists*. Ed. by Matthew Robshaw and Olivier Billet. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, pp. 179–190. ISBN: 978-3-540-68351-3 (cit. on p. 37).

[Hoe62] Wassily Hoeffding. *Probability inequalities for sums of bounded random variables*. Tech. rep. Chapel Hill, N. C.: University of North Carolina, May 1962 (cit. on p. 62).

[Hu+20] Kai Hu, Siwei Sun, Meiqin Wang, and Qingju Wang. "An Algebraic Formulation of the Division Property: Revisiting Degree Evaluations, Cube Attacks, and Key-Independent Sums". In: *ASIACRYPT 2020, Part I*. Ed. by Shiho Moriai and Huaxiong Wang. Vol. 12491. LNCS. Springer, Heidelberg, Dec. 2020, pp. 446–476 (cit. on pp. 132, 145).

[IK03] Tetsu Iwata and Kaoru Kurosawa. "OMAC: One-Key CBC MAC". In: *FSE 2003*. Ed. by Thomas Johansson. Vol. 2887. LNCS. Springer, Heidelberg, Feb. 2003, pp. 129–153 (cit. on p. 53).

[Jea13] Jérémy Jean. "Cryptanalysis of Symmetric-Key Primitives Based on the AES Block Cipher". PhD thesis. Ecole Normale Supérieure de Paris - ENS Paris, Sept. 2013 (cit. on pp. 13, 73).

[JK97] Thomas Jakobsen and Lars R. Knudsen. "The Interpolation Attack on Block Ciphers". In: *FSE'97*. Ed. by Eli Biham. Vol. 1267. LNCS. Springer, Heidelberg, Jan. 1997, pp. 28–40 (cit. on p. 132).

[Jou09] Antoine Joux. *Algorithmic Cryptanalysis*. Chapman & Hall/CRC, 2009. ISBN: 978-1-4200-7002-6 (cit. on pp. 48, 50, 135).

[Kar84] N. Karmarkar. "A new polynomial-time algorithm for linear programming". en. In: *Combinatorica* 4.4 (Dec. 1984), pp. 373–395. ISSN: 1439-6912 (cit. on p. 97).

[Ker83] Auguste Kerckhoffs. "La cryptographie militaire". In: *Journal des sciences militaires* (1883) (cit. on p. 25).

[Kim+03] Jongsung Kim, Seokhie Hong, Jaechul Sung, Changhoon Lee, and Sangjin Lee. "Impossible Differential Cryptanalysis for Block Cipher Structures". In: *INDOCRYPT 2003*. Ed. by Thomas Johansson and Subhamoy Maitra. Vol. 2904. LNCS. Springer, Heidelberg, Dec. 2003, pp. 82–96 (cit. on p. 128).

[Knu+10] Lars R. Knudsen, Gregor Leander, Axel Poschmann, and Matthew J. B. Robshaw. "PRINTcipher: A Block Cipher for IC-Printing". In: *CHES 2010*. Ed. by Stefan Mangard and François-Xavier Standaert. Vol. 6225. LNCS. Springer, Heidelberg, Aug. 2010, pp. 16–32 (cit. on p. 101).

[Knu95] Lars R. Knudsen. "Truncated and Higher Order Differentials". In: *FSE'94*. Ed. by Bart Preneel. Vol. 1008. LNCS. Springer, Heidelberg, Dec. 1995, pp. 196–211 (cit. on pp. 13, 71, 131).

[Knu98] Lars Knudsen. *DEAL - A 128-bit Block Cipher*. en. Tech. rep. 151. Norway: Department of Informatics,University of Bergen, Feb. 1998 (cit. on p. 71).

[Kwo+04] Daesung Kwon, Jaesung Kim, Sangwoo Park, Soo Hak Sung, Yaekwon Sohn, Jung Hwan Song, Yongjin Yeom, E-Joong Yoon, Sangjin Lee, Jaewon Lee, Seongtaek Chee, Daewan Han, and Jin Hong. "New Block Cipher: ARIA". In: *ICISC 03*. Ed. by Jong In Lim and Dong Hoon Lee. Vol. 2971. LNCS. Springer, Heidelberg, Nov. 2004, pp. 432–445 (cit. on p. 126).

[Lai94] Xuejia Lai. "Higher Order Derivatives and Differential Cryptanalysis". In: *Communications and Cryptography: Two Sides of One Tapestry*. Ed. by Richard E. Blahut, Daniel J. Costello, Ueli Maurer, and Thomas Mittelholzer. Boston, MA: Springer US, 1994, pp. 227–233. ISBN: 978-1-4615-2694-0 (cit. on p. 131).

[Lim98] Chae Hoon Lim. *CRYPTON: a New 128-bit Block Cipher*. AES Proposal. 1998 (cit. on p. 92).

[Lim99] Chae Hoon Lim. "A Revised Version of Crypton - Crypton V1.0". In: *FSE'99*. Ed. by Lars R. Knudsen. Vol. 1636. LNCS. Springer, Heidelberg, Mar. 1999, pp. 31–45 (cit. on p. 88).

[LMM91] Xuejia Lai, James L. Massey, and Sean Murphy. "Markov Ciphers and Differential Cryptanalysis". In: *EUROCRYPT'91*. Ed. by Donald W. Davies. Vol. 547. LNCS. Springer, Heidelberg, Apr. 1991, pp. 17–38 (cit. on p. 66).

[Loi06] Pierre Loidreau. "A Welch–Berlekamp Like Algorithm for Decoding Gabidulin Codes". In: *Coding and Cryptography*. Ed. by Øyvind Ytrehus. Berlin, Heidelberg: Springer Berlin Heidelberg, 2006, pp. 36–45 (cit. on p. 168).

[Loi17] Pierre Loidreau. "A new rank metric codes based encryption scheme". In: *Post-Quantum Cryptography 2017*. Vol. 10346. LNCS. Springer, 2017, pp. 3–17 (cit. on pp. 18, 167, 174, 183).

[LP20] Gaëtan Leurent and Thomas Peyrin. "SHA-1 is a Shambles: First Chosen-Prefix Collision on SHA-1 and Application to the PGP Web of Trust". In: *USENIX Security 2020*. Ed. by Srdjan Capkun and Franziska Roesner. USENIX Association, Aug. 2020, pp. 1839–1856 (cit. on p. 51).

[LS18] Gaëtan Leurent and Ferdinand Sibleyras. "The Missing Difference Problem, and Its Applications to Counter Mode Encryption". In: *EUROCRYPT 2018, Part II*. Ed. by Jesper Buus Nielsen and Vincent Rijmen. Vol. 10821. LNCS. Springer, Heidelberg, 2018, pp. 745–770 (cit. on p. 49).

[LTW18] Gregor Leander, Cihangir Tezcan, and Friedrich Wiemer. "Searching for Subspace Trails and Truncated Differentials". In: *IACR Trans. Symm. Cryptol.* 2018.1 (2018), pp. 74–100. ISSN: 2519-173X (cit. on pp. 14, 74, 88, 89, 93).

[Lu+08] Jiqiang Lu, Orr Dunkelman, Nathan Keller, and Jongsung Kim. "New Impossible Differential Attacks on AES". In: *INDOCRYPT 2008*. Ed. by Dipanwita Roy Chowdhury, Vincent Rijmen, and Abhijit Das. Vol. 5365. LNCS. Springer, Heidelberg, Dec. 2008, pp. 279–293 (cit. on p. 71).

[Luo+14] Yiyuan Luo, Xuejia Lai, Zhongming Wu, and Guang Gong. "A unified method for finding impossible differentials of block cipher structures". In: *Inf. Sci.* 263 (2014), pp. 211–220 (cit. on p. 128).

[LV04] Yi Lu and Serge Vaudenay. "Cryptanalysis of Bluetooth Keystream Generator Two-Level E0". In: *ASIACRYPT 2004*. Ed. by Pil Joong Lee. Vol. 3329. LNCS. Springer, Heidelberg, Dec. 2004, pp. 483–499 (cit. on p. 37).

[Mai21] Patrick Maigron. *Regional Internet Registries Statistics*. `https://www-public.imtbs-tsp.eu/~maigron/RIR_Stats/RIR_Delegations/World/ASN-ByNb.html`. 2021 (cit. on p. 22).

[Mal+10] Hamid Mala, Mohammad Dakhilalian, Vincent Rijmen, and Mahmoud Modarres-Hashemi. "Improved Impossible Differential Cryptanalysis of 7-Round AES-128". In: *INDOCRYPT 2010*. Ed. by Guang Gong and Kishan Chand Gupta. Vol. 6498. LNCS. Springer, Heidelberg, Dec. 2010, pp. 282–291 (cit. on p. 71).

[McC56] Edward J. McCluskey. "Minimization of Boolean functions". In: *The Bell SystemTechnical Journal* 35.6 (1956), 1417–1444 (cit. on p. 99).

[McE78] Robert J. McEliece. "A Public-Key System Based on Algebraic Coding Theory". In: DSN Progress Report 44. Jet Propulsion Lab, 1978, pp. 114–116 (cit. on pp. 57, 58, 167).

[MJB04] Alexander Maximov, Thomas Johansson, and Steve Babbage. "An Improved Correlation Attack on A5/1". In: *SAC 2004*. Ed. by Helena Handschuh and Anwar Hasan. Vol. 3357. LNCS. Springer, Heidelberg, Aug. 2004, pp. 1–18 (cit. on p. 36).

[Mor+16] Kathleen Moriarty, Burt Kaliski, Jakob Jonsson, and Andreas Rusch. *PKCS #1: RSA Cryptography Specifications Version 2.2*. RFC 8017. Nov. 2016 (cit. on pp. 30, 60).

[Mou+11] Nicky Mouha, Qingju Wang, Dawu Gu, and Bart Preneel. "Differential and Linear Cryptanalysis Using Mixed-Integer Linear Programming". In: *Inscrypt 2011*. Vol. 7537. Lecture Notes in Computer Science. Springer, 2011, pp. 57–76 (cit. on pp. 15, 70, 95, 98, 123).

[MS11] Piotr Mroczkowski and Janusz Szmidt. *Corrigendum to: The Cube Attack on Stream Cipher Trivium and Quadraticity Tests*. Cryptology ePrint Archive, Report 2011/032. `https://eprint.iacr.org/2011/032`. 2011 (cit. on p. 141).

[Nisa] *Advanced Encryption Standard (AES)*. Tech. rep. FIPS PUB 197. National Institute of Standards and Technology (NIST),U.S. Department of Commerce, Nov. 2001 (cit. on pp. 12, 42, 43, 45).

[Nisb] *Data Encryption Standard (DES)*. Tech. rep. FIPS PUB 46-3. National Institute of Standards and Technology (NIST),U.S. Department of Commerce, Oct. 1999 (cit. on p. 42).

[Nisc] *Digital Signature Standard (DSS)*. Tech. rep. FIPS PUB 186-4. National Institute of Standards and Technology (NIST),U.S. Department of Commerce, July 2013 (cit. on p. 30).

[Nisd] *Secure Hash Standards (SHS)*. Tech. rep. FIPS PUB 180-4. National Institute of Standards and Technology (NIST),U.S. Department of Commerce, Aug. 2015 (cit. on pp. 50, 51).

[Nise] *SHA-3 Standard: Permutation-Based Hash and Extendable-Output Functions*. Tech. rep. FIPS PUB 202. National Institute of Standards and Technology (NIST),U.S. Department of Commerce, Aug. 2015 (cit. on pp. 51, 119).

[Nisf] *Transitions: Recommendation for Transitioning the Use of Cryptographic Algorithms and Key Lengths*. Tech. rep. SP 800-131A. National Institute of Standards and Technology (NIST),U.S. Department of Commerce, Jan. 2011 (cit. on p. 50).

[NL18] Yoav Nir and Adam Langley. *ChaCha20 and Poly1305 for IETF Protocols*. RFC 8439. June 2018 (cit. on p. 36).

[Nyb94] Kaisa Nyberg. "Differentially Uniform Mappings for Cryptography". In: *EUROCRYPT'93*. Ed. by Tor Helleseth. Vol. 765. LNCS. Springer, Heidelberg, May 1994, pp. 55–64 (cit. on p. 65).

[Ove08] Raphael Overbeck. "Structural Attacks for Public Key Cryptosystems based on Gabidulin Codes". In: *J. Cryptology* 21.2 (2008), pp. 280–301 (cit. on pp. 19, 169, 174).

[Per18] Trevor Perrin. *The Noise Protocol Framework*. `https://noiseprotocol.org/noise.html`. 2018 (cit. on p. 51).

[Poi05] David Pointcheval. "Provable Security for Public Key Schemes". In: *Contemporary Cryptology*. Basel: Birkhäuser Basel, 2005, pp. 133–190. ISBN: 978-3-7643-7394-8 (cit. on p. 27).

[Poi18] Louis Poinsignon. *BGP leaks and cryptocurrencies*. `https://blog.cloudflare.com/bgp-leaks-and-crypto-currencies/`. 2018 (cit. on p. 22).

[Poi21] David Pointcheval. *MPRI: Cryptographic protocols: formal and computational proofs*. `https://www.di.ens.fr/david.pointcheval/index.php#teaching`. 2021 (cit. on p. 27).

[pro19] Bluetooth SIG proprietary. *Bluetooth Core Specification*. `https://www.bluetooth.com/specifications/specs/core-specification/`. 2019 (cit. on p. 37).

[Qui52] Willard Van Orman Quine. "The Problem of Simplifying Truth Functions". In: *The American Mathematical Monthly* 59.8 (1952), pp. 521–531 (cit. on p. 99).

[Qui55] Willard Van Orman Quine. "A Way to Simplify Truth Functions". In: *The American Mathematical Monthly* 62.9 (1955), pp. 627–631 (cit. on p. 99).

[RBH17] Sondre Rønjom, Navid Ghaedi Bardeh, and Tor Helleseth. "Yoyo Tricks with AES". In: *ASIACRYPT 2017, Part I*. Ed. by Tsuyoshi Takagi and Thomas Peyrin. Vol. 10624. LNCS. Springer, Heidelberg, Dec. 2017, pp. 217–243 (cit. on p. 73).

[RD08] Eric Rescorla and Tim Dierks. *The Transport Layer Security (TLS) Protocol Version 1.2*. RFC 5246. Aug. 2008 (cit. on p. 48).

[Res18] Eric Rescorla. *The Transport Layer Security (TLS) Protocol Version 1.3*. RFC 8446. Aug. 2018 (cit. on pp. 23, 24, 54).

[Riv92] Ronald L. Rivest. *The MD5 Message-Digest Algorithm*. RFC 1321. Apr. 1992 (cit. on p. 50).

[Rob55] Herbert Robbins. "A Remark on Stirling's Formula". In: *The American Mathematical Monthly* 62.1 (1955), pp. 26–29. ISSN: 00029890, 19300972 (cit. on p. 62).

[Rog+01] Phillip Rogaway, Mihir Bellare, John Black, and Ted Krovetz. "OCB: A Block-Cipher Mode of Operation for Efficient Authenticated Encryption". In: *ACM CCS 2001*. Ed. by Michael K. Reiter and Pierangela Samarati. ACM Press, Nov. 2001, pp. 196–205 (cit. on pp. 35, 53).

[Rog02] Phillip Rogaway. *The EMD Mode of Operation (A Tweaked, Wide-Blocksize, Strong PRP)*. Cryptology ePrint Archive, Report 2002/148. `https://eprint.iacr.org/2002/148`. 2002 (cit. on p. 54).

[Rog11] Phillip Rogaway. *Evaluation of Some Blockcipher Modes of Operation*. Tech. rep. Davis, California, USA: University of California, Feb. 2011 (cit. on pp. 46, 47, 52, 54).

[RSA78] Ronald L. Rivest, Adi Shamir, and Leonard M. Adleman. "A Method for Obtaining Digital Signatures and Public-Key Cryptosystems". In: *Communications of the Association for Computing Machinery* 21.2 (1978), pp. 120–126 (cit. on p. 56).

[SA15] Markku-Juhani O. Saarinen and Jean-Philippe Aumasson. *The BLAKE2 Cryptographic Hash and Message Authentication Code (MAC)*. RFC 7693. Nov. 2015 (cit. on p. 51).

[Sha49] Claude E. Shannon. "Communication theory of secrecy systems". In: *Bell Systems Technical Journal* 28.4 (1949), pp. 656–715 (cit. on pp. 12, 25, 30, 32, 40, 41).

[Shi+02] Takeshi Shimoyama, Hitoshi Yanami, Kazuhiro Yokoyama, Masahiko Takenaka, Kouichi Itoh, Jun Yajima, Naoya Torii, and Hidema Tanaka. "The Block Cipher SC2000". In: *FSE 2001*. Ed. by Mitsuru Matsui. Vol. 2355. LNCS. Springer, Heidelberg, Apr. 2002, pp. 312–327 (cit. on p. 119).

[Sib20] Ferdinand Sibleyras. "Security of Modes of Operation and other provably secure cryptographic schemes". Theses. Sorbonne Université, Oct. 2020 (cit. on p. 46).

[ST17a] Yu Sasaki and Yosuke Todo. "New Algorithm for Modeling S-box in MILP Based Differential and Division Trail Search". In: *SecITC 2017*. Vol. 10543. Lecture Notes in Computer Science. Springer, 2017, pp. 150–165 (cit. on pp. 100, 103, 104, 106, 107).

[ST17b] Yu Sasaki and Yosuke Todo. "New Impossible Differential Search Tool from Design and Cryptanalysis Aspects - Revealing Structural Properties of Several Ciphers". In: *EUROCRYPT 2017, Part III*. Ed. by Jean-Sébastien Coron and Jesper Buus Nielsen. Vol. 10212. LNCS. Springer, Heidelberg, 2017, pp. 185–215 (cit. on pp. 16, 98, 100, 121, 128, 129).

[Ste+17] Marc Stevens, Elie Bursztein, Pierre Karpman, Ange Albertini, and Yarik Markov. "The First Collision for Full SHA-1". In: *CRYPTO 2017, Part I*. Ed. by Jonathan Katz and Hovav Shacham. Vol. 10401. LNCS. Springer, Heidelberg, Aug. 2017, pp. 570–596 (cit. on p. 51).

[Sun+14a] Siwei Sun, Lei Hu, Peng Wang, Kexin Qiao, Xiaoshuang Ma, and Ling Song. "Automatic Security Evaluation and (Related-key) Differential Characteristic Search: Application to SIMON, PRESENT, LBlock, DES(L) and Other Bit-Oriented Block Ciphers". In: *ASIACRYPT 2014, Part I*. Ed. by Palash Sarkar and Tetsu Iwata. Vol. 8873. LNCS. Springer, Heidelberg, Dec. 2014, pp. 158–178 (cit. on pp. 15, 70, 98, 99, 102–104).

[Sun+14b] Siwei Sun, Lei Hu, Meiqin Wang, Peng Wang, Kexin Qiao, Xiaoshuang Ma, Danping Shi, Ling Song, and Kai Fu. *Towards Finding the Best Characteristics of Some Bit-oriented Block Ciphers and Automatic Enumeration of (Related-key) Differential and Linear Characteristics with Predefined Properties*. Cryptology ePrint Archive, Report 2014/747. `https://eprint.iacr.org/2014/747`. 2014 (cit. on pp. 15, 98, 99, 102, 103, 106, 107, 119).

[Sun+16a] Bing Sun, Meicheng Liu, Jian Guo, Longjiang Qu, and Vincent Rijmen. "New Insights on AES-Like SPN Ciphers". In: *CRYPTO 2016, Part I*. Ed. by Matthew Robshaw and Jonathan Katz. Vol. 9814. LNCS. Springer, Heidelberg, Aug. 2016, pp. 605–624 (cit. on p. 73).

[Sun+16b] Bing Sun, Meicheng Liu, Jian Guo, Vincent Rijmen, and Ruilin Li. "Provable Security Evaluation of Structures Against Impossible Differential and Zero Correlation Linear Cryptanalysis". In: *EUROCRYPT 2016, Part I*. Ed. by Marc Fischlin and Jean-Sébastien Coron. Vol. 9665. LNCS. Springer, Heidelberg, May 2016, pp. 196–213 (cit. on p. 130).

[TM16] Yosuke Todo and Masakatu Morii. "Bit-Based Division Property and Application to Simon Family". In: *FSE 2016*. Ed. by Thomas Peyrin. Vol. 9783. LNCS. Springer, Heidelberg, Mar. 2016, pp. 357–377 (cit. on pp. 151, 153).

[Tod+17] Yosuke Todo, Takanori Isobe, Yonglin Hao, and Willi Meier. "Cube Attacks on Non-Blackbox Polynomials Based on Division Property". In: *CRYPTO 2017, Part III*. Ed. by Jonathan Katz and Hovav Shacham. Vol. 10403. LNCS. Springer, Heidelberg, Aug. 2017, pp. 250–279 (cit. on pp. 17, 132, 140, 145, 151–153).

[Tod15] Yosuke Todo. "Structural Evaluation by Generalized Integral Property". In: *EUROCRYPT 2015, Part I*. Ed. by Elisabeth Oswald and Marc Fischlin. Vol. 9056. LNCS. Springer, Heidelberg, Apr. 2015, pp. 287–314 (cit. on pp. 17, 132, 145).

[Wag99] David Wagner. "The Boomerang Attack". In: *FSE'99*. Ed. by Lars R. Knudsen. Vol. 1636. LNCS. Springer, Heidelberg, Mar. 1999, pp. 156–170 (cit. on p. 101).

[Wan+04] Xiaoyun Wang, Dengguo Feng, Xuejia Lai, and Hongbo Yu. *Collisions for Hash Functions MD4, MD5, HAVAL-128 and RIPEMD*. Cryptology ePrint Archive, Report 2004/199. `https://eprint.iacr.org/2004/199`. 2004 (cit. on p. 50).

[Wan+18] Qingju Wang, Yonglin Hao, Yosuke Todo, Chaoyun Li, Takanori Isobe, and Willi Meier. "Improved Division Property Based Cube Attacks Exploiting Algebraic Properties of Superpoly". In: *CRYPTO 2018, Part I*. Ed. by Hovav Shacham and Alexandra Boldyreva. Vol. 10991. LNCS. Springer, Heidelberg, Aug. 2018, pp. 275–305 (cit. on p. 152).

[Wan+19] SenPeng Wang, Bin Hu, Jie Guan, Kai Zhang, and Tairong Shi. "MILP-aided Method of Searching Division Property Using Three Subsets and Applications". In: *ASIACRYPT 2019, Part III*. Ed. by Steven D. Galbraith and Shiho Moriai. Vol. 11923. LNCS. Springer, Heidelberg, Dec. 2019, pp. 398–427 (cit. on p. 153).

[WJ19] Qian Wang and Chenhui Jin. "More accurate results on the provable security of AES against impossible differential cryptanalysis". In: *Des. Codes Cryptogr.* 87.12 (2019), pp. 3001–3018 (cit. on p. 130).

[WW11] Shengbao Wu and Mingsheng Wang. *Security Evaluation against Differential Cryptanalysis for Block Cipher Structures*. Cryptology ePrint Archive, Report 2011/551. `https://eprint.iacr.org/2011/551`. 2011 (cit. on p. 95).

[WW12] Shengbao Wu and Mingsheng Wang. "Automatic Search of Truncated Impossible Differentials for Word-Oriented Block Ciphers". In: *INDOCRYPT 2012*. Ed. by Steven D. Galbraith and Mridul Nandi. Vol. 7668. LNCS. Springer, Heidelberg, Dec. 2012, pp. 283–302 (cit. on p. 128).

[WYY05] Xiaoyun Wang, Yiqun Lisa Yin, and Hongbo Yu. "Finding Collisions in the Full SHA-1". In: *CRYPTO 2005*. Ed. by Victor Shoup. Vol. 3621. LNCS. Springer, Heidelberg, Aug. 2005, pp. 17–36 (cit. on p. 50).

[Xia+16] Zejun Xiang, Wentao Zhang, Zhenzhen Bao, and Dongdai Lin. "Applying MILP Method to Searching Integral Distinguishers Based on Division Property for 6 Lightweight Block Ciphers". In: *ASIACRYPT 2016, Part I*. Ed. by Jung Hee Cheon and Tsuyoshi Takagi. Vol. 10031. LNCS. Springer, Heidelberg, Dec. 2016, pp. 648–678 (cit. on p. 145).

[YT19] Chen-Dong Ye and Tian Tian. "Revisit Division Property Based Cube Attacks: Key-Recovery or Distinguishing Attacks?" In: *IACR Trans. Symm. Cryptol.* 2019.3 (2019), pp. 81–102. ISSN: 2519-173X (cit. on p. 152).

[ZXF13] Bin Zhang, Chao Xu, and Dengguo Feng. "Real Time Cryptanalysis of Bluetooth Encryption with Condition Masking (Extended Abstract)". In: *CRYPTO 2013, Part I*. Ed. by Ran Canetti and Juan A. Garay. Vol. 8042. LNCS. Springer, Heidelberg, Aug. 2013, pp. 165–182 (cit. on p. 37).

[Bar+18] Achiya Bar-On, Orr Dunkelman, Nathan Keller, Eyal Ronen, and Adi Shamir. "Improved Key Recovery Attacks on Reduced-Round AES with Practical Data and Memory Complexities". In: *CRYPTO 2018, Part II*. Ed. by Hovav Shacham and Alexandra Boldyreva. Vol. 10992. LNCS. Springer, Heidelberg, Aug. 2018, pp. 185–212 (cit. on p. 73).

[Hel+06] M. Hell, T. Johansson, A. Maximov, and W. Meier. "A Stream Cipher Proposal: Grain-128". In: *2006 IEEE International Symposium on Information Theory*. 2006, pp. 1614–1618 (cit. on p. 144).

[The20] The Sage Developers. *SageMath, the Sage Mathematics Software System (Version 9.1)*. `https://www.sagemath.org`. 2020 (cit. on pp. 102, 105).

Appendix A

A.1 DDT of the `Sbox` of `PRESENT`

	b															
a	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	16	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	4	0	0	0	4	0	4	0	0	0	4	0	0
2	0	0	0	2	0	4	2	0	0	0	2	0	2	2	2	0
3	0	2	0	2	2	0	4	2	0	0	2	2	0	0	0	0
4	0	0	0	0	0	4	2	2	0	2	2	0	2	0	2	0
5	0	2	0	0	2	0	0	0	0	2	2	2	4	2	0	0
6	0	0	2	0	0	0	2	0	2	0	0	4	2	0	0	4
7	0	4	2	0	0	0	2	0	2	0	0	0	2	0	0	4
8	0	0	0	2	0	0	0	2	0	2	0	4	0	2	0	4
9	0	0	2	0	4	0	2	0	2	0	0	0	2	0	4	0
10	0	0	2	2	0	4	0	0	2	0	2	0	0	2	2	0
11	0	2	0	0	2	0	0	0	4	2	2	2	0	2	0	0
12	0	0	2	0	0	4	0	2	2	2	2	0	0	0	2	0
13	0	2	4	2	2	0	0	2	0	0	2	2	0	0	0	0
14	0	0	2	2	0	0	2	2	2	2	0	0	2	2	0	0
15	0	4	0	0	4	0	0	0	0	0	0	0	0	0	4	4

A.2 `MixColumns` MILP model for the AES

Algorithm 10 applied on the AES `MixColumns` outputted the matrices below. The lines are represented as hexadecimal numbers with the first column element being the least significant bit (i.e. the rightmost one). Moreover, the matrix $\boldsymbol{Q}$ is given by

$$\boldsymbol{Q} = \begin{pmatrix} \boldsymbol{Q}_0 & & & & & & & \\ & \boldsymbol{Q}_0 & & & & & & \\ & & \boldsymbol{Q}_0 & & & & & \\ & & & \boldsymbol{Q}_0 & & & & \\ & & & & \boldsymbol{I} & & & \\ & & & & & \boldsymbol{I} & & \\ & & & & & & \boldsymbol{I} & \\ & & & & & & & \boldsymbol{I} \end{pmatrix}.$$

$\boldsymbol{A}$	$\boldsymbol{P}$	$\boldsymbol{Q}_0$	$\boldsymbol{P}\cdot\boldsymbol{A}\cdot\boldsymbol{Q}$
101018180	10101		1010101000080
202028381	2		202020301
404040602	4		404040602
808088c84	8000808		8000808840c0000
1010109888	10		1010101808
2020203010	20		2020203010
4040406020	40		4040406020
808080c040	80		808080c040
10001818001	1010001		101000100800100
20002838102	200		20002030102
40004060204	400		40004060204
800088c8408	8080800		80808000000840c
100010988810	1000	81	100010180810
200020301020	2000	2	200020301020
400040602040	4000	4	400040602040
800080c04080	8000	88	800080c04080
1000081800101	1010100	10	101010000008001
2000083810202	20000	20	2000003010202
4000006020404	40000	40	4000006020404
800008c840808	80808	80	808080c000084
10000098881010	100000		10000018081010
20000030102020	200000		20000030102020
40000060204040	400000		40000060204040
800000c0408080	800000		800000c0408080
100000080010181	1000101		100010180010000
200000081020283	2000000		200000001020203
400000002040406	4000000		400000002040406
80000008408088c	8080008		808000800840c00
1000000088101098	10000000		1000000008101018
2000000010202030	20000000		2000000010202030
4000000020404060	40000000		4000000020404060
80000000408080c0	80000000		80000000408080c0

A.3 Linear layers of `Saturnin` and `Aria`

`Saturnin`'s $\boldsymbol{M}$ transformation is given by

$$\boldsymbol{M}:\begin{pmatrix}\boldsymbol{a}\\ \boldsymbol{b}\\ \boldsymbol{c}\\ \boldsymbol{d}\end{pmatrix}\mapsto\begin{pmatrix}\alpha^2(\boldsymbol{a})\oplus\alpha^2(\boldsymbol{b})\oplus\alpha(\boldsymbol{b})\oplus\boldsymbol{c}\oplus\boldsymbol{d}\\ \boldsymbol{a}\oplus\alpha(\boldsymbol{b})\oplus\boldsymbol{b}\oplus\alpha^2(\boldsymbol{c})\oplus\boldsymbol{c}\oplus\alpha^2(\boldsymbol{d})\oplus\alpha(\boldsymbol{d})\oplus\boldsymbol{d}\\ \boldsymbol{a}\oplus\boldsymbol{b}\oplus\alpha^2(\boldsymbol{c})\oplus\alpha^2(\boldsymbol{d})\oplus\alpha(\boldsymbol{d})\\ \alpha^2(\boldsymbol{a})\oplus\boldsymbol{a}\oplus\alpha^2(\boldsymbol{b})\oplus\alpha(\boldsymbol{b})\oplus\boldsymbol{b}\oplus\boldsymbol{c}\oplus\alpha(\boldsymbol{d})\oplus\boldsymbol{d}\end{pmatrix}$$

where $\alpha(\boldsymbol{x})=\begin{pmatrix}0&1&0&0\\0&0&1&0\\0&0&0&1\\1&1&0&0\end{pmatrix}\cdot\boldsymbol{x}$.

Aria's diffusion layer matrix is given by

$$\begin{pmatrix}
0 & 0 & 0 & 1 & 1 & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 1 & 1 & 0 \\
0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 1 \\
0 & 1 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 1 \\
1 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 1 & 1 & 0 \\
1 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 1 \\
0 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\
1 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 1 & 1 & 0 & 0 \\
0 & 1 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 1 & 1 & 1 & 0 & 0 \\
1 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 \\
1 & 1 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 0 \\
0 & 0 & 1 & 1 & 0 & 1 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 1 \\
0 & 0 & 1 & 1 & 1 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 0 \\
0 & 1 & 1 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 0 \\
1 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 1 & 0 & 1 & 0 & 0 & 1 & 0 & 0 \\
1 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 1 & 0 \\
0 & 1 & 1 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 1
\end{pmatrix}.$$