# Optimization of core components of block ciphers

Baptiste Lambin

▶ **To cite this version:**

Baptiste Lambin. Optimization of core components of block ciphers. Cryptography and Security [cs.CR]. Université de Rennes, 2019. English. NNT : 2019REN1S036 . tel-02380098

**HAL Id: tel-02380098**
**https://theses.hal.science/tel-02380098**

Submitted on 26 Nov 2019

**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.

UNIVERSITE BRETAGNE LOIRE MATHSTIC

UNIVERSITÉ DE RENNES 1

# THÈSE DE DOCTORAT DE

L'UNIVERSITE DE RENNES 1
**C**OMUE **U**NIVERSITE **B**RETAGNE **L**OIRE

Ecole Doctorale N°601
*Mathématique et Sciences et Technologies de l'Information et de la Communication*
Spécialité : *Informatique*

Par


## Baptiste LAMBIN


## Optimization of Core Components of Block Ciphers

**Thèse présentée et soutenue à** RENNES, le 22/10/2019
**Unité de recherche : IRISA**

**Rapporteurs avant soutenance :**
Marine Minier, Professeur, LORIA, Université de Lorraine
Jacques Patarin, Professeur, PRiSM, Université de Versailles

**Composition du jury :**

| | |
|---|---|
| Examinateurs : | Marine Minier, Professeur, LORIA, Université de Lorraine |
| | Jacques Patarin, Professeur, PRiSM, Université de Versailles |
| | Jean-Louis Lanet, INRIA Rennes |
| | Virginie Lallemand, Chargée de Recherche, LORIA, CNRS |
| | Jérémy Jean, ANSSI |
| Dir. de thèse : | Pierre-Alain Fouque, IRISA, Université de Rennes 1 |
| Co-dir. de thèse : | Patrick Derbez, IRISA, Université de Rennes 1 |

# Remerciements

Je tiens à remercier en premier lieu mes directeurs de thèse, Pierre-Alain et Patrick. Merci à Pierre-Alain pour ses conseils avisés, sa patience (presque) à toute épreuve et sa gentillesse. Merci à Patrick pour m'avoir guidé et assisté durant cette thèse, avec ses idées parfois improbables qui surgissent sans prévenir. Merci également à la DGA pour avoir financé ma thèse. Je remercie ensuite Marine et Jacques pour avoir accepté de rapporter ce manuscrit, ainsi que le reste du jury, Jean-Louis, Jérémy et Virginie (merci également à cette dernière pour la relecture complète du manuscrit).

Je remercie bien sûr l'ensemble de l'équipe EMSEC qui fût un excellent environnement de travail pendant ces trois ans, toujours avec une bonne ambiance générale. En particulier je remercie Adeline pour ses conseils avisés et sa patience, Mohammed et Clémentine pour leur bonne humeur permanente, sans oublier Stéphanie, Barbara et Gildas. Ces trois ans ne se serait jamais aussi bien passé sans toute l'équipe de thésards et post-docs qui ont défilés dans l'équipe. Merci en particulier à Pauline, cette thèse aurait été très différente si tu n'avais pas été là, merci pour ton soutien, pour m'avoir supporté si longtemps, pour les voyages en voiture (merci Thomas !), les phrases mal tournées, le soutien dans l'humour beauf, la découverte des véritables bonnes bières, et pour tout le reste. Merci aussi à Angèle pour sa bonne humeur et son sourire omniprésent, ses conseils et son soutien, pour avoir rigolé à mes blagues nulles, pour m'avoir fait rigolé à des blagues nulles, pour les références à Brooklyn 99 et bien d'autres choses encore, la suite ne sera également pas pareil sans toi. Merci à Claire pour m'avoir supporté pendant 2 ans dans le bureau, pour les débats et discussions sans queue ni tête, on remet ça pour un an en Allemagne ! Merci à Victor pour avoir pris la relève de la Team Sym', pour ses explications sur le Brexit et autres problèmes géopolitiques. Je remercie également Chen pour les discussions improbables et les grands moments d'incompréhension et à Alexandre pour son accent et m'avoir montré qu'il existe bien des gens capables de finir un Iron Man (et d'aller continuer à courir deux jours après). Enfin merci à tous les autres, Alban, Thomas, Céline, Katharina, Guillaume, Weiqiang, Solène, Olivier, Gauthier, Joshua, au stagiaire blondinet (Gwendal sur l'état civil), ainsi qu'aux disparus : Florent, Xavier, Brice, Benjamin, Cyrille, Raphaël, Pierre, Pierre et Thomas. Durant ces trois années j'ai également rencontré beaucoup de gens formidables, comme les symétriciens de Paris et notamment Yann, André, Sébastien, Ferdinand, Léo, Anne, Maria et Christina, ainsi que les membres de l'ANR Decrypt que je n'ai pas encore cité comme Paul, Virginie, Pascal, Charles, Christine, Charlotte et François. Je remercie aussi toutes les personnes qui m'ont entouré durant ces dernières années et qui n'ont probablement jamais compris ce que je faisais réellement, Élodie, Émilie, Benben, Lulu, Erwan, Sofia, les détraqués du Pub et en particulier Régis et Aude, le Kenshikaï de Rennes ainsi que ma famille. Bien sûr, je remercie également tous ceux que j'ai oublié et/ou dont le nom m'échappe.

Pour finir, je tiens à remercier plusieurs personnes et groupes de personnes qui ne seront probablement jamais au courant de cette thèse, mais qui m'ont tout de même aidé durant ces dernières années. Merci notamment à GGG pour avoir créer Path of Exile et ainsi me permettre d'avoir de quoi me défouler le soir, Valve pour avoir créer Steam et Left 4 Dead 2, donnant lieu à plusieurs soirées mémorables, Capcom pour avoir enfin sorti un Monster Hunter sur PC, Team Cherry et Matt Makes Games pour ces deux magnifiques jeux que sont respectivement Hollow Knight et Celeste, et tous les autres développeurs qui ont sorti des jeux qui m'ont diverti durant ces trois ans. Merci à Flo et toute son équipe pour ces bons moments de détente et ces délicieux breuvages. Merci à tous les groupes dont la musique m'a accompagné tous les jours depuis plusieurs années que ce soit pendant les différents trajets ou bien (et surtout) pendant la rédaction, en particulier Tool (merci pour

avoir enfin sorti un superbe nouvel album depuis 13 ans), Meshuggah, Beyond Creation, Eluveitie, Obscura, Gloryhammer, Opeth, Alestorm, et tant d'autres. Merci aux différents écrivains qui m'ont fait voyagé notamment Robin Hobb, Jean-Christophe Grangé, Franck Thilliez, Maxime Chattam, Pierre Bottero (RIP), James Clemens et d'autres. Merci à tous les auteurs, acteurs, réalisateurs, scénaristes, mangakas, dessinateurs etc. qui ont amené tant de films, séries et anime de qualité durant ces dernières années.

Enfin pour finir, merci (ou pas ?) au monstre sanguinaire, la terreur des boulettes d'aluminium, le Némésis du papier-peint, la grosse bête poilue, le chauffe pied en hiver, toujours et malheureusement chauffe pied en été, le chat qui voulait apprendre à voler et qui ne sait pas miauler, Mojito, dit Gromomo, Le Monstre, Monstromomo ou tout simplement Momo.

# Contents

# Introduction et Résumé en Français

Sécuriser les communications a toujours été un problème majeur dans l'Histoire. Que ce soit l'envoi de messages politiques ou stratégiques par des gouvernements ou des entreprises, la communication d'un particulier avec sa banque ou son assurance, ou bien simplement un étudiant souhaitant envoyer un message à son voisin, nous avons toujours eu le besoin de protéger nos échanges contre les observateurs indiscrets, voir même mal intentionnés. Si les premiers systèmes de chiffrement étaient assez rudimentaires, comme le chiffrement de César, la complexité de tels systèmes a peu à peu évolué durant l'Histoire, arrivant jusqu'à la machine Enigma lors de la Seconde Guerre Mondiale, et explosant complètement avec l'essor des ordinateurs modernes.

Malgré la révolution amenée par la création des systèmes de chiffrement à clé publique, ou asymétriques, avec RSA en 1977 [114], la cryptographie à clé secrète, ou symétrique, reste au cœur de la sécurisation des communications modernes. L'essence même de la cryptographie symétrique, ainsi que l'origine du nom, repose sur le fait que les deux partis cherchant à communiquer possèdent une information connue d'eux seuls, la clé, qui est utilisée pour pouvoir chiffrer et déchiffrer les communications. Informellement, la conception d'algorithmes de chiffrement symétriques consiste donc à élaborer un processus prenant en entrée cette clé et un message *clair*, et de produire un message *chiffré* que seul le ou les possesseurs légitimes de la clé secrète peuvent déchiffrer. Trois notions principales de sécurité sont utilisées en cryptographie symétrique : la confidentialité, l'intégrité et l'authenticité. La confidentialité cherche à assurer que seules les personnes en possession de la clé secrète peuvent déchiffrer le message. Ce rôle est rempli à l'aide de primitives de *chiffrement*, qui sont également l'objet d'étude principal de cette thèse. L'intégrité quant à elle consiste à être sûr que le clair/chiffré n'ait pas été modifié durant la communication, ce rôle étant le plus souvent rempli à l'aide de *fonctions de hachage*. Enfin, l'authenticité permet de garantir que le chiffré provient bien d'une personne possédant la clé secrète, cette fois-ci en utilisant un *Code d'Authentification de message* (MAC). Pour finir, il est également possible de créer une primitive qui regroupe ces trois notions de sécurité, donnant lieu aux *chiffrements authentifiés*.

Bien que plus ancienne que la cryptographie à clé publique, la cryptographie symétrique est toujours un sujet de recherche très actif. Une étape notable dans l'histoire de la cryptographie symétrique est la normalisation du chiffrement DES en 1977 [56]. Au début des années 70, le *National Bureau of Standards* (ancien nom du NIST, *National Institute of Standards and Technology*) lance un appel pour concevoir un nouveau système de chiffrement pour créer une norme américaine. Le processus de sélection ne fût pas rendu public, mais leur choix se porta sur un ancien algorithme d'IBM créé par Horst Feistel, nommé Lucifer [64]. Cependant, la NSA (*National Security Agency*) demanda à IBM d'apporter certaines modifications à l'algorithme, notamment en réduisant la taille de la clé, ainsi qu'en modifiant l'un des composants appelé *boîte-S*. Ces modifications furent appliquées, donnant lieu au DES tel qu'on le connaît aujourd'hui. Cependant, la participation de la NSA dans sa conception ne fût pas appréciée par l'ensemble de la communauté. Notamment, quand Biham et Shamir découvrent la cryptanalyse différentielle [17], ils remarquent que DES est étonnamment résistant à cette forme de cryptanalyse inconnue du public jusque là. Il fût ensuite confirmé par Coppersmith, l'un des membres de l'équipe d'IBM, que cette technique était déjà connue lors de la conception des boîtes-S de DES [44].

Ce manque de transparence donna lieu à un nouveau format pour la sélection de nouvelles normes. En effet, le processus de sélection du successeur de DES prit le format d'une compétition publique, la compétition AES (*Advanced Encryption Standard*). Initiée par le NIST le 2 Janvier

1997, 15 candidats furent reçus et rendus publics, laissant ainsi la communauté scientifique réaliser différentes analyses de sécurité pour chacun de ces algorithmes. Après avoir réduit la liste à 5 candidats "finalistes" le 9 Août 1999 (Rijndael, MARS, RC6, Serpent et Twofish), le NIST annonce que Rijndael est le candidat retenu pour devenir l'AES que l'on connaît aujourd'hui. Le même processus fût utilisé pour le choix d'une nouvelle norme pour les fonctions de hachages, donnant lieu à l'ouverture de la compétition SHA-3 le 23 Janvier 2007, et dont le choix se porta sur Keccak après 5 ans d'analyse par la communauté. Cette normalisation de Keccak créa également un nouveau champ de recherche pour la cryptographie symétrique, puisque Keccak utilise une nouvelle construction appelée *construction éponge*, qui permet de créer des primitives de chiffrement et de hachage simplement à partir d'une permutation. D'autres appels à soumission de format similaires eurent également lieu, comme la compétition eSTREAM financée par l'Union Européenne, qui permit de donner un portfolio de chiffrements à flot recommandés, ou bien la compétition CAESAR, achevée au début de l'année 2019, qui apporta un portfolio pour les chiffrements authentifiés. Enfin, la dernière compétition initiée par le NIST en date concerne la normalisation d'une ou plusieurs primitives de *chiffrement léger*, i.e. des systèmes de chiffrement conçus pour les systèmes ayant de fortes contraintes techniques comme les cartes à puces. Cette compétition reçut 56 soumissions le 18 Avril 2019, avec une participation active de la communauté dans l'étude des différents candidats.

Étudier les différents critères pour élaborer un système de chiffrement symétrique sûr tout en conservant de bonnes performances et la viabilité d'une implémentation sur des composants à performances restreintes est donc toujours un problème crucial, d'où l'objectif de cette thèse, qui est de donner de nouvelles observations lors de l'élaboration de nouvelles primitives symétriques. De manière très générale, on peut donc donner la définition suivante, en considérant que tous les messages et clés sont représentés par une chaîne de bits de longueur arbitraire.

**Definition 0.1.** *Un chiffrement est une famille de fonctions $\mathcal{F} : \mathbb{F}_2^* \times \mathbb{F}_2^* \to \mathbb{F}_2^*$, où $\mathbb{F}_2^*$ est l'ensemble des chaines de bits de longueur arbitraire, telle que pour n'importe quel élément $p \in \mathbb{F}_2^*$ (le message) et $K \in \mathbb{F}_2^*$ (la clé), et en notant $c = \mathcal{F}(K, p)$ (le chiffré), alors il existe une fonction $\mathcal{F}' : \mathbb{F}_2^* \times \mathbb{F}_2^* \to \mathbb{F}_2^*$ calculable efficacement telle que $p = \mathcal{F}'(K, c)$.*

De plus en pratique, la clé est souvent de taille fixe, et ce sera le cas tout au long de cette thèse. Une exception existe cependant, et non des moindres, le *masque jetable*. Ce chiffrement est extrêmement simple : étant donné un message $p$ et une clé $K$ (aussi appelée masque dans ce cas), le chiffré est simplement $c = K \oplus p$. Malgré sa simplicité, ce chiffrement est le seul chiffrement parfait connu, ce qui a été prouvé par Shannon en 1949 [118], mais implique des contraintes particulièrement difficiles à utiliser en pratique :

- La clé doit faire *au moins* la même taille que le clair
- La clé doit être complètement *aléatoire*
- La clé ne doit être utilisée qu'une seule fois, i.e. on ne doit pas chiffrer deux messages, même identiques, avec la même clé.

Il est clair que ces contraintes sont presque impossibles à satisfaire en pratique dans un système à grande échelle, même si le système a été utilisé à plusieurs reprises au cours de l'Histoire.

Les cryptographes ont donc cherché à concevoir des solutions plus simples, tout en essayant de conserver un maximum de sécurité. En cryptographie symétrique, deux grandes familles de chiffrements existent. Le premier type est la famille des *chiffrements à flot*. L'origine de ce type de chiffrement est très simple : s'approcher au maximum du masque jetable, sans en avoir les contraintes. L'idée est donc, plutôt que de générer une clé aussi grande que le message, de partir

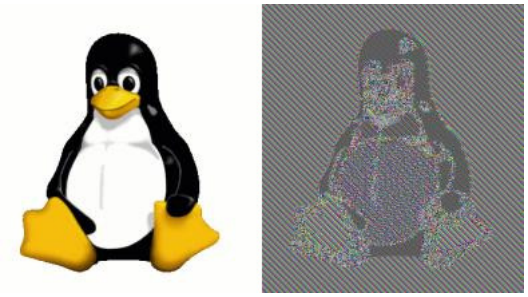

Figure 1.1: L'image originale (à gauche) et son chiffré avec ECB (à droite) [63]

d'une clé de taille fixe et de l'utiliser pour générer une *suite chiffrante* de la longueur du message que l'on utilisera de la même manière que le masque jetable. La conception de ce type de chiffrement revient donc à faire en sorte que cette suite chiffrante se comporte au maximum comme une suite aléatoire, afin de se rapprocher d'un masque jetable. De nombreux chiffrements à flot existent, par exemple RC4 [124] qui, malgré ses faiblesses connues, est toujours utilisé dans certaines applications comme le protocole WEP. La compétition eSTREAM mentionnée précédemment a cependant permis d'obtenir une liste de chiffrements à flot bien plus résistants que RC4 comme par exemple Trivium, et l'un des finalistes de la compétition CAESAR, ACORN [141], est également un chiffrement à flot. Cette thèse se concentre cependant sur la deuxième famille de chiffrements, les *chiffrements par bloc*, décrits dans la section suivante.

## 1 Chiffrements par Bloc

Nous commençons par la définition du principal sujet d'étude de cette thèse.

**Definition 1.1.** *Un chiffrement par bloc est une famille de fonctions* $E : \mathbb{F}_2^k \times \mathbb{F}_2^n \to \mathbb{F}_2^n$ *telle que chaque fonction* $E_K = E(K, \cdot) : \mathbb{F}_2^n \to \mathbb{F}_2^n$ *soit une permutation. On appellera* $k$ *la taille de la clé, et* $n$ *la taille de bloc.*

D'après la Définition 0.1, un chiffrement doit pouvoir chiffrer des clairs de longueur arbitraire, alors qu'un chiffrement par bloc ne peut chiffrer que des clairs de taille fixe $n$. Pour cela, nous avons besoin de *modes opératoires*, qui découpent le clair en blocs de longueur $n$, puis appliquent un algorithme permettent de chiffrer l'ensemble. Le mode le plus basique est le mode *Electronic Code Book* (ECB), où chaque bloc est chiffré indépendamment. Ce mode est fortement déconseillé à partir du moment où la longueur du clair est supérieure à $n$, car deux blocs identiques seront chiffrés de la même manière, donnant donc le même chiffré et pouvant donc laisser fuir de l'information, voir par exemple la Figure 1.1.

Un mode qui résout ce problème est par exemple le mode *Cipher Block Chaining* (CBC). Ce mode a besoin d'une entrée supplémentaire appelée un *vecteur d'initialisation*, qui doit être unique à chaque message chiffré. En notant $IV$ ce vecteur d'initialisation, et $p_i$ (resp. $c_i$) le $i$-ème bloc de clair (resp. de chiffré), alors le mode CBC est défini par

$$\begin{aligned} c_0 &= IV \\ c_i &= E_K(p_i \oplus c_{i-1}), i > 0. \end{aligned}$$

Cependant, le mode CBC n'est pas sans défaut, et de nombreux autres modes existent, comme par exemple OFB, CFB où CTR. Cette thèse ne se concentrant pas sur les modes opératoires, nous ne donnerons pas plus de détails ici.

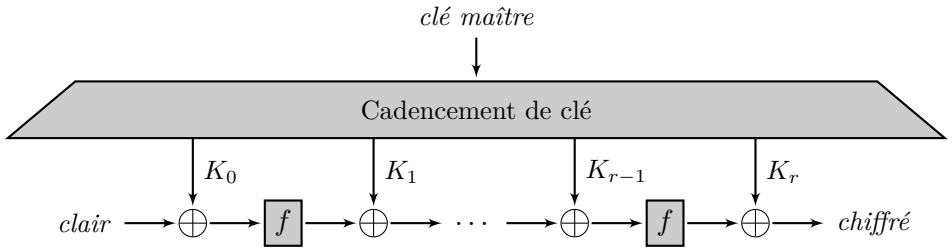


Figure 1.2: La construction d'un chiffrement par bloc itéré [78]

Construire directement un chiffrement par bloc est une tâche difficile, à la fois pour l'élaboration mais également pour étudier la sécurité de la primitive qui en résulte. De fait, la plupart des chiffrements par bloc modernes utilisent la construction itérée représentée dans la Figure 1.2. L'idée est de concevoir deux composants principaux. Le premier est l'algorithme de *cadencement de clé*, qui prend en entrée la clé, souvent appelée *clé maître* dans ce cas, et produit une suite de *clés de tour* $K_0, \ldots, K_r$, où $r$ est appelé le *nombre de tours*. Ensuite, la *fonction de tour* $f$, qui est utilisée avec le cadencement de clé pour construire la fonction de chiffrement

$$E_K = \oplus_{K_r} \circ f \circ \oplus_{K_{r-1}} \circ f \circ \cdots \circ f \circ \oplus_{K_1} \circ f \circ \oplus_{K_0},$$

où $K_0, \ldots, K_r$ sont les clés de tour dérivées de la clé maître $K$, et $\oplus_K(x) = x \oplus K$. Construire directement un chiffrement par bloc résistant est beaucoup plus difficile que de concevoir une fonction de tour acceptable, même si plus faible, qui sera répétée plusieurs fois. Ceci facilite également l'étude d'un chiffrement par bloc construit de cette manière, puisqu'on peut commencer par examiner la fonction de tour, ce qui est plus simple que d'étudier le chiffrement dans son ensemble.

Dans le Chapitre 3, nous donnons quelques observations sur l'élaboration du cadencement de clé. De manière générale, c'est un composant donc la conception est encore relativement mal comprise par la communauté, et beaucoup de constructions différentes existent. L'actuel standard de chiffrement AES possède un cadencement de clé non linéaire relativement complexe, alors que celui de PRESENT revient essentiellement à une permutation linéaire suivie par une opération non linéaire sur seulement 4 bits et l'addition d'une constante. Certains chiffrements par bloc utilisent un cadencement de clé linéaire, comme IDEA [93] ou Square [45], et certains vont même plus loin en utilisant presque directement la clé maître, ajoutant seulement une constante différente entre chaque tour, comme LED [73] et Midori [5]. La conception de la fonction de tour cependant est bien mieux comprise. Il y a deux principales constructions utilisées pour élaborer des chiffrements par bloc :

**Les Réseaux de Feistel.** Introduits lors de la création de l'ancien standard de chiffrement DES [56], les réseaux de Feistel sont toujours utilisés aujourd'hui dans des chiffrement par bloc comme Camellia [2] ou SIMON [8]. L'idée est de partager l'état interne de $n$ bits en deux moitiés. Ainsi, au tour $i$, l'état interne est représenté par $(L_i, R_i)$, où $L_i$ et $R_i$ sont tous deux des blocs de $n/2$ bits. La fonction de tour est ensuite définie par

$$(L_{i+1}, R_{i+1}) = (R_i, L_i \oplus F_{K_i}(R_i)),$$

où $F_{K_i}$ est une fonction qui dépend de la $i$-ème clé de tour. En plus de propriétés théoriques intéressantes (e.g. [100]), cette construction est efficace à la fois en implémentation logicielle et matérielle, en partie parce que son inverse consiste essentiellement en la même fonction, sauf que les clés de tour sont utilisées dans l'ordre inverse. Certaines généralisations de cette construction furent ensuite publiées par Zheng et al. [147] à CRYPTO'89, donnant notamment lieu à la construction

dite *Feistel de Type 2*, où la fonction de tour consiste en l'application parallèle de plusieurs réseaux de Feistel, suivie par une permutation. Cette construction fût par exemple utilisée pour concevoir les chiffrements par bloc TWINE [132] et Simpira [71]. Dans le Chapitre 1, nous donnons plus de détails sur cette construction de Feistel de Type 2, et plus précisément sur comment choisir cette permutation.

**Réseaux de Substitution et Permutation (SPN).** Cette construction plus récente fût proposée en premier lors de la création du chiffrement par bloc Square [45], qui donna lieu à AES. De nombreux chiffrements ont été élaborés de cette manière, comme PRESENT [33], Midori [5] ou SKINNY [10]. L'idée de cette construction est d'utiliser directement des composants qui apportent de la *diffusion* et de la *confusion*. Ces termes ont été introduits par Shannon en 1949 [118]. L'objectif de la *confusion* est de rendre la plus complexe possible la relation entre le clair, le chiffré et la clé. Quant à la *diffusion*, son rôle est de faire en sorte que chaque bit du clair et de la clé doit influencer le plus de bits possibles dans le chiffré. Dans cette construction, la fonction de tour est composée de trois étapes :

- **La couche de substitution**, dont le but est de fournir de la *confusion* en étant une fonction non-linéaire. Cette étape est souvent construite en concaténant plusieurs *boîtes-S*, qui sont de petites (en terme de nombre de bits) fonctions non-linéaires inversibles, et en les appliquant en parallèle sur tout ou sur une partie de l'état interne. Le choix de la boîte-S est donc crucial, puisqu'il s'agit du seul composant non-linéaire de la fonction de tour. Dans le Chapitre 2, nous donnons, parmi d'autres résultats, un nouveau critère pour le choix de ces boîtes-S. Il est à noter que certains chiffrements par bloc ont une couche de substitution plus élaborée, comme SPARX [58].

- **La couche de permutation**, dont le but est de donner de la *diffusion* à la fonction de tour. Bien que son nom suggère qu'il pourrait seulement s'agir d'une permutation au niveau des bits (comme dans PRESENT), cette construction est plus générale et cette couche consiste simplement en une application linéaire. La conception de cette étape est tout aussi importante que la couche de substitution. Certains chiffrements par bloc ont été conçus avec une couche linéaire très robuste, comme AES, qui fournit une diffusion très rapide à travers la fonction de tour. D'autres, comme SKINNY, utilisent une fonction linéaire avec une diffusion moindre, mais conduisent à de meilleures performances. Ceci est souvent le résultat d'un compromis entre performance et sécurité : une couche linéaire robuste va avoir une diffusion forte et donc besoin de moins de tours, mais elle peut être coûteuse à implémenter. D'un autre coté, une diffusion plus faible peut souvent amener à de meilleures performances pour la fonction de tour, au prix d'avoir besoin de plus de tours pour atteindre un niveau de sécurité similaire.

- **L'ajout de clé**, consiste simplement à ajouter la clé de tour à l'état avec un XOR. Elle peut être ajoutée sur l'état complet comme dans AES, ou bien seulement sur une partie de l'état comme dans SKINNY.

Une autre classe de chiffrement par bloc existe en parallèle de ces deux constructions, les chiffrements de type ARX comme SPECK [9]. Ces chiffrements par bloc utilisent essentiellement trois opérations principales : l'Addition modulaire, la Rotation bit-à-bit et le XOR. Il existe également la variante AndRX, utilisant un ET logique à la place de l'addition modulaire, e.g. SIMON [8]. Cette construction est un peu à part puisqu'elle est souvent utilisée en même temps que l'une des deux constructions précédentes, utilisant seulement les opérations ARX pour construire les différentes sous-fonctions nécessaires. Par exemple, SPARX [58] est un SPN utilisant les opérations ARX, alors que HIGHT [76], SPECK [8] ou XTEA [140] sont des réseaux de Feistel utilisant les opérations ARX

(ainsi que SIMON qui est également un réseau de Feistel, mais utilisant les opérations AndRX). Pour finir, les SPNs et les réseaux de Feistel ne sont pas les seules constructions existantes, on peut notamment citer les constructions Lai-Massey [93], Even-Mansour [62] ou bien la construction MISTY [104].

## 2 Modèles de sécurité

Étudier la sécurité des chiffrements par bloc demande de connaître les capacités de l'attaquant. On modélise souvent l'interaction entre l'attaquant et le chiffrement par bloc avec le concept d'*oracle*. Cet oracle permet d'étudier de manière théorique les possibilités de l'attaquant pour attaquer le chiffrement, en pratique il pourrait par exemple s'agir d'une carte à puce contenant un algorithme de chiffrement, ou bien un service web. Durant l'étude d'un chiffrement par bloc $E$, l'oracle va commencer par choisir une clé $K$ au hasard, qui reste secrète et ne change pas, puis va répondre aux requêtes de l'attaquant en utilisant la fonction de chiffrement correspondante $E_K$. L'attaquant va ensuite essayer de retrouver la clé $K$ simplement à partir des réponses de l'oracle, ou bien dans certains cas amenant à des attaques plus faibles, déchiffrer un chiffré sans connaître a priori le clair correspondant. Il est à noter que d'après le principe de Kerckhoffs, l'attaquant connaît exactement quel chiffrement par bloc est utilisé par l'oracle, ainsi que comment chaque opération est réalisée (e.g. la boîte-S, la couche linéaire etc.). La *seule* information tenue secrète est la clé.

Nous devons maintenant considérer quel type de requête l'attaquant est autorisé à envoyer, ceci définissant le modèle de sécurité considéré. Le modèle le plus faible autorise seulement l'attaquant à demander un (ou plusieurs) chiffrés à l'oracle. L'oracle va simplement générer un clair aléatoire, *inconnu* de l'attaquant, et renvoie le chiffré correspondant. Ce modèle est appelé le modèle à *chiffré connu*, puisque l'attaquant connaît seulement un ensemble de chiffrés sans les clairs correspondants. Une attaque dans ce modèle est désastreuse, puisque les contraintes pour l'attaquant sont extrêmement faibles : simplement avoir accès à un ensemble de chiffrés (aléatoires). Ce modèle est donc très faible, puisque le pouvoir de l'attaquant est très limité. Aucun chiffrement par bloc moderne sérieux n'est vulnérable à ce type d'attaque de nos jours, mais certains plus anciens comme les chiffrements de César ou de Vigenère peuvent facilement être cassés dans ce modèle. On peut cependant noter que le chiffrement PC1 utilisé dans les tablettes *Kindle* d'Amazon est vulnérable à une attaque dans ce modèle [27].

Un modèle légèrement plus fort est le modèle à *clair connu*. Dorénavant, en recevant une requête de l'attaquant, l'oracle va générer un clair aléatoire et calcule le chiffré correspondant, mais cette fois-ci envoie à la fois le clair et le chiffré à l'attaquant. Certaines attaques très connues dans ce modèle sont les attaques contre la machine Enigma par le Bletchley Park pendant la Seconde Guerre Mondiale, ou plus récemment sur le chiffrement par bloc FEAL [133, 105]. Certaines techniques de cryptanalyse moderne se placent dans ce modèle, comme la cryptanalyse linéaire [105] ou bien la cryptanalyse à corrélation nulle [34]. Cependant, ce n'est pas suffisant pour d'autres techniques de cryptanalyse.

Pour analyser un chiffrement par bloc, nous avons souvent besoin d'être au moins dans le modèle à *clair choisi*. Maintenant, l'attaquant peut choisir quel clair envoyer à l'oracle pour récupérer le chiffré correspondant. Ceci donne bien plus de pouvoir à l'attaquant et amène à des techniques de cryptanalyses comme la cryptanalyse différentielle ou intégrale. Plusieurs chiffrements par bloc modernes proposés par la communauté sont vulnérables à des attaques sur le nombre complet de tours dans ce modèle, comme par exemple MISTY1 [135], KLEIN-64 [94], ou Robin et Zorro [96]. Ce type d'attaque a également été appliqué sur des variantes réduites (i.e. avec moins de tours) d'essentiellement tous les chiffrements par bloc modernes. Ce modèle est considéré comme le niveau

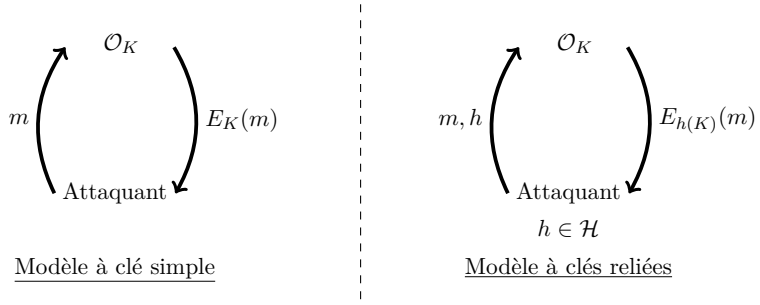


Figure 2.3: Modèle à clé simple vs. Modèle à clés reliées pour une requête en clair choisi.

de sécurité de base pour un chiffrement par bloc, et utiliser n'importe quel chiffrement vulnérable à une attaque dans ce modèle (ou l'un des précédent) serait une erreur de sécurité majeure.

Cependant, il existe également un modèle plus fort, le modèle à *chiffré choisi*. Ici, l'attaquant, en plus de pouvoir demander le chiffré correspondant à un clair de son choix, peut également demander l'inverse, c'est-à-dire envoyer un chiffré à l'oracle, qui envoie en retour le clair correspondant. Ce modèle amène à des attaques comme par exemple les attaques boomerang , qui ont permi de casser COCONUT98 avec une complexité pratique [138], ainsi que les attaques différentielles-linéaires [95] et les attaques yoyo [13].

Pour finir, les deux derniers modèles peuvent également être considérés dans une variante dite *adaptative*, où l'attaquant envoie un certain nombre de requêtes à l'oracle, récupère le résultat puis peut faire de nouvelles requêtes basées sur le résultat. Ceci est légèrement différent des modèles standards, où l'attaquant envoie un seul ensemble de requêtes à l'oracle, et n'est plus autorisé à interagir avec celui-ci par la suite. Ces variantes amènent à des modèles encore plus forts que leurs versions standard.

Nous avons donc décrit les principaux modèles permettant de représenter le pouvoir de l'attaquant. Cependant, d'autres facteurs sont à prendre en compte en même temps que les modèles précédemment décrits, donnés dans les sections suivantes.

## 2.1 Clé Simple ou Clés Reliées ?

Les modèles de sécurité précédemment décrits ont historiquement été donnés dans le modèle *à clé simple*. C'est-à-dire, une fois que l'oracle a choisi une clé aléatoire, cette clé ne change pas et l'attaquant n'a aucun contrôle sur cette dernière. Cependant, à EUROCRYPT'93 [12], Biham décrivit un nouveau type d'attaque, donnant lieu au modèle *à clés reliées*. Dans ce modèle, l'oracle choisit toujours une clé aléatoire $K$ tenue secrète pour l'attaquant. Cependant, quand l'attaquant envoie une requête à l'oracle, il peut également envoyer une fonction $h$ appartenant à une certaine classe de fonction $\mathcal{H}$, et l'oracle va répondre en utilisant la clé $h(K)$ au lieu de $K$. Par exemple, l'attaquant envoie une requête de chiffrement (modèle à clair choisi) pour le message $m$, ainsi qu'une fonction $h$, et l'oracle répondra donc avec $E_{h(K)}(m)$. Nous donnons une illustration de ce modèle dans la Figure 2.3.

Il est important de noter que l'oracle *ne révèle pas* $h(K)$ à l'attaquant. Ceci donne beaucoup plus de pouvoir à l'attaquant, et amena à une attaque pratique sur la version complète de KASUMI [60], ainsi que des attaques théoriques sur HIGHT [91] et même sur AES-192 et AES-256 [24]. Le choix de la fonction $h$ accorde plus ou moins de pouvoir à l'attaquant, le cas le plus commun étant l'ajout d'une différence, i.e. $h(x) = x \oplus \Delta$, où $\Delta$ est choisi par l'attaquant. À EUROCRYPT'03, Bellare et Kohno [11] ont donné une meilleure idée des possibilités pour le choix de la fonction $h$.

En particulier, ils fournissent des classes de fonctions telles que pour n'importe quel chiffrement par bloc $E$, si $h$ est autorisé à appartenir à l'une de ces classes, il existe toujours un distingueur (voir Section 3) pour $E$. Dans le Chapitre 3, nous décrivons comment on pourrait concevoir le cadencement de clé d'un chiffrement par bloc itéré pour améliorer sa résistance contre certaines attaques en clé reliées.

## 2.2 Une Histoire de Boîtes

Comme décrit dans les sections précédentes, l'oracle est essentiellement vu comme une *boîte noire*. C'est-à-dire que l'oracle va réaliser les différentes opérations demandées sans donner aucune information à l'attaquant à part le résultat de sa requête (en résumé, un ensemble de clairs et/ou chiffrés). Cependant en pratique, les chiffrements par bloc ne sont pas implémentés dans un tel scénario idéal. En effet, plusieurs informations peuvent fuir lors de l'exécution de la fonction de chiffrement, comme des différences dans le temps d'exécution, ou bien depuis le matériel, comme la consommation de courant, des radiations électro-magnétiques ou bien les accès au cache de données. Ceci amena à considérer un nouveau modèle, appelé le modèle en *boîte grise*, créant ainsi les *attaques par canaux auxiliaires*. Essentiellement, l'idée est de trouver une corrélation entre certains calculs dépendants de la clé pendant l'exécution de la fonction de chiffrement, et des phénomènes physiques observables de l'extérieur. Des implémentations non protégées peuvent être extrêmement sensibles à ce type d'attaque, amenant même parfois à la récupération de la clé utilisée [90]. En plus de simplement observer certaines informations données par le matériel, il y a également la possibilité de réaliser des *attaques par injection de faute*. En utilisant par exemple un laser sur le circuit matériel, un attaquant peut injecter des fautes durant l'exécution de la fonction de chiffrement, donnant par exemple lieu à des bits changeant d'état ou même à la non-exécution de certaines instructions, observer comment la sortie est modifiée et essayer d'en déduire de l'information sur la clé. Cette technique est également très puissante et peut également amener à des attaques *pratiques* désastreuses, comme par exemple les attaques par injection de faute sur AES [70]. Plusieurs contre-mesures ont été étudiées pour protéger ou minimiser l'impact de telles attaques, comme le masquage, ou bien une élaboration spécifique du circuit de calcul.

Cependant, il est possible d'aller encore plus loin et de considérer le modèle en *boîte blanche*. Cette fois-ci, on suppose que l'attaquant a un *accès total* à l'implémentation de l'oracle. Il est autorisé à l'exécuter comme il le souhaite, même partiellement, en modifiant des valeurs dans le code source, passer des instructions etc. Bien que ce modèle donne un pouvoir énorme à l'attaquant, il n'est pas complètement irréaliste, comme par exemple dans la gestion des droits digitaux (DRM). Le but de la cryptographie en boîte blanche est donc de fournir une implémentation logicielle sûre face à ce type d'attaquant. La première proposition pour une telle implémentation a été faite par Chow et al. à SAC'02 [42]. Bien qu'elle fût rapidement cassée [77, 20], elle amena à un nouveau domaine de recherche. Dans le Chapitre 4, nous donnons plus d'informations sur ce modèle, ainsi qu'une nouvelle attaque générique sur n'importe quelle implémentation en boîte blanche réalisée dans le *framework* introduit par Chow et al.. Notre travail montre que le problème mathématique sous-jacent est particulièrement faible, et ainsi, construire une implémentation en boîte blanche sûre reste un problème ouvert.

# 3 Résumé en Français

## 3.1 Recherche Efficace de Couche de Diffusion pour les Réseaux de Feistel Généralisés

La construction de type Feistel est une manière classique de concevoir des chiffrements par bloc, qui fût ensuite généralisée de plusieurs manières par Zheng et al. à CRYPTO'89 [147]. L'une d'entre elles, appelée *Feistel de Type 2*, consiste essentiellement en l'application parallèle de plusieurs réseaux de Feistel classiques, suivie d'une permutation. Dans le papier original de Zheng et al., la permutation utilisée était simplement un décalage cyclique, mais il fût proposé plus tard d'utiliser différentes permutations [112, 131], donnant donc lieu aux Réseaux de Feistel Généralisés.

Dans leur article à FSE'10 [131], Suzaki et Minnematsu se sont concentrés sur l'étude d'un critère spécifique appelé le *nombre de tour de diffusion complète* pour choisir la permutation à utiliser. Ce tour de diffusion complète revient essentiellement à trouver le nombre de tours nécessaires pour que chaque bloc du chiffré dépende de chaque bloc du clair. Ils ont été capables de mener une recherche exhaustive et d'évaluer pour ce critère toutes les permutations jusqu'à 16 blocs, et donc de trouver les permutations optimales pour ces paramètres. Ce faisant, ils ont également remarqué que pour ces nombres de blocs, toutes les permutations optimales étaient paire-impaires, c'est-à-dire que l'image d'un nombre pair est impair et vice-versa. Ils ont donc donné une construction générale pour trouver une *bonne* permutation paire-impaire quand le nombre de blocs est une puissance de deux. En particulier, ils ont donné une permutation paire-impaire dont le nombre tours pour atteindre une diffusion complète est de 10 tours pour 32 blocs. Cependant, la borne inférieure pour un tel nombre de blocs est 9, posant donc la question de l'optimalité de cette permutation. À FSE'19, Cauchois et al. [39] ont poursuivi cette étude et ont donné des permutations paires-impaires optimales jusqu'à 26 blocs, sans réussir à résoudre le problème d'optimalité pour 32 blocs.

Dans le Chapitre 1, nous résolvons ce problème ouvert depuis 10 ans en montrant que cette permutation n'est pas optimale, ainsi qu'en donnant des résultats jusqu'à 42 blocs. Plus précisément, nous commençons par donner une nouvelle caractérisation pour qu'une permutation atteigne une diffusion complète en un nombre de tours donné. Cette caractérisation nous permet d'élaborer un algorithme très efficace pour rechercher des permutations paires-impaires optimales de 28 à 42 blocs, s'exécutant en à peu près une heure pour chaque cas en utilisant 72 *threads*. Dans leur article, Suzaki et Minnematsu donnent une borne inférieure pour atteindre la diffusion complète pour un nombre arbitraire de blocs. Pour chacun des nombres de blocs que nous avons considéré, cette borne inférieure est de 9 tours. Ainsi, nous donnons *toutes* les permutations paires-impaires optimales qui atteignent une diffusion complète en 9 tours pour 28, 30, 32 et 36 blocs, montrant donc que la permutation trouvée dans [131] n'était pas optimale. Pour 34, 38, 40 et 42 blocs, notre algorithme nous a permis de montrer qu'il n'existe pas de permutation avec un tour de diffusion de 9, améliorant donc la borne inférieure à 10. Nous avons également trouvé une permutation paire-impaire optimale pour 34 blocs, qui atteint donc la diffusion complète en 10 tours, ainsi que de bons candidats atteignant 11 tours pour les cas restants.

## 3.2 Boîtes-S Linéairement Équivalentes et la *Division Property*

La *division property* a été un sujet de recherche actif dans la communauté de la cryptographie symétrique depuis son introduction par Todo à EUROCRYPT'15 [136], soutenue par le fait qu'elle a été utilisée pour donner la première attaque théorique sur la version complète de MISTY1 [135]. Bien que le premier algorithme donné par Todo pour rechercher des distingueurs basés sur la *division property* n'avait pas une complexité pratique pour les chiffrements par bloc usuels (essen-

tiellement exponentiel en la taille de bloc), ce problème fût rapidement résolu par Xiang et al. à ASIACRYPT'16 [143] où ils ont proposé d'utiliser des modèles de Programmation Linéaire pour étudier la *division property*. Cette technique, ainsi que plusieurs autres par la suite utilisant d'autres types de programmation déclarative comme les solveurs SMT ou SAT [129], ont permi à la communauté de mieux comprendre la *division property*, ainsi que de trouver de nouveaux distingueurs.

Cependant, nous affirmons que l'espace de recherche considéré dans les algorithmes de recherche précédents est incomplet. En effet, nous observons tout d'abord que bien que les boîtes-S linéairement équivalentes se comportent essentiellement de la même manière lorsqu'on considère les attaques différentielles et linéaires, ce n'est pas le cas pour la *division property*. Il se trouve qu'étudier la propagation de la *division property* dépend fortement de la représentation d'un chiffrement par bloc, et choisir la meilleure représentation pour étudier la *division property* est un problème difficile. Dans ce chapitre, nous résolvons partiellement ce problème en cherchant un distingueur sur un chiffrement par bloc linéairement équivalent, i.e. en cherchant un distingueur sur $L_{out} \circ E \circ L_{in}$ au lieu de $E$, où $L_{in}$ et $L_{out}$ sont des applications linéaires. Nous commençons par montrer comment réduire significativement l'espace de recherche, puis donnons un algorithme efficace pour parcourir cet espace réduit. Ainsi, nous avons donné un nouveau distingueur basé sur la *division property* sur 10 tours de RECTANGLE, alors que le meilleur distingueur connu jusque là était sur 9 tours.

Grâce à nos observations, nous donnons également une meilleure compréhension de la conception de boîtes-S pour résister à la *division property*. Plus spécifiquement, nous prouvons que si une boîte-S vérifie une certaine condition, alors elle est optimale quand on considère la résistance à la *division property* classique (i.e. sans considérer notre technique). Bien qu'un critère similaire ait été donné par Boura et al. dans [37], il se concentrait plus sur la conception d'une boîte-S *relativement résistante*, alors que notre critère est prouvé optimal et que ces deux critères sont incompatibles entre eux. Grâce à cela, nous sommes capables de trouver des boîtes-S alternatives pour PRESENT et RECTANGLE telles que la résistance aux distingueurs basés sur la *division property* est améliorée de deux tours.

## 3.3 Variantes du Cadencement de Clé d'AES pour de Meilleures Bornes sur les Différentielles Tronquées

Malgré son rôle central, le cadencement de clé est peut-être l'un des composants le moins bien compris dans les chiffrements par bloc. Si nous avons maintenant une bonne idée de comment construire de bonnes boîtes-S ou de bonnes couches linéaires, la conception du cadencement de clé reste difficile, et plusieurs idées très variées sont utilisées parmi les différents chiffrements par bloc. D'un côté, nous avons par exemple le cadencement de clé d'AES qui est relativement élaboré et compliqué, alors que pour la version 64-bit de LED [73] aucun cadencement de clé n'est utilisé et la clé maître est directement utilisée à chaque tour (et la version 128-bit n'a essentiellement pas de cadencement de clé non plus). De plus, la sécurité dans le modèle à clés reliées est devenue plus étudiée que par le passé, et il semble donc naturel de relier la sécurité dans ce modèle d'un chiffrement par bloc à son cadencement de clé, du moins en partie.

En contribution additionnelle de leur article de FSE'17, Khoo et al. [84] donnent une permutation pour remplacer le cadencement de clé d'AES-128 qui amène à une meilleure sécurité face aux attaques différentielles. Malgré le fait qu'ils aient donné des arguments sur le choix de cette permutation, son optimalité n'est pas considérée. Nous avons donc choisi de nous intéresser de plus près à ce problème, c'est-à-dire construire une permutation pour remplacer le cadencement de clé d'AES-128 afin de maximiser le nombre minimal de boîtes-S actives dans le modèle à clés reliées. Dans ce chapitre, nous commençons par donner des bornes génériques sur le nombre minimal de boîtes-S actives atteignable après un certain nombre de tours. En particulier, peu importe la per-

mutation utilisée, nous montrons que le nombre minimal de boîtes-S actives sur 5 tours est au plus 17. Nous nous concentrons donc sur le cas 6 tours, pour lequel nous donnons une permutation atteignant 20 boîtes-S actives au minimum. Pour ce faire, nous utilisons une combinaison de méta-heuristiques, ainsi qu'un nouveau modèle de Programmation par Contraintes permettant d'avoir une évaluation plus précise du nombre minimal de boîtes-S actives pour une permutation donnée. Nous donnons également des paires de permutations, remplaçant respectivement le cadencement de clé et l'opération `ShiftRows` d'AES-128, qui permettent d'atteindre 21 boîtes-S actives sur 6 tours. Pour finir, pour chaque permutation (resp. paire de permutations) trouvée, nous prouvons grâce à un autre modèle de Programmation par Contraintes [67] qu'il n'existe aucun chemin différentiel sur 6 tours avec une probabilité plus grande que $2^{-128}$.

## 3.4 Sur l'Extraction d'Encodages Affines dans les Implémentations en Boîte Blanche

Depuis le premier candidat pour une implémentation en boîte blanche par Chow et al. en 2002 [42], le problème de créer une implémentation en boîte blanche sûre pour AES persiste. En gardant les mêmes idées fondamentales de la proposition de Chow et al., plusieurs autres constructions virent le jour en restant dans ce même *framework*. Dans celui-ci, la fonction de tour du chiffrement par bloc est "encodée" en la composant avec des couches non-linéaires et affines appelés encodages. Malheureusement, toutes ces tentatives furent rapidement cassées par une suite d'attaques de plus en plus efficaces réussissant à supprimer ces encodages, permettant par la suite de révéler la fonction de tour, et donc la clé secrète. Ces attaques sont cependant dédiées au schéma attaqué et ne sont pas facilement applicables à d'autres propositions.

Dans ce chapitre, nous proposons donc un algorithme générique et efficace permettant de retrouver les encodages affines, pour *n'importe quel* chiffrement par bloc de type Réseau de Permutation-Substitution (SPN), comme AES, ainsi que pour *n'importe quelle* forme d'encodages affines. Plus précisément, étant donné une fonction de tour encodée de la forme $A \circ S \circ B$, où $A$ et $B$ sont des encodages linéaires (ou affines), notre algorithme est capable de trouver $A$ et $B$ (à équivalence près, la solution n'étant pas toujours unique), où $S$ est une couche de boîtes-S connues composées de plusieurs boîtes-S distinctes. En considérant les paramètres pour AES, c'est-à-dire des blocs de 128 bits partagés en 16 boîtes-S de 8 bits parallèles, les encodages affines sont extraits avec une complexité en temps estimée à $2^{32}$ opérations basiques, indépendamment de la manière dont sont construits les encodages. Nous illustrons cet algorithme sur une proposition récente de Baek, Cheon et Hong qui n'était pas analysée jusqu'à présent. Alors que les auteurs évaluent la sécurité de leur schéma à 110 bits, une application directe de notre algorithme générique est capable de casser ce schéma avec une complexité en temps estimée à seulement $2^{35}$ opérations basiques.

En étudiant cette proposition plus en détails, nous donnons une approche différente pour cryptanalyser le schéma de Baek et al., qui ramène l'analyse à un nouveau problème de combinatoire, et permettant au final de récupérer la clé secrète avec une complexité en temps de $2^{31}$. Nous fournissons également une implémentation de cette attaque, qui permet de récupérer la clé secrète en approximativement 12 secondes sur un ordinateur de bureau standard.

# Publications

Derbez, P., Fouque, P., Jean, J., Lambin, B.: Variants of the AES key schedule for better truncated differential bounds. In: Selected Areas in Cryptography - SAC 2018 - 25th International Conference, Calgary, AB, Canada, August 15-17, 2018, Revised Selected Papers. pp. 27–49 (2018). doi: 10.1007/978-3-030-10970-7_2, https://doi.org/10.1007/978-3-030-10970-7_2

Derbez, P., Fouque, P., Lambin, B., Minaud, B.: On recovering affine encodings in white-box implementations. IACR Trans. Cryptogr. Hardw. Embed. Syst. **2018**(3), 121–149 (2018). doi: 10.13154/tches.v2018.i3.121-149, https://doi.org/10.13154/tches.v2018.i3.121-149

Derbez, P., Fouque, P., Lambin, B., Mollimard, V.: Efficient search for optimal diffusion layers of generalized feistel networks. IACR Trans. Symmetric Cryptol. **2019**(2), 218–240 (2019). doi: 10.13154/tosc.v2019.i2.218-240, https://doi.org/10.13154/tosc.v2019.i2.218-240

**In submission at ToSC'19 Issue 4 :**
Derbez, P., Fouque, P., Lambin, B.: Linearly equivalent s-boxes and the division property. IACR Cryptology ePrint Archive **2019**, 97 (2019), https://eprint.iacr.org/2019/097

*"Wisdom is the offspring of Suffering and Time"*

— Izaro, Lord of the Labyrinth

# Introduction

Securing communications has always been a major concern in humanity's history. Let it be political or strategic messages from governments or companies, communications between someone and his bank or insurance company, or even just a student who wants to send a message to one of his friend during class, we always wanted, and needed, to protect our communications from malicious observers. While the first encryption systems were quite basic, like the Caesar Cipher, the complexity of such systems constantly evolved during history, up to the Enigma machine during World War II and completely skyrocketing with the arrival of modern computers.

Despite the revolution created by the discovery of public key, or asymmetric, encryption with RSA in 1977 [114], secret key, or symmetric, cryptography is still a major element in the security of modern communications. The core idea of symmetric cryptography, and the origin of the name, is that the two parties trying to communicate know a common information, the key, that only they know and which is used to encrypt communications. Informally, designing symmetric encryption algorithms comes down to creating a process taking as input the key and the *plaintext* which then output a *ciphertext* that only the legitimate owners of the secret key can decrypt back to the plaintext. There are three main security notions in symmetric cryptography : confidentiality, integrity and authenticity. The goal of confidentiality is to ensure that only the legitimate parties owning the secret key can decrypt the ciphertext. This is done using *encryption*, which is also the main topic of this thesis. Next, integrity ensure that the plaintext/ciphertext has not been tampered during the communication, most of the time this is done using *hash fonctions*. Finally, authenticity allows to guarantee that the ciphertext comes from someone owning the secret key, this time using *Message Authentication Codes* (MAC). It is also possible to create primitives which bundle these three properties together, thus leading to *authenticated encryption*.

Although older than public key cryptography, symmetric cryptography is still an active topic of research. A major step in its history is the standardization of the DES cipher in 1977 [56]. At the beginning of the 70's, the National Bureau of Standards (now called the NIST, National Institute of Standards and Technology) requested a new encryption scheme to create an American standard. The selection process was not made public, but their choice ended up being an algorithm previously designed by Horst Feistel at IBM called Lucifer [64]. However, the National Security Agency (NSA) asked IBM to modify some components in the algorithm called the *S-boxes*. These modifications were made, leading to the DES algorithm we know today. However, the involvement of the NSA in its design was not well received by the community. Especially, when Biham and Shamir discovered differential cryptanalysis [17], they noticed that DES is oddly resistant to this kind of cryptanalysis, which was previously unknown by the public. It was then confirmed by Coppersmith, one of the member of the IBM team, that this technique was indeed already known during the design of the S-boxes [44].

This lack of transparency led to a new way of selecting the next standards. Indeed, the selection process for the successor of DES was made as a public competition, the AES (Advanced Encryption Standard) competition. Initiated by the NIST on January 2, 1997, 15 candidates were submitted and publicly visible, leaving open the security analysis by the whole cryptography community. After reducing it to 5 finalist candidates on August 9, 1999 (Rijndael, MARS, RC6, Serpent and Twofish), the NIST made the announcement that Rijndael has been chosen to be the AES standard as we know it these days. The same process was then used to choose a new standard for hash functions, with the SHA-3 competition starting on January 23, 2007, and leading to choose Keccak after 5

years of cryptanalysis by the community. The standardization of Keccak also created a new field of research for symmetric cryptography. Indeed, the new construction, called *sponge construction*, used to design Keccak allows one to create both encryption schemes and hash functions from a single permutation, and has been a trending topic of research recently. Other similar competitions were also made, like the eSTREAM competition funded by the European Union, which led to a portfolio of recommended stream ciphers, or the CAESAR competition which ended in early 2019, bringing a portfolio of authenticated encryption schemes. Finally, the last competition to date is about choosing one or several primitives for *lightweight cryptography*, i.e. targeted for constrained devices like RFID. The NIST is funding this competition, with 56 submissions announced on April 18, 2019, and an already very active participation from the community to analyze the different candidates.

Studying the different criteria to design a secure symmetric encryption scheme while still having decent performances and a viable implementation on constraint hardware is thus still a major concern these days, hence the main topic of this thesis, which is to give new insights for the design of new symmetric primitives. We can thus give the following very generic definition, considering that plaintexts and keys are represented as bitstrings of arbitrary length.

**Definition 0.1.** *An encryption scheme is a family of functions* $\mathcal{F} : \mathbb{F}_2^* \times \mathbb{F}_2^* \to \mathbb{F}_2^*$*, where* $\mathbb{F}_2^*$ *is the set of arbitrary length bitstrings, such that for any* $p \in \mathbb{F}_2^*$ *(the plaintext) and* $K \in \mathbb{F}_2^*$ *(the key), and by denoting* $c = \mathcal{F}(K, p)$ *(the ciphertext), then there exist an efficiently computable function* $\mathcal{F}' : \mathbb{F}_2^* \times \mathbb{F}_2^* \to \mathbb{F}_2^*$ *such that* $p = \mathcal{F}'(K, c)$*.*

In practice, the size of the key is often fixed, and this will be the case for the rest of this thesis. One notable exception however is the *one-time pad*. This encryption scheme is extremely simple : given a plaintext $p$ and a key $K$, the ciphertext is defined as $c = K \oplus p$. Despite its simplicity, this scheme is actually the only perfect cipher known up to now, which was proven by Shannon in 1949 [118]. However, it implies some constraints which makes it quite unusable in practice :

- The key length must be *larger or equal* to the plaintext length.
- The key must be *truly random*.
- The key must be used only once, i.e. two plaintexts, even the same, shall not be encrypted using the same key.

It is quite clear that such constraints are really hard to satisfy in practice for a large-scale system, even though it has still been used over the history.

Cryptographers thus tried to design simpler solutions, while trying to keep a good amount of security. In symmetric cryptography, two main families of encryption schemes exist. The first one is the family of *stream ciphers*. The idea behind this kind of cipher is quite simple : being as close as possible to the one-time pad, while not having its constraints. Thus, instead of generating a key of the same length as the plaintext, stream ciphers take a fixed-size key and use it to derive a *keystream* which is then used in the same way as the one-time pad by XOR-ing it to the plaintext. Designing such kind of ciphers comes down to try to make this keystream as close as possible to a random bitstring such that it is close to the one-time pad. Several stream ciphers exist, such as one of the first publicly know, RC4 [124], which is still used these days despite its known flaws, e.g. in the WEP protocol. However, the eSTREAM competition mentioned previously led to a list of much better stream ciphers than RC4 such as Trivium, and one of the finalists of the CAESAR competition, ACORN [141], is also a stream cipher. This thesis however will focus on the second family of ciphers, called *block ciphers*, which are described in the next section.

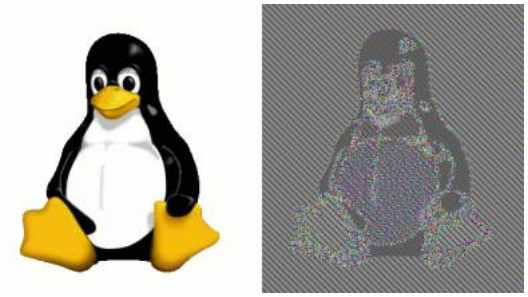

Figure 1.4: The original image (left) and its encryption using ECB (right) [63]

# 1 Block ciphers

We first start with a definition of the main object we study in this thesis.

**Definition 1.1.** *A block cipher is a family of functions* $E : \mathbb{F}_2^k \times \mathbb{F}_2^n \to \mathbb{F}_2^n$ *such that each function* $E_K = E(K, \cdot) : \mathbb{F}_2^n \to \mathbb{F}_2^n$ *defines a permutation. Here,* $k$ *is called the key length and* $n$ *the block length.*

From Definition 0.1, a cipher needs to be able to encrypt plaintexts of arbitrary length, while a block cipher can only manage plaintexts of a fixed length $n$. To do so, we need to use a *mode of operation*, which splits the plaintext into blocks of size $n$ and then uses an algorithm to encrypt everything together. The most basic mode is *Electronic Code Book* (ECB), where each block is encrypted independently. This mode is highly vulnerable as soon as the length of the plaintext is higher than $n$, as two identical plaintext blocks will be encrypted to the same ciphertext block, thus leaking information, see Figure 1.4.

Directly constructing a block cipher is a hard task, both to design and to study the resulting primitive. Hence, most modern block ciphers use the *iterated block cipher* construction, pictured in Figure 1.5. The idea is to design two main components. First, the *key schedule*, which takes as input the key, often called *master key* in that case, and derives a sequence of *round keys* $K_0, \ldots, K_r$, where $r$ is called the *number of rounds*. Then, the *round function* $f$, which is used with the key schedule to build the whole encryption function as

$$E_K = \oplus_{K_r} \circ f \circ \oplus_{K_{r-1}} \circ f \circ \cdots \circ f \circ \oplus_{K_1} \circ f \circ \oplus_{K_0},$$

where $K_0, \ldots, K_r$ are the round keys derived from the master key $K$ and $\oplus_K(x) = x \oplus K$. Essentially, it comes down that designing a strong block cipher directly is much harder than designing a decent, albeit weaker, round function which is then iterated many times. It is also easier to study the behavior of a block cipher designed that way, as we can first focus on the study of the round function, which is simpler than looking at the whole block cipher itself.

In Chapter 3, we give some insights about the design of the key-schedule. Overall, this is still something that is not very well understood by the community, and there is a wide range of different constructions. The current standard AES for example has a quite intricate non-linear key-schedule, the PRESENT one is essentially a linear permutation, followed by a non-linear operation on only 4 bits and the addition of a constant. Some block ciphers only use a linear key-schedule, for example IDEA [93] or Square [45], and some even go further and directly use the master key, only adding a different constant at each round, such as LED [73] or Midori [5]. The design of the round function however is much better understood. There are two main prevalent designs used to build block ciphers :

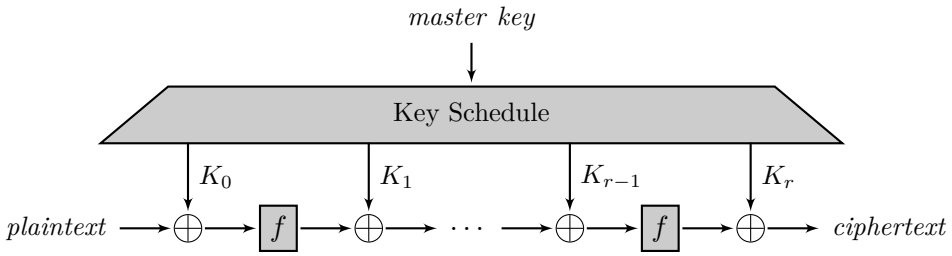


Figure 1.5: Generic iterated cipher construction [78]

**Feistel Networks.** Introduced during the creation of the Data Encryption Standard (DES) [56], Feistel Networks are still used these days with block ciphers such as Camellia [2] or SIMON [8]. The idea is to split the $n$-bit state into two halves of $n/2$ bits. Thus at round $i$, the state is represented as $(L_i, R_i)$ where both $L_i$ and $R_i$ are $n/2$-bit blocks. Then, the round function is defined as

$$(L_{i+1}, R_{i+1}) = (R_i, L_i \oplus F_{K_i}(R_i)),$$

where $F_{K_i}$ is a function depending on the $i$-th round key. Along with some nice theoretical properties (e.g. [100]), this construction is also efficient both in software and hardware, as its inverse is essentially the same function, except that the round keys are used in reverse order. Some generalizations of the Feistel Network were then described by Zheng et al. [147] at CRYPTO'89, leading to the *Type-2 Feistel* construction for example, where the round function is now several parallel Feistels, followed by a permutation. This was for example used to design the block ciphers TWINE [132] and Simpira [71]. In Chapter 1, we go more in-depth about this Type-2 Feistel construction, more precisely about how to choose the permutation.

**Substitution Permutation Networks (SPN).** This more recent design was first proposed for the Square block cipher [45], which then lead to the current standard AES. A vast number of block ciphers were designed this way, such as PRESENT [33], Midori [5] or SKINNY [10]. The idea of this construction is to directly build components giving some *diffusion* and *confusion*. These terms were introduced by Shannon in 1949 [118]. The goal of *confusion* is to make the relation between the plaintext, the ciphertext and the key as complex as possible. Then, *diffusion* should make each bit of the plaintext and each bit of the key to influence many bits of the ciphertext. In this construction, the round function is composed of three steps :

- **The substitution layer**, which goal is to provide some *confusion* by being a non-linear mapping. This step is often built by concatenating several *S-boxes*, which are small (in term of number of bits) non-linear invertible applications, and applying them to all the state. The choice of the S-box is thus crucial, as it is the only non-linear component in the whole round function. In Chapter 2, we give, among other results, a new criterion for choosing those S-boxes. Note that some block ciphers have a more elaborate substitution layer, such as SPARX [58].

- **The permutation layer**, which aims at giving some *diffusion* to the round function. While the name suggests that it could only be a bit-permutation (such as in PRESENT), the construction is more generic and this layer is simply a linear layer. The design of this layer is not less important than the substitution layer. Some block ciphers are designed with a strong linear layer, such as AES, which provides a quick diffusion through the round function. Others, like SKINNY, use a linear mapping with a lower diffusion, but with better performances. This

is often a result of a trade-off between performance and security : a strong linear layer will have high diffusion and thus needs less rounds, but the linear mapping might be costly to implement. On the other hand, a weaker diffusion can often lead to better performances for the round function, at the cost of needing more rounds to achieve the same security.

- **The key addition** consists simply in XORing the round key with the state. It can be done over the whole internal state like in AES, or only partially like in SKINNY.

There is also another class of block ciphers considered aside these two constructions, which is the ARX construction. These block ciphers essentially rely on three main operations : modular Addition, bitwise Rotation and XOR. There is also a variant AndRX, using a bitwise And instead of the modular addition, e.g. SIMON [8]. This construction is a bit on the side, as it is often used in conjunction with one of the two previous designs, only using the ARX operations to build the different sub-function needed. For example, SPARX [58] is a SPN using ARX operations, while HIGHT [76], SPECK [8] or XTEA [140] are Feistel networks using ARX operations (and SIMON which is also a feistel network, but using AndRX operations). Finally, SPNs and Feistels are not the only existing designs, and among others, we can cite the Lai-Massey construction [93], Even-Mansour [62] or the MISTY construction [104].

## 2 Security Models

Studying the security of block ciphers requires to know the attacker capabilities. We often model the interactions between the attacker and the block cipher using the concept of *oracle*. When studying a block cipher $E$, the oracle will first choose a key $K$ at random, which stays secret and does not change, and will answer to the attacker's requests using the corresponding encryption function $E_K$. The attacker will then try to recover $K$ only from the answers of the oracle, or for a weaker attack, decrypt a ciphertext without knowing beforehand the corresponding plaintext. Note that according to the Kerckhoffs' principle, the attacker knows exactly which block cipher is used by the oracle, including how every operation is done (e.g. the S-box, the linear layer etc.). The *only* information kept secret from the attacker is the key.

We now need to consider which requests the attacker is allowed to sent, and this will define the security model. The weaker model is when the attacker is only allowed to ask the oracle for a ciphertext. The oracle will simply generate a random plaintext, *unknown* from the attacker, and returns the corresponding ciphertext. This is called the *known ciphertext* model, as the attacker only knows a set of ciphertexts without their corresponding plaintexts. An attack in this model is devastating, as the constraints for the attacker are very small : simply having access to some (random) ciphertexts. This model is considered very weak, as the attacker has a very minimal power. No serious block cipher is susceptible to such attack these days, but more ancient ones like the Caesar or the Vigenère ciphers can easily be broken is this model. There is however a recent stream cipher which was broken in this model, namely the PC1 cipher used in Amazon's *Kindle* tablet.

A slightly stronger model is the *known plaintext* model. Here, upon request from the attacker, the oracle will again generate a random plaintext and compute its corresponding ciphertext, but this time send *both* the plaintext and the ciphertext to the attacker. Some very well known attacks in this model are the attack on Enigma from Bletchley Park during World War II, or more recently against the block cipher (and its subsequent variants) FEAL [133, 105]. Some modern cryptanalysis techniques use this model, such as linear cryptanalysis [105] or zero-correlation cryptanalysis [34]. Still, this is not enough for other cryptanalysis techniques.

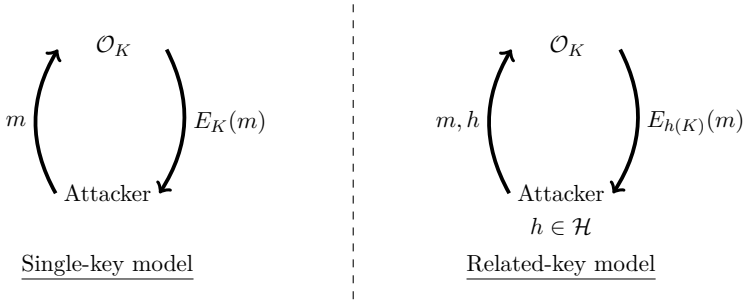


Figure 2.6: Single-key model vs. Related-key model for a chosen-plaintext request.

To analyze modern block ciphers, we often need to be at least in the *chosen plaintext* model. Here, the attacker can choose which plaintext to send to the oracle and get the corresponding ciphertext. This is much more powerful, and leads to cryptanalysis techniques such as, for example, differential or integral cryptanalysis. Several modern block ciphers have attacks on the full number of rounds in this model, for example MISTY1 [135], KLEIN-64 [94], or Robin and Zorro [96]. These attacks were also applied on reduced-round variants of essentially all modern block ciphers. This is the baseline security considered nowadays, and using any cipher susceptible to an attack in this model (or one of the previously described ones) would be a major security mistake.

However, there is also a stronger model, which is the *chosen ciphertext* model. Here, along with requesting the corresponding ciphertext of a plaintext of its choice, the attacker can also ask for the converse, i.e. send a ciphertext to the oracle, which send back the corresponding plaintext. This model leads to attacks such as for example boomerang attacks (which are more specifically in the *adaptive* chosen ciphertext model), which were able to break COCONUT98 with practical complexity [138], as well as differential-linear [95] and yoyo attacks [13].

Finally for the last two models, we can consider an *adaptative* variant, where the attacker sends some request to the oracle, gets the result and can do some others requests based on the result. This is slightly different than the standard models, where the attacker sends a set of request to the oracle once and for all, and is not allowed to interact with it later. Such variants are stronger than their respective standard variants.

We thus described the main models concerning the attacker power. However, there are also some other factors we can consider jointly with the previous models, which are described in the next sections.

## 2.1 Are you single ?

The security models previously described were historically made in the *single-key model*. That is, once the oracle chose a random key, this key would not change and the attacker has no power over it. However, at EUROCRYPT'93 [12], Biham described a new kind of attack, leading to the *related-key model*. In this model, the oracle still chooses a random key $K$ kept secret from the attacker. However, when the attacker sends a request to the oracle, he can also send along a function $h$ from a class of functions $\mathcal{H}$, and the oracle will answer using the key $h(K)$ instead of $K$. For example, the attacker sends an encryption request (chosen-plaintext model) for the message $m$, along with the function $h$, and the oracle will thus answer with $E_{h(K)}(m)$. We give an illustration of this model in Figure 2.6.

Note that the oracle does *not* reveal $h(K)$ to the attacker. This gives much more power to the attacker, and it leads to successful practical cryptanalysis of the full version of KASUMI [60], as well

as theoretical cryptanalysis of HIGHT [91] and even AES-192 and AES-256 [24]. The choice of the function $h$ can lead to a variable power bestowed to the attacker, the most common case being to add a difference, i.e. $h(x) = x \oplus \Delta$, where $\Delta$ is chosen by the attacker. At EUROCRYPT'03, Bellare and Kohno [11] gave more insights about the possibilities available for the function $h$. Especially, they give some classes of functions such that for any block cipher $E$, if $h$ is allowed to belong to such a class, there is always a distinguisher (see Section 3) against $E$. In Chapter 3, we will see how we could design the key-schedule of an iterated block cipher to have a better resistance against some types of related-key attacks.

### 2.2 A Box Story

As described in the previous sections, the oracle is essentially seen as a *black-box*. That is, the oracle will do the required computation without giving any information to the attacker except the result of his request (essentially, some plaintexts and/or ciphertexts). However in practice, block ciphers are not implemented in such an ideal scenario. Indeed, several information can leak from the execution of the encryption function itself, such a difference in timings, or from the hardware, e.g. power consumption, electromagnetic radiations or cache accesses. It led to consider a new model, called the *gray-box* model, and creating *side-channel attacks*. Essentially, the idea is to try to find a correlation between some key-dependent computations made during the execution of the encryption function, and some observable physical phenomena. Unprotected implementations can possibly be extremely susceptible to such attacks, sometimes even leading to a key-recovery [90]. Beside only observing some external information leaked by the hardware, there is also the possibility of *fault-injection attacks*. Using e.g. a laser on the hardware circuit, the attacker can inject faults during the execution process, leading to bit flips or even instructions skipping for example, see how the output is modified and try to deduce some key information. This is also very powerful and can also lead to devastating *practical* attacks, see for example [70] for fault injection attacks on AES. Several countermeasures have been studied to protect implementations from such side-channel attacks, such as masking.

However, we can go even further and consider the *white-box* model. Here, the assumption becomes that the attacker has *full access* to the implementation of the oracle. He is allowed to execute it however he wants, let it be only partially, by modifying values in the source code, skipping instructions etc. While this model gives a large amount of power to the attacker, it is not unreasonable, for example when thinking about Digital Rights Management. The goal of white-box cryptography is thus to provide a secure software implementation in a hostile environment. The first proposal for an implementation in this model was made by Chow et al. at SAC'02 [42]. While being quickly broken [77, 20], it led to a new field of research. In Chapter 4, we give more information about the white-box model, as well as some new generic attack on any white-box implementation made in the framework introduced by Chow et al.. Our work shows that the underlying mathematical problem turns out to be quite weak, and thus building a good white-box implementation still remains an open problem.

## 3 Distinguishers for Block Ciphers

Most of the time when cryptanalyzing a block cipher, we first need to build a *distinguisher* for this block cipher. Informally, a distinguisher is an algorithm which is able to decide (with a certain probability to be right) if a set of plaintexts/ciphertexts comes from a random function or a block cipher. To find such a distinguisher, we have to look for behavior from the block cipher which happens with a significantly different probability than what it would be if it was a random function.

For example with differential distinguishers (detailed in the next section), the relation $f(x \oplus \texttt{0x13}) \oplus f(x) = \texttt{0x37}, x \in \mathbb{F}_2^n$ should hold with probability $2^{-(n-1)}$ for a random function. However, if we are able to show that this relation holds with a much higher probability for a given block cipher, then this can result in a distinguisher.

Once we have a distinguisher for a block cipher, we can mount a generic attack on the last round. Assume that the distinguisher covers $r-1$ rounds of the block cipher and that the last round key is shorter than the master key (e.g. in Feistel Networks). The attacker asks for a set of ciphertexts, then makes a guess on the last round key to partially decrypt these ciphertexts. He thus has a set of ciphertexts coming from the encryption through $r-1$ rounds of the block cipher, and can thus applies the distinguisher. The main hypothesis made is that if the guess is wrong, then the resulting set of ciphertexts will behave as if it was produced by a random function. As such, if the distinguisher outputs that the set comes from a random function, then the key will be assumed to be incorrect. Otherwise, it will be assumed to be correct, and then the attacker can repeat the attack to incrementally decrypt the whole set of ciphertexts by recovering each round key (which usually at some point ends up being equivalent to recovering the master key).

In the sections below, we give a short introduction to different types of distinguisher mentioned in this thesis. Note that several other types of cryptanalysis exist, like linear, algebraic, boomerang or Meet-in-the-Middle attacks.

## 3.1 Differential Distinguishers

Introduced in the early 90s by Biham and Shamir at CRYPTO'90 [17], differential cryptanalysis quickly became one of the most prevalent way to attack block ciphers. It was used to mount the first attack on the full DES [19], as well as to attack FEAL [18], PUFFIN [30] and Zorro [72], and many reduced-round variants of block ciphers. Essentially the question is, given two plaintexts $p_0, p_1$ with difference $\Delta_p = p_0 \oplus p_1$ chosen by the attacker, is it possible to predict, with a good probability and independently from the key, the resulting difference on the corresponding ciphertext ? This is indeed a first order differential, as we are trying to predict the value of $\Delta_c = E_k(p_0) \oplus E_k(p_0 \oplus \Delta_p)$. Note that for a random $n$-bit permutation, the probability of such a differential is expected to be about $2^{-n}$. Thus, if a block cipher has a differential with a probability significantly higher than this, it would be possible to get a distinguisher for this block cipher.

Evaluating the probability of such a *differential* $(\Delta_p, \Delta_c)$ is however a challenging task, especially since we want to find a differential with a high enough probability. We thus usually consider *differential characteristics*, which essentially give the propagation of the input differential through the different round functions composing the block cipher. For a $r$-round block cipher, a differential characteristic is a sequence $\Delta_0, \ldots, \Delta_r$ with $\Delta_0 = \Delta_p$ and $\Delta_r = \Delta_c$ such that the difference $\Delta_i$ propagates to $\Delta_{i+1}$ through the $i$-th round function with a given probability $p_i$. Assuming independence between the different rounds[1], the probability of the characteristic is given by

$$\mathcal{P}(\Delta_0, \ldots, \Delta_r) = \prod_{i=0}^{r-1} p_i.$$

Evaluating the exact probability of this differential would require to compute *all* differential characteristics starting with $\Delta_0$ and ending with $\Delta_r$, which is often impractical. There are still some works to handle a subset of all characteristics, see for example [75]. Most of the time however, we focus on finding a characteristic with a reasonably high probability, without necessarily considering

[1] while this does not actually hold in practice, it still gives a decent approximation most of the time

this clustering effect. Indeed, it is commonly assumed that the best characteristic gives a good approximation of the actual probability of the corresponding differential.

To study differential characteristics, we observe that most of the time, the different steps composing a round function are actually much easier to study than the round function itself as a whole. Indeed, if we take the SPN construction for example, the propagation of differences through the linear layer and the key-addition is predictable with probability 1, as those are linear (resp. affine) operations. The only operation where the propagation becomes probabilistic is the S-box layer. However, the S-box layer itself is composed of several S-boxes, which are much smaller functions. For an S-box $S$, we can thus compute the exact probability of all differentials $(\Delta_{in}, \Delta_{out})$, leading to the *differential distribution table* (DDT). This DDT is a double entry table such that $\mathrm{DDT}(\Delta_{in}, \Delta_{out})$ is the probability that $S(x) \oplus S(x \oplus \Delta_{in}) = \Delta_{out}$ holds over the whole input space of the S-box (i.e. over all $x \in \mathbb{F}_2^m$, where $m$ is the size of the S-box).

### Truncated Differentials

Searching for such differential characteristics however still remains a challenging task in some cases. We thus often also consider *truncated differential characteristics*, where we do not study the exact differential propagation, but only whether some part of the state has a non-zero difference or not. For example for AES, each operation of the round function is defined over the bytes of the state. Thus, we can only study whether a given byte is *active* (non-zero difference) or *inactive* (zero difference). While these truncated differential characteristics can be used themselves to find distinguishers, they can also be used to help finding exact differential characteristics [69]. Moreover, truncated differential characteristics can be used to give an approximation of the security of a block cipher against differential cryptanalysis.

As said in the previous section, the only component where the propagation is non-deterministic is the S-box layer. For a given truncated characteristic, we can thus count the number of *active S-boxes*, i.e. S-boxes with a non-zero difference at the input/output. While we do not know the exact difference, from the DDT of the S-box we can get the *maximal differential probability*, i.e. the probability $p_{max}$ which is maximal over all the probabilities in the DDT (except from the zero transition to itself, which is always 1). Hence, if a given truncated characteristic has $n_S$ active S-boxes, we know that any differential characteristic matching this truncated characteristic would have at best a probability of $p_{max}^{n_S}$. Thus, to get a decent approximation of the resistance against differential cryptanalysis, we can first try to find the *minimal number of active S-boxes* $n_{min}$, allowing us to know that any differential characteristic would have a probability of at most $p_{max}^{n_{min}}$. If this probability is lower than $2^{-n}$, then we can make the safe assumption that the cipher seems resistant to differential cryptanalysis. Note that this does not mean that the tools introduced by differential cryptanalysis would be useless to attack this block cipher, for example we will see in the next section that differentials with probability 0 can also be exploited. Boomerang attacks [138] are also a type of attack using techniques from differential cryptanalysis where the minimal number of active S-boxes is not enough to prove security.

### Searching for (truncated) differentials

Several techniques to search for (truncated) differential trails and the minimal number of active S-boxes were designed over the years. One of the first work to study this extensively is the Wide Trail Strategy by Daemen and Rijmen [47]. This strategy to design the round function of an SPN block cipher was notably used to create the current standard AES. Essentially, the idea is that if the round function is properly built, one can easily prove a lower bound on the number of active

S-boxes for a few rounds. For example in AES, this allowed them to prove that 4 rounds of AES have at least 25 S-boxes over 4 rounds [46]. Since the maximal differential probability of the AES S-box is $2^{-6}$, this leads to a differential characteristic of probability at most $2^{-150}$ over 4 rounds. Considering that the number of rounds for AES goes from 10 for the 128-bit key variant up to 14 rounds for the 256-bit key variant, it is safe to assume that AES is resistant against basic differential cryptanalysis.

Several works [82, 120, 32] also provided some similar results for different kind of Feistel constructions, but the first automatic search algorithm is given by Mouha et al. in 2011 [109]. By giving a way to modelize the propagation of truncated differences through the block cipher operations into linear inequalities, this allows to use a *Mixed Linear Integer Programming* (MILP) solver to automatically search the minimal number of active S-boxes. This technique is very general and can be applied to most block ciphers, and requires a relatively low amount of programming effort to get a result. However, the search for truncated differentials and the minimal number of active S-boxes becomes more complicated when considering the related-key model. Nonetheless, some tools were designed to tackle this problem, for example Fouque et al. at CRYPTO'13 [65] gave a generic algorithm for SPN block ciphers, especially AES-like block ciphers. Even though its complexity can be improved when considering that we are specifically studying AES, it still requires a high amount of memory. At CP'16, Gérault et al. [69] described how to build a *Constraint Programming* (CP) model to search for truncated characteristics in the related-key model, as well as showing that computing valid truncated characteristics can then be used to find actual differential characteristics with another CP model. Constraint programming was also used in [68] to improve several related-key attacks on AES, and later in [130] to show improved results on PRESENT, SKINNY and HIGHT.

#### Impossible Differentials

While the previous paragraph focused on differentials with high probability, it actually turns out that differentials with probability zero can also be used to build distinguishers. The idea was first given by Knudsen [87] when he designed the DEAL block cipher and exhibited an attack exploiting a differential with probability zero. The term *Impossible Differential* was then coined by Biham et al. at EUROCRYPT'99 [14] when it was used to attack 31 rounds (over 32) of the block cipher Skipjack. This leads to attacks on several block ciphers, like IDEA and Khufu [16] or Crypton [102]. As for differentials, impossible differentials have led to different techniques to search for them. One of them called the $\mathcal{U}$-method was introduced by Jongsung et al. [85], later improved by Yiyuan et al. [101][2] and later by Shengbao and Mingsheng [142]. These techniques, while already powerful, were only able to search for word-wise impossible differentials, i.e. could not always manage to find very fine-grained contradiction propriety. This was then improved by Sasaki and Todo at EUROCRYPT'17 [115] using, again, MILP. This tool is able to detect contradiction properties that previous algorithms could not, for example exhibiting contradictions at a bit-level on S-boxes differentials. In the end, it allowed to improve previously known impossible differential distinguishers for several block ciphers, such as Midori128, Lilliput and Minalpher.

### 3.2 Integral Distinguishers

In this section, we regroup three kinds of distinguishers that, while using different techniques, end up achieving the same goal. Namely, we will quickly describe high-order differential, structural integral and division property distinguishers.

[2]Note that the paper was originally published on eprint in 2009 at https://eprint.iacr.org/2009/627

### High-order Differentials

The generic goal of integral distinguishers is to be able to predict, with probability 1, the value of the sum of some bits in the encryption of a set of plaintexts. More precisely, given a set of plaintexts $\mathcal{P} = \{p_1, \ldots, p_n\}$ and the corresponding set of ciphertexts $\mathcal{C} = \{c_1, \ldots, c_n\}$ with $c_i = E_k(p_i)$, and by denoting $c_i^j$ the $j$-th bit of the ciphertext $c_i$, the goal is to predict the value of

$$\bigoplus_{c_i \in \mathcal{C}} c_i^j$$

for a well chosen value (or several of them) of $j$. This idea was first given by Lai in 1994 [92] when he described some properties of high-order differentials of boolean functions. Especially, from Section 2 in this paper, we have the following definition and proposition, adapted for vectorial boolean functions.

**Definition 3.1.** *For a function $f : \mathbb{F}_2^n \to \mathbb{F}_2^m$, the derivative of $f$ at the point $a \in \mathbb{F}_2^n$ is defined as*

$$\Delta_a f(x) = f(x \oplus a) \oplus f(x).$$

*The $i$-th order derivative of $f$ at $(a_1, \ldots, a_i)$ with $a_j \in \mathbb{F}_2^n$ for all $j \in \{1, \ldots, i\}$ is defined as*

$$\Delta^{(i)}_{a_1,\ldots,a_i} f(x) = \Delta_{a_i}(\Delta^{(i-1)}_{a_1,\ldots,a_{i-1}} f(x))$$

**Proposition 3.2.** *Let $L[a_1, \ldots, a_i]$ denote the set of all $2^i$ possible linear combinations of $a_1, \ldots, a_i$, then*

$$\Delta^{(i)}_{a_1,\ldots,a_i} f(x) = \bigoplus_{c \in L[a_1,\ldots,a_i]} f(x \oplus c).$$

Thus, we can see that especially, if $f$ corresponds to one of the components of an encryption function and $(p_1, \ldots, p_i)$ is a set of linearly independent plaintexts, the $i$-th order derivative at $(p_1, \ldots, p_i)$ corresponds exactly to the sum of all of the $2^i$ corresponding ciphertexts. As for the derivatives of real functions, Lai also showed that a first order derivative lowers the degree of the function by at least 1. Thus, if $f$ is of degree at most $i$, the $i$-th order derivative will be constant, and finally the $(i+1)$-th order derivative will always be the zero function. This then gives us our first kind of distinguisher, known as *high-order differential distinguisher*. Indeed, if we are able to show that the component $E_k^j$ of an encryption function $E_k$ is of degree at most $i$, then by taking $i+1$ linear independent plaintexts and the set $\mathcal{P}$ of their $2^{i+1}$ linear combinations, we know that for all $j$, we have

$$\bigoplus_{p \in \mathcal{P}} E_k^j(p) = \bigoplus_{c \in \mathcal{C}} c^j = 0$$

with probability one, where $\mathcal{C}$ is the corresponding set of ciphertexts. Of course, this assumes that we are able to generate $i+1$ linearly independent plaintexts. In particular, if $E_k$ is an $n$-bit encryption invertible function, we know that the maximal degree of any of its component is $n-1$. Using this directly would need to generate the set of *all* of the $2^n$ plaintexts, thus not leading to a distinguisher, since the zero-sum property would also be true for a random $n$-bit invertible function. However, if we are able to show that at least one specific component $E_k^j$ is of degree strictly lower than $n-1$, then we are able to get a distinguisher.

Knudsen presented one of the first way to exploit high-order differentials to mount a generic attack on a 5-round Feistel [88]. Several high-order differential attacks were also published against reduced-round variants of different block ciphers, for example CLEFIA [119], CAST [108], Camellia

[126], MISTY1 [6] or KASUMI [125]. Finally, some work was done to get a better bound on the degree of the components of a block cipher. Indeed, if the round function is of degree $d$, then using $r$ rounds, the degree $d'$ of the block cipher is trivially upper bounded by $d^r$, i.e. $d' \leq d^r$. However, it was shown in [36] that this approximation is, in some cases, quite far from the reality. Especially, for the block cipher Rijndael-256, while the trivial bound could lead to think that 3 rounds are sufficient to reach full degree, Boura and Canteaut actually show that it needs at least 7 rounds to achieve maximal degree.

### Structural Integrals

High-order differentials focus on just evaluating the degree of (at least) one component of the encryption function. Thus, assuming this component is of degree $d < n-1$, *any* set of $d+1$ linearly independent plaintexts can be used to generate the set $\mathcal{P}$ of $2^{d+1}$ plaintexts leading to the zero-sum property. This is very general and does not seem to exploit the structure of the block cipher as much as it could. At FSE'97, Daemen et al. proposed the block cipher Square, as well as exhibiting a new attack which is essentially the first *Structural Integral attack* [3]. The idea is that when setting a byte of the plaintext to take all possible values and the others to be constant, thus generating a set of $2^8$ plaintexts, one can show that after 3 rounds of encryption, every bit will have the zero-sum property.

Rather than tracking the degree, the idea here is to track the structure of the set of plaintexts when going through each operation of the round function. This leads to a distinguisher over 3 rounds, which can be extended to 4 rounds and then result in an attack on at most 6 rounds of the block cipher. Since AES has a structure very close to Square, essentially the same attack can be applied [46], and also over 6 rounds of Crypton [57] with small modifications. Knudsen and Wagner also gave a more formal description of such distinguishers at FSE'02 [89] as well as new distinguishers over MISTY1 and MISTY2. Searching for such distinguishers is often done with a basic algorithm where one first finds a distinguisher with a single word taking all possible values (a word often being of the size of an S-box), and then try to extend the distinguisher backward. For example this was used by the designers to find a distinguisher over 10 rounds of SKINNY [10]. Note that in some case, we can have some upper bound on the maximal number of rounds on which we can build a structural integral distinguisher, as in [131] where Suzaki and Minematsu were able to tie a notion called *diffusion round* to this maximal number of rounds[4] for a specific type of block cipher called *Generalized Feistel Networks*. There will be more details about this diffusion round in Chapter 1.

### Division Property

Generalizing both high-order differentials and structural integral, Todo introduced the *Division Property* at EUROCRYPT'15 [136]. This is a much more fine grained property than an upper bound on the degree of the components of the block cipher (as in high-order differentials), but also takes some ideas in structural integrals. Essentially, the idea is that some component may be of a lower degree when considering only a subset of the input variables (i.e. the plaintext variables) and this can be used in the same way as in the previous sections. Let $f$ be one of the component of the encryption function, and let $\mathcal{P}$ be a set of plaintexts such that $c$ bits are always constant, and the $n-c$ remaining bits take all values through the set. For example, the set $\{1000, 1010, 1100, 1110\}$

[3] Also often called "Square Attack".

[4] As well a for impossible differentials.

has the first and last bits constant, while the other two take all 4 possibles values. Note that such a set is essentially an affine subspace of dimension $n - c$. Now let consider $f$ when restricting its input to such a set. Since there are now $c$ constant bits, the maximal degree of $f$ can only be as high as $n - c$, as there are now only $n - c$ variables. Thus, for well chosen positions for constant bits, we may be able to show that $f$, when restricted to this set, actually has a degree strictly lower than $n - c$. This essentially comes down to showing that the (only) monomial of degree $n - c$ does not appear in some component of the block cipher.

Division property is thus essentially a technique to track which monomials may or may not appear after the different operations of the round function. When introducing the division property, Todo also gave very generic attacks for AES-like and Feistel block ciphers, as well as how to propagate this property through some operations. However, the most powerful form of division property, called *Bit-Based Division Property*, is introduced by Todo and Morii at FSE'16 [137]. While the original division property was still very high-level and generic, bit-based division property works on a much lower level (hence the name) and tracks much more precisely the different monomials appearing in the encryption function. Along with a precise way to propagate bit-based division property through basic operations of block ciphers, they also showed improved attacks over the 32-bit variant of the SIMON block cipher. However, the proposed search algorithm essentially has a complexity of $\mathcal{O}(2^n)$, thus impractical for standard (64 or 128 bits) block cipher sizes. This was widely improved later by using some technique based on either MILP [127, 143, 146] or SAT/SMT solver [61, 129], thus greatly facilitating the search for such distinguishers. As it is the focus of one specific chapter, more details about the division property can be found in Chapter 2.

## 4 About Affine Equivalent Permutations

Finally, we quickly describe here a notion which is used several times in this thesis, which are *affine equivalent permutations*.

**Definition 4.1.** *Two permutations $f, g : \mathbb{F}_2^n \to \mathbb{F}_2^n$ are said to be affine equivalent if and only if there exists two affine mappings $A, B$ such that $g = B \circ f \circ A$.*

This notion is very useful in cryptography as it often (but not always) reduces the cases we have to consider for e.g. designing an S-box. For example, we can see that two affine equivalent S-boxes have essentially the same behavior for differential and linear cryptanalysis. Indeed, given two $m$-bit S-boxes $S$ and $S'$ such that $S' = B \circ S \circ A$, if there is a differential $(\Delta_i, \Delta_o) \in \mathbb{F}_2^{2m}$ such that $S(x) \oplus S(x \oplus \Delta_i) = \Delta_o$ holds with probability $p$, then since $A$ and $B$ are linear and invertible, there is a differential $(\Delta_i', \Delta_o') = (A^{-1}.\Delta_i, B.\Delta_o)$ of the same probability for $S'$. Hence the DDT is essentially the same, and we expect that it should not drastically change the resistance against differential attacks compared to using the original S-box, and the same kind of observations can be made for linear attacks. However, we will see in Chapter 2 that affine equivalence does not always preserve the properties of an S-box, this is for example the case with the division property.

To check if two permutations are affine equivalent, Biryukov et al. [22] gave an algorithm taking as input two permutations $S_1, S_2$ over $\mathbb{F}_2^n$, and outputting whether these permutations are affine equivalent, and if so, also enumerating all pairs of linear mappings $(A, B)$ such that $S_2 = B \circ S_1 \circ A$. This algorithm has a complexity of $\mathcal{O}(n^3 2^{2n})$, but was later improved by Dinur [59] to get it down to $\mathcal{O}(n^3 2^n)$. Note that these algorithms are at the center of our work in Chapter 4.

It is also worth noting that there are several other types of equivalence relations for S-boxes. One can of course consider sub-cases of the affine equivalence, such that linear equivalence, where $A$ and $B$ are constrained to be only linear, or even permutation equivalence, where $A$ and $B$ are

now linear permutations. This can help when a property is not invariant through affine equivalence, such that the division property which is however invariant under permutation equivalence. On the other hand, one can consider more evolved equivalence notion, like the CCZ-equivalence [38] which also preserves some differential properties of S-boxes.

# 5 Overview of this thesis

We now give a short overview of each chapter of this thesis. In Chapter 1, we study the permutation step of a Generalized Feistel Network. By exhibiting a new characterization of a specific criterion related to the *diffusion* of such a construction, we are able to find new optimal permutations according to this criterion, thus improving previously known results and solving a 10-year old problem. In Chapter 2, we extend and improve the division property based cryptanalysis, leading to a new attack on RECTANGLE. According to our observation, we give a new criterion to design optimal S-boxes to resist such kind of cryptanalysis, improving the resistance of both PRESENT and RECTANGLE by two rounds. In Chapter 3, we show that we can find a better key-schedule for AES, both in terms of security and performances. We also give new theoretical bounds on the resistance of AES in the related-key model when using a permutation as key-schedule, and we end up finding almost optimal permutations. Finally in Chapter 4, we consider the stronger model known as the white-box model. We first give a generic attack breaking any white-box scheme in the framework introduced by Chow et al. in 2002 with low complexity for standard parameters and with a good scaling. We also show a new dedicated attack on a specific scheme which leads to even better results than our generic attack, validated by an implementation recovering the secret key in about 12 seconds.

## 5.1 Efficient Search for Optimal Diffusion Layers of Generalized Feistel Networks

The Feistel construction is a very well known and standard way to build modern block ciphers, which was then generalized to several constructions by Zheng et al. at CRYPTO'89 [147]. One of them, called *Type-2 Feistel*, is essentially a parallel application of several Feistels, followed by a permutation. In the original paper from Zheng et al., this permutation consisted in a cyclic shift, but it was later proposed to use a different permutation [112, 131], leading to the Generalized Feistel Network construction.

In their paper at FSE'10 [131], Suzaki and Minematsu focused on a specific criterion called *diffusion round* to choose the permutation used. This diffusion round essentially comes down to the number of round needed such that each block of the ciphertext depends on each block of the plaintext. They were then able to exhaust and evaluate all permutations according to this criterion for up to 16 blocks, thus exhibiting optimal permutations. They also noticed that for these numbers of blocks, all optimal permutations were even-odd, meaning that the image of an even-numbered block is an odd-numbered block and vice-versa. Thus, they give a generic construction to find a *good* even-odd permutation when the number of blocks is a power of two. Specifically, they exhibit an even-odd permutation with a diffusion round of 10 for 32 blocks. However, the lower bound for such a number of blocks is 9, hence raising the question of the optimality of this permutation. At FSE'19, Cauchois et al. [39] went further and gave optimal even-odd permutations for up to 26 blocks, still leaving the question about the optimality of a permutation on 32 blocks open.

In Chapter 1, we solve this 10-year old problem by proving that this permutation is not optimal, as well as giving results on up to 42 blocks. More precisely, we first give a new characterization

for a permutation reaching full diffusion in a given number of rounds. This characterization allows us to build a very efficient algorithm to search for optimal even-odd permutations for 28 up to 42 blocks, running in about one hour for each case when using 72 threads. For all of these numbers of block, the lower bound to reach full diffusion is 9 rounds. As a result, we give *all* optimal even-odd permutations reaching full diffusion after 9 rounds for 28, 30, 32 and 36, thus showing that the permutation given by [131] was not an optimal one. For 34, 38, 40 and 42 blocks, our algorithm allowed us to show that there is no permutation with a diffusion round of 9, thus improving the lower bound to 10. We also found an optimal even-odd permutation for 34 blocks, thus reaching 10 rounds for full diffusion, as well as good candidates reaching 11 rounds for the remaining cases.

## 5.2 Linearly Equivalent S-boxes and the Division Property

Division property has been a trending topic in symmetric cryptography since its introduction by Todo at EUROCRYPT'15 [136], and supported by the fact that it has been used to describe the first theoretical attack on full MISTY1 [135]. While the first algorithm described by Todo to search for division property distinguishers was impractical for classical block ciphers (exponential in the block size), this problem was quickly circumvented by Xiang et al. at ASIACRYPT'16 [143] when they proposed to use some *Mixed Integer Linear Programming* (MILP) models to study the division property. This technique, as well as subsequent ones using different kind of declarative programming like SMT-solvers or SAT-solvers [129], allowed the community to better understand the division property, as well as finding new distinguishers.

However, we claim that the search space considered in the previous search algorithms is incomplete. Indeed, we first make the observation that while linearly equivalent S-boxes essentially behave the same when considering differential and linear attacks, this is not the case for division property. It comes down that studying the propagation of division property relies on the *representation* of a block cipher, and choosing the best representation to study division property is a difficult problem. In this chapter, we partially solve this problem by searching a distinguisher over a linearly equivalent block cipher, i.e. searching a distinguisher over $L_{out} \circ E \circ L_{in}$ instead of $E$, where $L_{in}$ and $L_{out}$ are linear mappings. We first describe how to reduce the search space significantly, and then give an efficient algorithm to go through this reduced search space. As a result, we are able to exhibit a new division property based distinguisher over 10 rounds of RECTANGLE, while the best previously known distinguisher was over 9 rounds.

According to our observations, we also give new insights about the design of S-boxes to resist division property. Specifically, we prove that if an S-box satisfies a given condition, then it is optimal w.r.t resistance against classical division property (i.e. without considering our technique). While a criterion was already given by Boura et al. in [37], it was more about building an S-box with a *good enough* resistance, while our is proven optimal, and these two criteria are incompatible with each other. According to this, we are able to find alternative S-boxes for both PRESENT and RECTANGLE such that the resistance against division property based distinguisher is improved by two rounds.

## 5.3 Variants of the AES Key Schedule for Better Truncated Differential Bounds

Albeit being central, the key-schedule may be one of the less understood component in block ciphers. While we have a good idea of how to build good S-boxes or linear layers, designing the key-schedule remains a challenge, and very different designs emerge. For example, on one side we have the AES key-schedule, which is quite complicated, whereas in the 64-bit version of LED [73] no key-schedule

is used and the master key is just used itself at each round (and the 128-bit version essentially has no key-schedule either). Moreover, the related-key model now becomes more prevalent, and it seems natural to tie, at least partially, the security of a block cipher in this model to its key-schedule.

As an auxiliary contribution to their work at FSE'17, Khoo et al. [84] give a permutation to replace the AES-128 key-schedule which leads to better security against differential attacks. Even though they give some arguments about how they chose this permutation, it is not clear whether it was an optimal choice or not. As such, we took a more precise look into this problem, namely, designing a permutation to replace the AES-128 key-schedule which maximize the minimal number of active S-boxes in the related-key model. In this chapter, we begin with generic bounds for the achievable minimal number of S-boxes for a given number of rounds. In particular, whatever permutation is used as key-schedule, we show that the minimal number of active S-boxes over 5 rounds is at most 17. We thus focus our search for 6 rounds of AES-128 , for which we are able to give a permutation reaching 20 active S-boxes over 6 rounds. To do so, we use a combination of meta-heuristics, along with a new Constraint Programming model able to handle more precisely the evaluation of the minimal number of active S-boxes. We also show some pairs of permutations which allow to reach 21 S-boxes over 6 rounds when used to replace, respectively, the key-schedule and the `ShiftRows` operation of AES-128 . Finally, for all permutations (resp. pairs of permutations) found, we prove with another Constraint Programming model from [67] that no characteristic over 6 rounds or more has a probability larger than $2^{-128}$.

## 5.4 On Recovering Affine Encodings in White-Box Implementations

Ever since the first candidate white-box implementation by Chow *et al.* in 2002 [42], producing a secure white-box implementation of AES has remained an enduring challenge. Following the footsteps of the original proposal by Chow *et al.*, other constructions were later built around the same framework. In this framework, the round function of the cipher is "encoded" by composing it with non-linear and affine layers known as encodings. However, all such attempts were broken by a series of increasingly efficient attacks that are able to peel off these encodings, eventually uncovering the underlying round function, and with it the secret key. These attacks, however, were generally ad-hoc and did not enjoy a wide applicability.

In this chapter, we thus propose a generic and efficient algorithm to recover affine encodings, for *any* Substitution-Permutation-Network (SPN) cipher, such as AES, and *any* form of affine encoding. More precisely, given an encoded round function of the form $A \circ S \circ B$, where $A$ and $B$ are linear (or affine) encodings, our algorithm is able to recover $A$ and $B$ (up to equivalence, as the solution may not be unique), where $S$ is a known S-box layer composed of distinct S-boxes. For AES parameters, namely 128-bit blocks split into 16 parallel 8-bit S-boxes, affine encodings are recovered with a time complexity estimated at $2^{32}$ basic operations, independently of how the encodings are built. We illustrate this on a recent proposal due to Baek, Cheon and Hong, which was not previously analyzed. While Baek et al. evaluate the security of their scheme to 110 bits, a direct application of our generic algorithm is able to break the scheme with an estimated time complexity of only $2^{35}$ basic operations.

Going further, we show a different approach to cryptanalyzing the Baek et al. scheme, which reduces the analysis to a standalone combinatorial problem, ultimately achieving key recovery in time complexity $2^{31}$. We also provide an implementation of the attack, which is able to recover the secret key in about 12 seconds on a standard desktop computer.

*"Do. Or do not. There is no try."*

— Master Yoda

# Chapter 1

# Efficient Search for Optimal Diffusion Layers of Generalized Feistel Networks

The Feistel construction is one of the most studied ways of building block ciphers. Several generalizations were then proposed in the literature, leading to the Generalized Feistel Network, where the round function first applies a classical Feistel operation in parallel on an even number of blocks, and then a permutation is applied to this set of blocks. In 2010 at FSE, Suzaki and Minematsu studied the diffusion of such construction, raising the question of how many rounds are required so that each block of the ciphertext depends on all blocks of the plaintext. They thus gave some optimal permutations, with respect to this diffusion criteria, for a Generalized Feistel Network consisting of 2 to 16 blocks, as well as giving a good candidate for 32 blocks. Later at FSE'19, Cauchois et al. went further and were able to propose optimal even-odd permutations for up to 26 blocks.

In this chapter, we complete the literature by building optimal even-odd permutations for 28, 30, 32, 36 blocks which to the best of our knowledge were unknown until now. The main idea behind our constructions and impossibility proof is a new characterization of the total diffusion of a permutation after a given number of rounds. In fact, we propose an efficient algorithm based on this new characterization which constructs all optimal even-odd permutations for the 28, 30, 32, 36 blocks cases and proves a better lower bound for the 34, 38, 40 and 42 blocks cases. Note that our algorithm essentially uses branch-and-bound techniques, and thus it is hard to evaluate the exact complexity. However, the size of the search space goes from $2^{43}$ for 28 blocks up to $2^{75}$ for 42 blocks, but we were able to treat each of these cases in less than one hour each when using 72 threads. Moreover, this characterization has a very efficient implementation which allowed us to re-find all optimal even-odd permutations for up to 26 blocks with a basic exhaustive search in a few hours, showing that for these cases, there is no need for sophisticated techniques as in [39]. In particular, we improve the 32 blocks case by exhibiting optimal even-odd permutations with diffusion round of 9. The existence of such a permutation was an open problem for almost 10 years and the best known permutation in the literature had a diffusion round of 10. Furthermore, for 34, 38, 40 and 42 blocks, we prove with this method that there is no even-odd permutation with a diffusion round of 9, which is the lower bound on the diffusion round for these sizes given in [131]. We were also able to find even-odd permutations with a diffusion round of 10 for 34 blocks (which is thus optimal), as well as even-odd permutations with diffusion round 11 for 38,40 and 42 blocks. Finally, we evaluate the security of our constructed permutations against impossible differentials and differentials (by computing the minimum number of active S-boxes). In particular, for the 32 blocks case, and impossible differentials, all our permutations have a one-round shorter longest impossible differential distinguisher compared to what was proposed by [39], which brings it down to 17 rounds.

## 1.1 Introduction

The Feistel network is one of the main generic designs for building modern block ciphers. It was initially proposed in the data encryption standard DES [56], and is still used in more recent ciphers such as Twofish [117], Camellia [2] or SIMON [8]. The idea behind this construction is to split the plaintext into two halves $x_0, x_1$, and build the round function which sends $(x_0, x_1)$ to $(x_1, x_0 \oplus F_i(x_1))$, where $F_i$ is a non-linear function for the $i$-th round. One of the main advantage of this construction is that $F_i$ does not need to be invertible, and thus it allows to transform a pseudorandom *function* (PRF) into a pseudorandom *permutation* (PRP). Moreover, there are theoretical arguments suggesting that it is a good method to construct block ciphers, as Luby and Rackoff proved in 1988 [100] that if each $F_i$ is a pseudorandom function and all three are independent, then 3 rounds of the Feistel construction are enough to get a block cipher which is indistinguishable from a random permutation under the Chosen Plaintext Attack (CPA) model, and 4 rounds with 4 independent functions are enough in the Chosen Ciphertext Attack (CCA) model. This was later improved by Pieprzyk in 1990 [113] : if one takes $f$ as a pseudorandom function, 4 rounds of Feistel with $F_i = f$ for $i = 1, 2, 3$ and $F_4 = f^2$ are sufficient to obtain a block cipher that is indistinguishable from a random permutation in the CPA model. In 1989 at CRYPTO, Zheng et al. [147] proposed some generalizations of the Feistel construction. Especially, they defined the *Type-2 Feistel*[1] construction, which splits the message into $2k$ blocks and uses a round function of the form

$$(x_0, \ldots, x_{2k-1}) \mapsto (x_{2k-1}, x_0 \oplus F_{i,0}(x_1), x_1, x_2 \oplus F_{i,1}(x_3), x_3, \ldots, x_{2k-2} \oplus F_{i,k-1}(x_{2k-1})),$$

where each $F_{i,j}$ is a pseudorandom function for the $i$-th round. This is essentially a parallel application of $k$ Feistels followed by a cyclic shift of the blocks. They also showed that when all $F_{i,j}$ are pseudorandom functions, then $2k + 1$ rounds of such a construction provide a block cipher that is indistinguishable from a random permutation. Moreover, the Type-2 construction is inherently easier to compute in parallel, and the corresponding decryption function is basically the same except that the functions $F_{i,j}$ are applied in reverse order, i.e. for $r$ rounds, the first round of decryption uses the functions $F_{r,j}$. Both of these properties make this construction very efficient in practice, both on hardware and software, e.g. TWINE [132] and Simpira [71]. All of these arguments lead to some block ciphers based on this Type-2 Feistel construction, such as HIGHT [76] and CLEFIA [122].

At ASIACRYPT'96, Nyberg [112] studied a variant of the Type-2 Feistel construction using a different permutation than the cyclic shift, called Generalized Feistel Network. Such a construction was used to design block ciphers such as TWINE [132] and Piccolo [121]. However, Nyberg only focused on one specific permutation. Suzaki and Minematsu thus studied at FSE'10 [131] a more general case where the cyclic shift is replaced by any other permutation of the blocks. Their work was focused on finding permutations with the lowest *diffusion round*. The diffusion round is close to the concept of *diffusion* introduced by Shannon in 1949 [118]. Essentially, a block cipher has *full diffusion* if every bit of the ciphertext depends on every bit of the plaintext. In the context of Generalized Feistel Network (GFN), [131] defined the diffusion round as the minimal number of rounds such that every *block* of the ciphertext depends on every *block* of the plaintext. Focusing on blocks instead of bits allows them to get rid of the precise specification of the functions $F_{i,j}$ as well as the exact size of the blocks, thus giving structural results. Especially, they tied the diffusion round of a given GFN to its resistance against Impossible Differential distinguishers [15], proving that if a GFN has a diffusion round of $DR$, then it needs strictly more than $2DR + 1$ rounds to

[1] Note that some papers use the term Type-2 Generalized Feistel to denote this construction

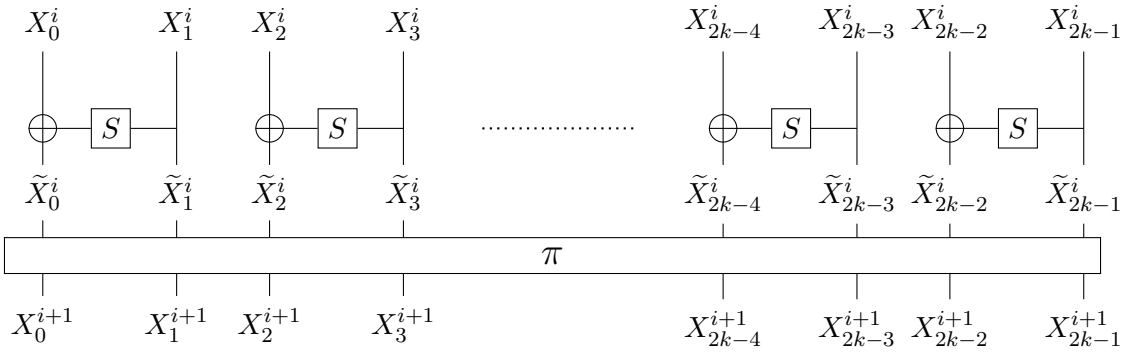


Figure 1.1: Generalized Feistel Network

avoid any Impossible Differential distinguisher. Along with a lower bound on the diffusion round of a GFN of $2k$ blocks, they gave optimal permutations (w.r.t the diffusion round) for $2 \leq 2k \leq 16$. It is worthy to note that such an optimal permutation was then used to design block ciphers such as TWINE [132]. At FSE'19, Cauchois et al. [39] went further and gave optimal permutations for $18 \leq 2k \leq 26$, as well as good candidates for $2k = 32$ (which was already found in [131]), as well as for $2k = 64$ and 128 using a sophisticated technique that they called *Collision-free exhaustive search*. Note that these permutations are even-odd, i.e. the image of an even number is an odd number. On a side note, relaxing the condition that the permutation is the same in each round makes the problem easier and in [81], Kales et al. give such a construction for any number of blocks.

## 1.2 Preliminaries

### 1.2.1 Generalized Feistel Networks (GFN)

Zheng et al. [147] introduced Type-2 Feistels as a generalization of the original Feistel construction. Given an even number $2k$ of blocks $(X_0, \ldots, X_{2k-1})$, it first applies the Feistel construction on the pairs of blocks which yields $(X_0 \oplus S_0(X_1),\ X_1, \ldots,\ X_{2k-2} \oplus S_{k-1}(X_{2k-1}), X_{2k-1})$. The blocks are then cyclically right shifted to obtain the result. Later, it was proposed to use another permutation than the cyclic shift in [112], leading to Generalized Feistel Networks.

**Definition 1.2.1.** *Let $2k$ be an even number, $n, r$ be positive integers, and $\{F_{i,j}\}_{i\in\{1,\ldots,r\},j\in\{0,\ldots,k-1\}}$ be a set of cryptographic keyed functions from $\mathbb{F}_2^n$ to $\mathbb{F}_2^n$. Let $\pi$ be a permutation over $2k$ elements. A Generalized Feistel Network (GFN) is a block cipher built as $\mathcal{R}_r \circ \cdots \circ \mathcal{R}_1$, where $\mathcal{R}_i$ is the round function*

$$\mathcal{R}_i :\ (X_0, \ldots, X_{2k-1}) \to \pi(X_0 \oplus F_{i,0}(X_1),\ X_1, \ldots,\ X_{2k-2} \oplus F_{i,k-1}(X_{2k-1}), X_{2k-1})$$

Note that for this chapter, neither the exact definition of the keyed functions $F_{i,j}$ nor their sizes are relevant. We can thus consider all of them as an arbitrary S-box $S$, leading to the framework depicted in Figure 1.1[2]. As the only variable parameters are thus $k$ and $\pi$, we denote by $GFN_\pi^k$ a GFN with $2k$ blocks that uses the permutation $\pi$.

[2]In practice, one should carefully study the primitive if the same F-function is used, e.g. [71]

### 1.2.2 Diffusion Round

We use the notations depicted in Figure 1.1. The input variables of the $i$-th round of a GFN are denoted by $(X_0^i, X_1^i, \ldots, X_{2k-1}^i)$. We also denote by $(\widetilde{X}_0^i, \widetilde{X}_1^i, \ldots, \widetilde{X}_{2k-1}^i)$ the variables which are at the input of the permutation $\pi$, i.e.

$$(X_0^{i+1}, X_1^{i+1}, \ldots, X_{2k-1}^{i+1}) = \pi(\widetilde{X}_0^i, \widetilde{X}_1^i, \ldots, \widetilde{X}_{2k-1}^i)$$

It is easy to see from Definition 1.2.1 that $X_{\pi(0)}^1$ depends on $X_0^0$ and $X_1^0$. More generally, any block $\widetilde{X}_j^r$ depends on a certain number of blocks from the round 0, i.e. computing $\widetilde{X}_j^r$ requires some blocks $\{X_{j_0}^0, \ldots, X_{j_l}^0\}$. Note that this does not depend on the size of the functions $F_{i,j}$ in the GFN. As in [131], we say in that case that any of these $X_{j_i}^0$ *diffuses* to $\widetilde{X}_j^r$, and we focus our study on the number of rounds needed to reach *full diffusion*.

**Definition 1.2.2.** *Let $\pi$ be a permutation over $2k$ elements. We say that a block $X_j^0$ fully diffuses after $r$ rounds if for all $i \in \{0, \ldots, 2k-1\}$, $X_j^0$ diffuses to $\widetilde{X}_i^r$. We say that $\pi$ reaches full diffusion after $r$ rounds if for all $j \in \{0, \ldots, 2k-1\}$, $X_j^0$ fully diffuses after $r$ rounds. The smallest $r$ that verifies this property for the block $X_i^0$ is called the diffusion round of the block $X_i^0$.*

Note that we need to study both the diffusion over the encryption *and* the decryption process. Indeed, there is no guarantee that an encryption function with good diffusion also keeps this property for its inverse. Since we have $(GFN_\pi^k)^{-1} = GFN_{\pi^{-1}}^k$, we need to study both the diffusion of $\pi$ and $\pi^{-1}$. Naturally, we would like both $\pi$ and $\pi^{-1}$ to fully diffuse as quickly as possible, which leads to the following definition.

**Definition 1.2.3.** *Let $\pi$ be a permutation over $2k$ elements. Denote by $DR_i(\pi)$ the minimum number of rounds $r$ such that $X_i^0$ fully diffuses after $r$ rounds in $GFN_\pi^k$.*

*The diffusion round of a permutation $\pi$ is:*

$$DR_{max}(\pi) = \max_{0 \leq i \leq 2k-1} \left\{ DR_i(\pi), DR_i(\pi^{-1}) \right\} \tag{1.1}$$

This definition gives the same importance to the total diffusion of both $\pi$ and $\pi^{-1}$. Definition 1.2.3 defines a natural partial order on the permutations: a permutation $\pi_1$ is better (at diffusing) than a permutation $\pi_2$ if $DR_{max}(\pi_1) \leq DR_{max}(\pi_2)$. Searching the best permutations (for the diffusion) directly can be difficult. As a result the methodology we adopt in this work is to search for permutations that diffuse totally in the forward direction and then check if their respective inverse also diffuses totally.

### 1.2.3 Even-odd Permutations

A naive way to search for optimal permutation would be to simply go through all of them and check the diffusion one permutation by one. However, there are $(2k)!$ permutations, which quickly grows beyond practical means. For example with $2k = 32$, approximately $2^{117}$ permutations should be checked. To reduce the number of permutations that will be tested, we will restrict ourselves to a specific class of permutations and give an equivalence relation which further reduces the number of permutations to be considered.

In [131], Suzaki and Minematsu did an exhaustive search for $1 \leq k \leq 8$, and made the observation that every optimal permutation (for such $k$) mapped even-number input blocks to odd-number output blocks and vice versa. We call such permutations even-odd. In the rest of this chapter, we

will use the following notation for even-odd permutations. An even-odd permutation $\pi$ of size $2k$ will be denoted by the pair of permutations $(p, q)$ of size $k$ verifying $\forall i \in [0, k-1],\ \pi(2i) = 2 \cdot p(i)+1$ and $\pi(2i+1) = 2 \cdot q(i)$. The search space is now reduced to $(k!)^2$ permutations.

According to this, [131] gives the following lower-bound on the diffusion round of even-odd permutations $(p, q)$.

**Proposition 1.2.4.** *Let $\mathcal{F}_i$ be the Fibonacci sequence, i.e. $\mathcal{F}_0 = 0, \mathcal{F}_1 = 1$ and $\mathcal{F}_i = \mathcal{F}_{i-1}+\mathcal{F}_{i-2}, i \geq 2$. Let $\pi = (p, q)$ be an even-odd permutation over $2k$ elements, and $i$ be the smallest integer such that $\mathcal{F}_i \geq k$. Then $DR_{max}(\pi) \geq i + 1$.*

For a given permutation $\pi$, if the inequality is tight, we say that $\pi$ is tight. A proof of this proposition already exists in both [131] and [39]. According to our results, we will give another proof of this proposition in Section 1.3. We will also show in Section 1.3 that this bound is tight for the cases $2k = 28, 30, 32, 36$ and strict for $2k = 34, 38, 40, 42$.

### 1.2.4 Equivalence Classes of Even-odd Permutations

To further reduce the size of the search space, as in [39], we use some equivalence classes, given by the following definition.

**Definition 1.2.5.** *Let $\pi$ and $\pi'$ be two even-odd permutations over $2k$ elements. We say that $\pi$ and $\pi'$ are equivalent if there exists a permutation $\varphi$ over $2k$ elements such that*

$$\pi' = \varphi \circ \pi \circ \varphi^{-1}.$$

From [39], we can then give a set of permutations $\mathbb{P}_k$ such that for any equivalence class, there exists at least one $\pi \in \mathbb{P}_k$ which belongs to this class. This effectively gives us a set of class representatives (in which a few of them are redundant), and this set can be built from the following proposition, proven in [39]. Recall that any permutation can be decomposed into a composition of cycles. We call *cycle structure* the unordered set of the length of these cycles, for example the permutation

$$(0\ 1\ 2\ 3)(4\ 5)(6\ 7)(8)$$

has a cycle structure of $\{4, 2, 2, 1\}$.

**Proposition 1.2.6.** *Let $\mathbb{P}_k$ be a set of even-odd permutations $\pi = (p, q)$ over $2k$ elements constructed as follows. For each possible cycle structure $c$ of a permutation over $k$ elements, pick one permutation $p$ which has a cycle structure equal to $c$. Then, for every permutation $q$ over $k$ elements, add $(p, q)$ in the set $\mathbb{P}_k$. By doing so, $\mathbb{P}_k$ contains at least one representative of each equivalence class induced by Definition 1.2.5. Moreover, $\mathbb{P}_k$ contains exactly $\mathcal{N}_k.k!$ elements, where $\mathcal{N}_k$ is the number of partitions of the integer $k$.*

This allows us to only consider $\mathcal{N}_k.k!$ permutations instead of $(k!)^2$. This is a significant improvement, as for example with $k = 16$, there are only $231 \times 16! \simeq 2^{52}$ permutations to go through, instead of $(16!)^2 \simeq 2^{88}$. However when $k$ grows, it is still too big a number to try an exhaustive search. As such, we propose in Section 1.4 an efficient search algorithm to find all optimal even-odd permutations for a given $k$, without needing to do an exhaustive search.

## 1.3 Characterization of Full Diffusion

In this section, we will explain our strategy to search for a tight even-odd permutation, that is, a permutation with a diffusion round reaching the Fibonacci bound given in Proposition 1.2.4. We will first give an algebraic characterization for a permutation to have full diffusion, then give an algorithm to exploit this characterization and quickly search all such permutations. Note that here we only focus on the diffusion round of the permutation when considering encryption. That is, for a given permutation $\pi$, we focus only on $DR(\pi) = \max_{0 \leq i \leq 2k-1} \{DR_i(\pi)\}$. Then, once we found a permutation reaching the Fibonacci bound, we can easily check if $\pi^{-1}$ also reaches this bound, and if that is the case, we found a tight permutation.

We describe here the main tools we used to design our search algorithm. Note that for two permutations $p, q$, we denote the composition $p \circ q$ by $pq$ for better reading. We first begin by giving the following proposition.

**Proposition 1.3.1.** *Let $\pi = (p, q)$ be an even-odd permutation over $2k$ elements. Then $\pi$ achieves full diffusion after $r$ rounds if and only if each block $X_j^0$ is diffused to at least one block of each pair at the input of the $(r-1)$-th round, i.e. diffused to either $X_{2j'}^{r-1}$ or $X_{2j'+1}^{r-1}$ for each $j' \in \{0, \ldots, k-1\}$.*

*Proof.* Suppose that a given block $X_i^0$ has been fully diffused, i.e. to every block $\widetilde{X}_{2j}^r$ and $\widetilde{X}_{2j+1}^r, j \in \{0, \ldots, k-1\}$. Then $X_i^0$ must have diffused to at least $X_{2j+1}^r$ for every $j$, as it is the only way to reach $\widetilde{X}_{2j+1}^r$. Thus, $X_i^0$ must have diffused to $\widetilde{X}_{2j'}^{r-1}$ with $j' = p^{-1}(j)$, which means that it has diffused to either $X_{2j'}^{r-1}$ or $X_{2j'+1}^{r-1}$.

On the other hand, suppose that a given block $X_i^0$ has diffused to an even block $X_{2j}^{r-1}$, then $X_i^0$ will be diffused to only $\widetilde{X}_{2j}^{r-1}$. If $X_i^0$ has diffused to an odd block $X_{2j+1}^{r-1}$, it will be diffused to both $\widetilde{X}_{2j}^{r-1}$ and $\widetilde{X}_{2j+1}^{r-1}$. In both cases, it will be diffused to $\widetilde{X}_{2j}^{r-1}$, then to $X_{2j'+1}^r$ with $j' = p(j)$, and finally to both $\widetilde{X}_{2j'}^r$ and $\widetilde{X}_{2j'+1}^r$. Thus, if for all $j \in \{0, \ldots, k-1\}$, $X_i^0$ is diffused to any block of the $j$-th pair at the input of the $(r-1)$-th round, it will be diffused to every block $\widetilde{X}_{2p(j)}^r$ and $\widetilde{X}_{2p(j)+1}^r$, and since $p$ is a permutation, this means that we have full diffusion for $X_i^0$. □

**Corollary 1.3.2.** *Let $\pi = (p, q)$ be an even-odd permutation over $2k$ elements. Then $\pi$ achieves full diffusion after $r$ rounds if and only if each even block $X_{2j}^0, j \in \{0, \ldots, k-1\}$ diffuses to every even block $\widetilde{X}_{2j'}^{r-1}, j' \in \{0, \ldots, k-1\}$.*

*Proof.* For the proof of the previous theorem, we can easily see that a block $X_j^0$ diffuses to either $X_{2j'}^{r-1}$ or $X_{2j'+1}^{r-1}$ if and only if $X_j^0$ diffuses to $\widetilde{X}_{2j'}^{r-1}$. Moreover, we can easily see that if $X_{2j}^0$ is fully diffused, so is $X_{2j+1}^0$. Indeed, $X_{2j}^0$ being fully diffused is the same as $\widetilde{X}_{2j}^0$ being fully diffused, and $X_{2j+1}^0$ is always diffused to $\widetilde{X}_{2j}^0$. □

Thus we only need to focus on the diffusion of each block $X_{2j}^0$ to each block $\widetilde{X}_{2j}^{r-1}$. Now we can take a look a what would happen in an ideal scenario. Assume that we are studying the diffusion of a block $X_{2j}^0$. Then $X_{2j}^0$ is diffused to $\widetilde{X}_{2j_0^1}^0$ with $j_0^1 = j$. It is then diffused to both $\widetilde{X}_{2j_0^2}^1$ and $\widetilde{X}_{2j_0^2+1}^1$, with $j_0^2 = p(j_0^1)$. Then again :

- $\widetilde{X}_{2j_0^2}^1$ is diffused to both $\widetilde{X}_{2j_0^3}^2$ and $\widetilde{X}_{2j_0^3+1}^2$, with $j_0^3 = p(j_0^2)$.
- $\widetilde{X}_{2j_0^2+1}^1$ is diffused to $\widetilde{X}_{2j_1^3}^2$ with $j_1^3 = q(j_0^2)$

Assuming an ideal scenario, we would have $j_0^3 \neq j_1^3$, i.e. $X_{2j}^0$ has diffused to two different blocks after 4 rounds (minus the application of $\pi$ on the fourth round). We can then keep going and get a series of $j_\ell^i$ which gives us the blocks on which $X_{2j}^0$ has diffused after $i+1$ rounds minus the last application of $\pi$, always assuming that we never have $j_\ell^i = j_{\ell'}^i$ for $\ell \neq \ell'$. The propagation for up to 7 rounds is given in Figure 1.2.

However, we cannot have $j_\ell^i \neq j_{\ell'}^i$ with $\ell \neq \ell'$ forever. Indeed, since we only have $k$ blocks, we are bound at some point to have $j_\ell^i = j_{\ell'}^i$ and $\ell \neq \ell'$. However, we can easily compute the actual value of each $j_\ell^i$. Indeed, if we take for example $j_6^6$ in Figure 1.2, then we know that

$$j_6^6 = (qppqp)(j_0^1) = (qppqp)(j).$$

Denote by $\mathbb{J}_j^i$ the set of equations obtained by expressing every $j_\ell^i$ that way. For example, we would have

$$\begin{aligned}\mathbb{J}_j^6 = \{&(ppppp)(j),\\ &(qpppp)(j),\\ &(pqppp)(j),\\ &(ppqpp)(j),\\ &(qpqpp)(j),\\ &(pppqp)(j),\\ &(qppqp)(j),\\ &(pqpqp)(j)\}\end{aligned}$$

According to this, we can give a generic way to compute $\mathbb{J}_j^i$. We start with $\mathbb{J}_j^1 = \{j\}$ and $\mathbb{J}_j^2 = \{p(j)\}$. To build $\mathbb{J}_j^i$ from $\mathbb{J}_j^{i-1}$, we begin by adding $p(x)$ to $\mathbb{J}_j^i$ for every term $x$ in $\mathbb{J}_j^{i-1}$. Then, for every term $x$ in $\mathbb{J}_j^{i-1}$ such that $x$ can be written as $x = p(y)$ for some $y \in \mathbb{J}_j^{i-2}$, we also add $q(x)$ to $\mathbb{J}_j^i$.

We can justify this construction as follows. Suppose that a given $j'$ belongs to $\mathbb{J}_j^{i-2}$ because $X_{2j}^0$ diffuses to $\widetilde{X}_{2j'}^{i-2}$. Then $X_{2j}^0$ diffuses to both $\widetilde{X}_{2j''}^{i-1}$ and $\widetilde{X}_{2j''+1}^{i-1}$ with $j'' = p(j')$. Thus for the next round, $X_{2j}^0$ will diffuse to both $X_{2\widetilde{j}+1}^i$ and $X_{2\widetilde{j'}}^i$, with $\widetilde{j} = p(j'')$ and $\widetilde{j'} = q(j'')$.

On the other hand, suppose that $j'$ belongs to $\mathbb{J}_j^{i-2}$ because $X_{2j}^0$ diffuses to $\widetilde{X}_{2j'+1}^{i-2}$. In that case, $X_{2j}^0$ will only diffuse to $\widetilde{X}_{2j''}^{i-1}$ with $j'' = q(j')$. For the next round, $X_{2j}^0$ only diffuses to $X_{2\widetilde{j}+1}^i$ with $\widetilde{j} = p(j'')$.

Thus in both cases, we need to have $\widetilde{j} = p(j'')$, but we only require $\widetilde{j'} = q(j'')$ in the first case, which corresponds exactly to the case where the previous term started with a composition by $p$.

Note that from this construction, we can deduce the following proposition.

**Proposition 1.3.3.** *The size of $\mathbb{J}_j^i$ is exactly $\mathcal{F}_i$ where $\mathcal{F}_i$ is the $i$-th term of the Fibonacci sequence.*

*Proof.* We can prove this by induction. Both $\mathbb{J}_j^1$ and $\mathbb{J}_j^2$ are of size 1, which corresponds to $\mathcal{F}_1$ and $\mathcal{F}_2$. We first add an element $p(x)$ to $\mathbb{J}_j^i$ for every $x \in \mathbb{J}_j^{i-1}$, thus $\mathcal{F}_{i-1}$ elements. Then, for every $x$ in $\mathbb{J}_j^{i-1}$ such that $x = p(y)$ with $y \in \mathbb{J}_j^{i-2}$, we add $q(x)$ to $\mathbb{J}_j^i$. However, according to our construction, $\mathbb{J}_j^{i-1}$ contains such an element $x = p(y)$ for *every* term $y \in \mathbb{J}_j^{i-2}$. Thus, there are $\mathcal{F}_{i-2}$ such terms. In the end, $\mathbb{J}_j^i$ contains $\mathcal{F}_{i-1} + \mathcal{F}_{i-2} = \mathcal{F}_i$ elements, which concludes the induction. □

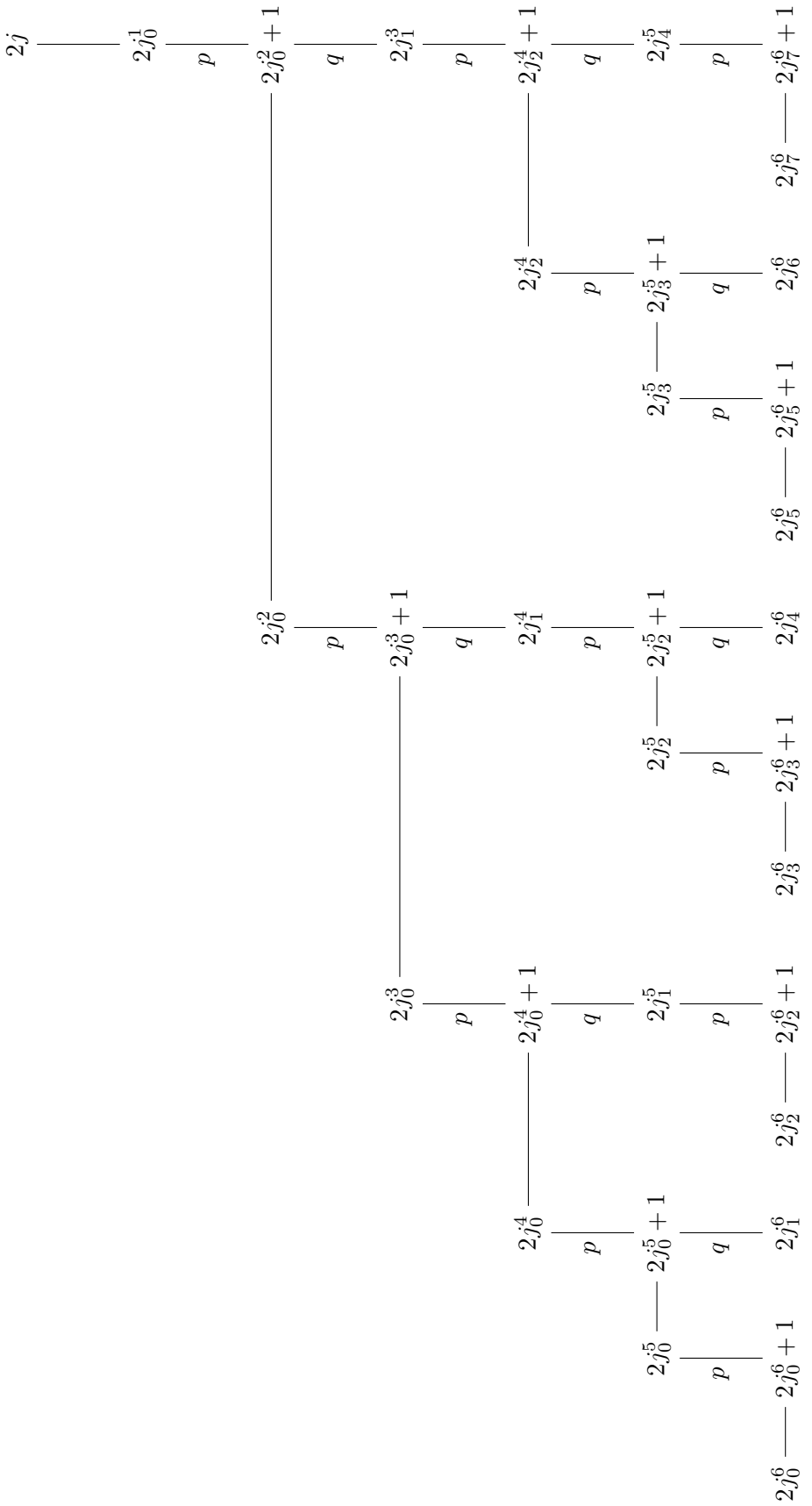


Figure 1.2: Propagation tree for 7 rounds (minus the last application of $\pi$)

We can now use those sets $\mathbb{J}_j^i$ to fully characterize the fact that a block fully diffuses when using a given permutation.

**Theorem 1.3.4.** *Let $\mathbb{J}_j^{r-1}$ be the set of equations as defined above. Then for a given permutation $\pi = (p, q)$ over $2k$ blocks, $X_{2j}^0$ is fully diffused after $r$ rounds if and only if $\mathbb{J}_j^{r-1}$ contains every number in $\{0, \dots, k-1\}$ at least once.*

*Proof.* As $\mathbb{J}_j^{r-1} = \{j_0^{r-1}, \dots, j_{\ell_{r-1}}^{r-1}\}$ is defined, it basically represents that $X_{2j}^0$ diffuses to every $\widetilde{X}_{2j_\ell^i}^i$. Thus, if $\mathbb{J}_j^i$ contains every number in $\{0, \dots, k-1\}$ at least once, it exactly means that $X_{2j}^0$ diffuses to each block $\widetilde{X}_{2j'}^i, j' \in \{0, \dots, k-1\}$. According to Corollary 1.3.2, this means that $X_{2j}^0$ achieves full diffusion after $i+1$ rounds. □

We can then easily deduce the following corollary.

**Corollary 1.3.5.** *Let $\pi = (p, q)$ be a permutation over $2k$ elements. Then we have $DR(\pi) = i+1$ if and only if $i$ is the smallest integer such that for every $j \in \{0, \dots, k-1\}$, $\mathbb{J}_j^i$ contains every number in $\{0, \dots, k-1\}$ at least once.*

This gives us another proof for the Fibonacci bound given in Proposition 1.2.4. Indeed, for $\mathbb{J}_j^i$ to contain every number in $\{0, \dots, k-1\}$ at least once, $\mathbb{J}_j^i$ must contain at least $k$ terms. Thus, and since the size of $\mathbb{J}_j^i$ does not depend on $j$, the minimal number of rounds needed to have full diffusion for every block must be such that $\left|\mathbb{J}_j^i\right| = \mathcal{F}_i \geq k$. According to the previous corollary, if $i$ is the smallest integer such that $\mathcal{F}_i \geq k$, this exactly means that $DR_{max}(\pi) \geq i+1$

Note that from the construction of any $\mathbb{J}_j^i$, each term starts with a composition by $p$. Since $p$ is a permutation, and we want full diffusion for every blocks, we can remove this first $p$ from every term to get a smaller representation. Essentially, this means that we are considering the diffusion of the block $p^{-1}(j)$, but we will still write $\mathbb{J}_j^i$. As such, $\mathbb{J}_j^6$ for example is thus rewritten as

$$\begin{aligned}\mathbb{J}_j^6 = \{&(p^4)(j),\\ &(qp^3)(j),\\ &(pqp^2)(j),\\ &(p^2qp)(j),\\ &(qpqp)(j),\\ &(p^3q)(j),\\ &(qp^2q)(j),\\ &(pqpq)(j)\}\end{aligned}$$

To illustrate the previous characterization, we introduce what we call the *diffusion table (of rank i)* of an even-odd permutation $(p, q)$ of size $2k$. The columns are indexed by the numbers from 0 to $k-1$ and the row are indexed by the products of $p$ and $q$ used to generate all sets $\mathbb{J}_j^i$. Each cell of the table is the value obtained by applying the permutation indexing the row to the value indexing the column of the cell. For example, the cell indexed by $p^i$ and 0 contains $p^i(0)$. This provides a clear visualization of our characterization, as the $j$-th column is exactly $\mathbb{J}_j^i$.

Thus, we can easily illustrate Corollary 1.3.5 by verifying that every column of this table contains every possible values. We thus add one more row at the end of diffusion table called *diff* which contains the number of different values in a column. By construction, this is exactly the number

| $x$ | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|---|
| $p^5$ | 3 | 4 | 5 | 6 | 7 | 0 | 1 | 2 |
| $p^4q$ | 4 | 5 | 6 | 7 | 0 | 1 | 2 | 3 |
| $p^3qp$ | 4 | 5 | 6 | 7 | 0 | 1 | 2 | 3 |
| $p^2qp^2$ | 4 | 5 | 6 | 7 | 0 | 1 | 2 | 3 |
| $pqp^3$ | 4 | 5 | 6 | 7 | 0 | 1 | 2 | 3 |
| $qp^4$ | 4 | 5 | 6 | 7 | 0 | 1 | 2 | 3 |
| $p^2qpq$ | 5 | 6 | 7 | 0 | 1 | 2 | 3 | 4 |
| $pqp^2q$ | 5 | 6 | 7 | 0 | 1 | 2 | 3 | 4 |
| $qp^3q$ | 5 | 6 | 7 | 0 | 1 | 2 | 3 | 4 |
| $pqpqp$ | 5 | 6 | 7 | 0 | 1 | 2 | 3 | 4 |
| $qp^2qp$ | 5 | 6 | 7 | 0 | 1 | 2 | 3 | 4 |
| $qpqp^2$ | 5 | 6 | 7 | 0 | 1 | 2 | 3 | 4 |
| $qpqpq$ | 6 | 7 | 0 | 1 | 2 | 3 | 4 | 5 |
| diff | 4 | 4 | 4 | 4 | 4 | 4 | 4 | 4 |

| $x$ | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|---|
| $p^5$ | 4 | 3 | 5 | 1 | 6 | 7 | 0 | 2 |
| $p^4q$ | 3 | 2 | 1 | 4 | 0 | 6 | 7 | 5 |
| $p^3qp$ | 2 | 6 | 7 | 5 | 1 | 3 | 4 | 0 |
| $p^2qp^2$ | 6 | 7 | 4 | 0 | 5 | 2 | 3 | 1 |
| $pqp^3$ | 1 | 4 | 3 | 2 | 0 | 6 | 7 | 5 |
| $qp^4$ | 2 | 5 | 7 | 6 | 3 | 1 | 4 | 0 |
| $p^2qpq$ | 7 | 1 | 0 | 6 | 3 | 5 | 2 | 4 |
| $pqp^2q$ | 4 | 5 | 2 | 1 | 7 | 0 | 6 | 3 |
| $qp^3q$ | 5 | 0 | 6 | 2 | 4 | 3 | 1 | 7 |
| $pqpqp$ | 5 | 0 | 6 | 3 | 2 | 4 | 1 | 7 |
| $qp^2qp$ | 0 | 3 | 1 | 7 | 6 | 5 | 2 | 4 |
| $qpqp^2$ | 3 | 1 | 2 | 4 | 7 | 0 | 5 | 6 |
| $qpqpq$ | 1 | 6 | 4 | 3 | 5 | 7 | 0 | 2 |
| diff | 8 | 8 | 8 | 8 | 8 | 8 | 8 | 8 |

Table 1.1: Diffusion tables for the cyclical shift (left table) and one optimal permutation proposed by [39] (right table).

of elements of $\mathbb{J}_j^i$ where $j$ is the index of the column. In tables constructed as described, the full diffusion of a permutation corresponds to a *diff* row containing only the value $k$.

For example, we give in Table 1.1 the diffusion tables for the cyclical shift (i.e. $p = (7, 0, 1, 2, 3, 4, 5, 6)$ and $q = (0, 1, 2, 3, 4, 5, 6, 7)$) and one of the optimal permutation proposed by [39] (i.e. $p = (6, 3, 7, 1, 0, 2, 4, 5)$ and $q = (3, 5, 1, 6, 4, 0, 2, 7)$) for $k = 8$ and $i = 7$, thus the optimal permutation clearly have a diffusion round of 8.

Finally, we can reformulate the problem of finding optimal even-odd permutations with these tables. Indeed, it corresponds to finding the minimal $i$ and even-odd permutations of size $2k$ such that their diffusion table have their *diff* row containing only $k$.

## 1.4 Searching for an Optimal Permutation over 9 Rounds

### 1.4.1 Efficient Search Algorithm

First, we can see that our characterization can be very efficiently implemented, as testing if $\pi = (p, q)$ has full diffusion mostly requires only a few table lookups. Indeed, its efficiency allowed us to recover all optimal even-odd permutations for $k \leq 13$ with a basic exhaustive search. Especially, for $k = 13$, we were able to go through all $\mathcal{N}_{13}.13! \simeq 2^{39}$ permutations and check them in about 410 minutes on a single core. While these optimal permutations were already known, it shows that the sophisticated techniques introduced in [39] were not necessary for these cases.

However for $k \geq 14$, it becomes too expensive to make this exhaustive search. We thus focus on finding optimal even-odd permutations for $14 \leq k \leq 21$, hence such permutations would have a diffusion round of 9. Given a cycle structure for $p$, we can easily find a permutation $p$ with such structure and thus we need to search $q$ such that $\pi = (p, q)$ needs 9 rounds to reach full diffusion, i.e., such that each $\mathbb{J}_j^8$ contains all numbers from 0 to $k - 1$.

Note that we cannot exploit $\mathbb{J}_j^8$ directly. Indeed, one might want to guess parts of $q$ and check if $\mathbb{J}_j^8$ does not contains too many duplicates. However, to fully compute $\mathbb{J}_j^8$, we need to guess $q$ in

its entirety, which makes this strategy too expensive. We thus describe an efficient way to exploit this characterization to find optimal even-odd permutations.

First for a given $j$, if we take a look at $\mathbb{J}_j^6$, we can see that we need to make only 7 guesses over the images of $q$ to fully compute $\mathbb{J}_j^6$. Indeed, we need to know

$$q(j), (qp)(j), (qp^2)(j), (qp^3)(j), (qpq)(j), (qp^2q)(j) \text{ and } (qpqp)(j).$$

Let $\mathbb{X}_j^6$ and $\mathbb{Y}_j^6$ be two subsets of $\mathbb{J}_j^6$, such that $\mathbb{X}_j^6 \cup \mathbb{Y}_j^6 = \mathbb{J}_j^6$, with

$$\mathbb{X}_j^6 = \{p^4(j), (pqp^2)(j), (p^2qp)(j), (p^3q)(j), (pqpq)(j)\}$$

$$\text{and } \mathbb{Y}_j^6 = \{(qp^3)(j), (qpqp)(j), (qp^2q)(j)\}.$$

According to the construction of $\mathbb{J}_j^8$, we can actually write

$$\mathbb{J}_j^8 = p^2(\mathbb{X}_j^6 \cup \mathbb{Y}_j^6) \cup (pq)(\mathbb{X}_j^6) \cup (qp)(\mathbb{X}_j^6 \cup \mathbb{Y}_j^6).$$

Assume that we made the 7 guesses mentioned above. In that case, we know the exact values in both $\mathbb{X}_j^6$ and $\mathbb{Y}_j^6$. Moreover, since $p$ is known, we know exactly the values in $p^2(\mathbb{X}_j^6 \cup \mathbb{Y}_j^6)$. Finally, since we guessed 7 images of $q$, there might be some values in $(pq)(\mathbb{X}_j^6)$ and $(qp)(\mathbb{X}_j^6 \cup \mathbb{Y}_j^6)$ that are known.

Hence, we create three sets $\mathbb{K}_j, \widetilde{\mathbb{X}}_j^6$ and $\widetilde{\mathbb{Y}}_j^6$ :

- $\mathbb{K}_j$ is the set of all known values of $\mathbb{J}_j^8$. Thus $p^2(\mathbb{X}_j^6 \cup \mathbb{Y}_j^6) \subset \mathbb{K}_j$ and there might be a few elements from $(pq)(\mathbb{X}_j^6)$ and $(qp)(\mathbb{X}_j^6 \cup \mathbb{Y}_j^6)$ in $\mathbb{K}_j$ too.
- $\widetilde{\mathbb{X}}_j^6$ is the subset of $\mathbb{X}_j^6$ such that for any $x \in \widetilde{\mathbb{X}}_j^6$, the value of $q(x)$ yet remains to be determined.
- In the same way, $\widetilde{\mathbb{Y}}_j^6$ is the subset of $p(\mathbb{X}_j^6 \cup \mathbb{Y}_j^6)$ such that for any $x \in \widetilde{\mathbb{Y}}_j^6$, the value of $q(x)$ is not determined.

For $j$ to be fully diffused, we thus have the constraint

$$C_j : \left|\mathbb{K}_j \cup q(\widetilde{\mathbb{Y}}_j^6) \cup (pq)(\widetilde{\mathbb{X}}_j^6)\right| \geq k.$$

We then check if this constraint is valid, i.e. if there exist some guesses for the remaining images of $q$ such that $C_j$ holds, and this is described in the next section.

Now if we take a look at $\mathbb{J}_{j'}^6$ where $j' = p(j)$, we can see that we only need 3 more guesses to compute it, instead of 7 as before. Indeed, we already guessed

$$\begin{aligned}(qp)(j) &= q(j')\\ (qp^2)(j) &= (qp)(j')\\ (qp^3)(j) &= (qp^2)(j')\\ (qpqp)(j) &= (qpq)(j')\end{aligned}$$

and thus it only remains to guess

$$\begin{aligned}(qp^4)(j) &= (qp^3)(j')\\ (qp^2qp)(j) &= (qp^2q)(j')\\ (qpqp^2)(j) &= (qpqp)(j').\end{aligned}$$

By doing these guesses, we can build the sets $\mathbb{K}_{j'}, \mathbb{X}^6_{j'}$ and $\mathbb{Y}^6_{j'}$ as before, and thus get another constraint that needs to be checked

$$C_{j'} : \left| \mathbb{K}_{j'} \cup q(\widetilde{\mathbb{Y}}^6_{j'}) \cup (pq)(\widetilde{\mathbb{X}}^6_{j'}) \right| \geq k.$$

However by making those three new guesses, we might be able to compute new values in $\widetilde{\mathbb{X}}^6_j$ and $\widetilde{\mathbb{Y}}^6_j$. We thus need to update the constraint $C_j$ according to these guesses, and then check again if $C_j$ is valid.

This can be repeated until we have fully guessed $q$, in which case we have a solution, or show that no matter which guesses we made there is no solution which satisfies all constraints. This is the core of our algorithm, which is described from a high-level point of view in Algorithm 2.

**Algorithm 1** Searching for optimal even-odd permutations over 9 rounds

**function** NEXTGUESS$(p, q, j, \mathbb{C})$ *$\mathbb{C}$ is the list of known constraints*
  **if** $q$ is fully determined **then**
    Print $p, q$
  **else**
    **while** all guesses are not made **do**
      Guess $(qp^3)(j), (qp^2q)(j)$ and $(qpqp)(j)$
      Update every constraints in $\mathbb{C}$ according to those guesses
      Deduce the new constraint $C_j$
      $\mathbb{C}' \leftarrow \mathbb{C} \cup \{C_j\}$
      **if** $\exists$ invalid constraint in $\mathbb{C}'$ **then**
        Make a new guess
      **else**
        NEXTGUESS$(p, q, p(j), \mathbb{C}')$
      **end if**
    **end while**
  **end if**
**end function**

$p \leftarrow$ chosen permutation with a given structure cycle
$j \leftarrow$ an element from the smallest cycle of $p$
**while** all guesses are not made **do**
  Guess $q(j), (qp)(j), (qp^2)(j), (qp^3)(j), (qpq)(j), (qp^2q)(j)$ and $(qpqp)(j)$
  Deduce the constraint $C_j$
  **if** $C_j$ is a valid constraint **then**
    $\mathbb{C} \leftarrow \{C_j\}$
    NEXTGUESS$(p, q, p(j), \mathbb{C})$
  **end if**
**end while**

Note however that the actual algorithm is a bit more sophisticated. Indeed, it might occur at some point that $p(j)$ was already processed, i.e. $C_{p(j)}$ is already a constraint we have. When this happens, we need to choose another starting block $j$, and re-apply the algorithm, while still keeping all previously computed constraints. In practice, we found that the most efficient strategy is to use an element from the shortest cycle of $p$ as the first starting block. Then, if we need to choose another starting block, we pick an element in the next shortest cycle of $p$ and so on. Moreover,

when making some guesses for the images of $q$, it might happen that we already made this guess. This is not a problem, as this guess basically becomes free and does not add any more cost. Finally, except for the first seven guesses, we update and check all constraints after *each* guess.

### 1.4.2 Checking the Constraints

We first give a naive way to check if a constraint is valid. We are given three sets $\mathbb{K}, \mathbb{X}$ and $\mathbb{Y}$, resulting in the constraint

$$C : |\mathbb{K} \cup q(\mathbb{Y}) \cup (pq)(\mathbb{X})| \geq k.$$

We know the full permutation $p$, and for any $x \in \mathbb{X} \cup \mathbb{Y}$, $q(x)$ is still unknown. Let $\mathbb{A}$ denote the set of values $a$ for which we still do not know the preimage of $a$ through $q$, i.e. for any $a \in \mathbb{A}$, we do not know which $x$ results in $q(x) = a$. Considering the guesses we already made on $q$, we always know this set $\mathbb{A}$, and thus have the following two relations $(pq)(\mathbb{X}) \subset p(\mathbb{A})$ and $q(\mathbb{Y}) \subset \mathbb{A}$. According to this, we can write

$$|\mathbb{K} \cup q(\mathbb{Y}) \cup (pq)(\mathbb{X})| \leq |\mathbb{K} \cup \mathbb{A} \cup p(\mathbb{A})|.$$

Hence if $|\mathbb{K} \cup \mathbb{A} \cup p(\mathbb{A})| < k$, we know that the constraint $C$ cannot be valid. However, we can actually go further and get more precise information by doing the following.

We can formulate our problem in the following generic way. We are given three sets $\mathbb{K}, \mathbb{A}$, and $\mathbb{B}$ $(= p(\mathbb{A}))$, and we search for two sets $\widetilde{\mathbb{A}} \subset \mathbb{A}$ and $\widetilde{\mathbb{B}} \subset \mathbb{B}$ such that $|\mathbb{K} \cup \widetilde{\mathbb{A}} \cup \widetilde{\mathbb{B}}|$ is maximal, with $\widetilde{\mathbb{A}} = q(\mathbb{Y})$ and $\widetilde{\mathbb{B}} = (pq)(\mathbb{X})$. Note that, since $p$ and $q$ are permutations, we have $|\widetilde{\mathbb{A}}| = |\mathbb{X}|$ and $|\widetilde{\mathbb{B}}| = |\mathbb{Y}|$. Hence our idea is to determine whether there is at least one such pair $(\widetilde{\mathbb{A}}, \widetilde{\mathbb{B}})$ satisfying $|\mathbb{K} \cup \widetilde{\mathbb{A}} \cup \widetilde{\mathbb{B}}| \geq k$. Indeed if no such pair exists then constraint $C$ does not hold. Note that if $\mathbb{X} \cap \mathbb{Y} \neq \emptyset$ then it is possible for such pair to exist while $C$ does not hold. However we found this filter powerful enough for our need.

We can partition $\mathbb{K} \cup \mathbb{A} \cup \mathbb{B}$ into the following eight disjoint sets:

$$\begin{array}{ll} \mathbb{S}_0 = \mathbb{K} \cap \mathbb{A} \cap \mathbb{B} & \mathbb{S}_1 = \mathbb{K}^c \cap \mathbb{A} \cap \mathbb{B} \\ \mathbb{S}_2 = \mathbb{K} \cap \mathbb{A}^c \cap \mathbb{B} & \mathbb{S}_3 = \mathbb{K} \cap \mathbb{A} \cap \mathbb{B}^c \\ \mathbb{S}_4 = \mathbb{K}^c \cap \mathbb{A}^c \cap \mathbb{B} & \mathbb{S}_5 = \mathbb{K}^c \cap \mathbb{A} \cap \mathbb{B}^c \\ \mathbb{S}_6 = \mathbb{K} \cap \mathbb{A}^c \cap \mathbb{B}^c & \mathbb{S}_7 = \mathbb{K}^c \cap \mathbb{A}^c \cap \mathbb{B}^c \end{array}$$

Let $k_A$ (resp. $k_B$) denote the cardinality of $\widetilde{\mathbb{A}}$ (resp. $\widetilde{\mathbb{B}}$), and $k_A^i, k_B^i$ be such that

$$k_A^i = |\widetilde{\mathbb{A}} \cap \mathbb{S}_i| \leq \min(|\mathbb{S}_i|, k_A), \quad k_B^i = |\widetilde{\mathbb{B}} \cap \mathbb{S}_i| \leq \min(|\mathbb{S}_i|, k_B).$$

Since all $\mathbb{S}_i$ are disjoint, $\widetilde{\mathbb{A}} \subset \mathbb{A}$ and $\widetilde{\mathbb{B}} \subset \mathbb{B}$, notice that we have

$$\begin{aligned} k_A^2 = k_A^4 = k_A^6 = k_A^7 = 0 \text{ and } k_A = k_A^0 + k_A^1 + k_A^3 + k_A^5 \\ k_B^3 = k_B^5 = k_B^6 = k_B^7 = 0 \text{ and } k_B = k_B^0 + k_B^1 + k_B^2 + k_B^4. \end{aligned}$$

By selecting the two sets $\widetilde{\mathbb{A}} \cap \mathbb{S}_1$ and $\widetilde{\mathbb{B}} \cap \mathbb{S}_1$ as disjoint as possible we have:

$$\begin{aligned} |\mathbb{K} \cup \widetilde{\mathbb{A}} \cup \widetilde{\mathbb{B}}| = |K| + k_A + k_B - k_A^0 - k_B^0 - k_B^2 - k_A^3 \\ - \max(k_A^1 + k_B^1 - |\mathbb{S}_1|, 0) \end{aligned}$$

Indeed, first we have at most $|K| + k_A + k_B$ elements in $\mathbb{K} \cup \widetilde{\mathbb{A}} \cup \widetilde{\mathbb{B}}$. However among all those elements, some might be the same, which explains the remaining terms :

- Elements of $\widetilde{\mathbb{A}}$ and $\widetilde{\mathbb{B}}$ included in $\mathbb{S}_0$, $\mathbb{S}_2$ or $\mathbb{S}_3$ are duplicates since they all belong to $\mathbb{K}$.
- We need to take $k_A^1$ (resp. $k_B^1$) elements from $\mathbb{A}$ (resp. from $\mathbb{B}$), where all those elements belongs to $\mathbb{S}_1$. We thus have two cases. If $k_A^1+k_B^1 \leq |\mathbb{S}_1|$, we can freely choose all those elements without having duplicates between $\widetilde{\mathbb{A}}$ and $\widetilde{\mathbb{B}}$. Indeed for example, if we have $k_A^1 = k_B^1 = 1$ and $\mathbb{S}_1 = \{0, 1, 2\}$, then we can put 0 in $\widetilde{\mathbb{A}}$ and 1 in $\widetilde{\mathbb{B}}$, thus resulting in no duplicates between $\widetilde{\mathbb{A}}$ and $\widetilde{\mathbb{B}}$ . However if we have $k_A^1 + k_B^1 > |\mathbb{S}_1|$, then no matter what, we will have duplicates. Thus in the best case, we have $\max(k_A^1 + k_B^1 - |\mathbb{S}_1|, 0)$ duplicates that we need to count out.

Hence, maximizing $|\mathbb{K} \cup \widetilde{\mathbb{A}} \cup \widetilde{\mathbb{B}}|$ is straightforward as there is one specific order in which to find the values of $k_A^i$ and $k_B^i$ that always maximize the size of the union. We only give the way to optimally build $\widetilde{\mathbb{A}}$ since it is fully similar for $\widetilde{\mathbb{B}}$ :

- First, using elements from $\mathbb{S}_5$ to build $\widetilde{\mathbb{A}}$ does not add any duplicate, thus we first pull elements from $\mathbb{S}_5$ and $k_A^5 = \min(k_A, |\mathbb{S}_5|)$.
- As mentioned above, using one element from $\mathbb{S}_1$ adds either zero or one duplicate, thus we then pull elements from $\mathbb{S}_1$ and $k_A^1 = \min(k_A - k_A^5, |\mathbb{S}_1|)$.
- Finally, elements from either $\mathbb{S}_0$ and $\mathbb{S}_3$ necessarily add duplicates, so we freely choose any $k_A^0 \leq |\mathbb{S}_0|$ and $k_A^3 \leq |\mathbb{S}_3|$ such that $k_A^0 + k_A^3 = k_A - k_A^5 - k_A^1$.

Finally, computing the maximal value for $|\mathbb{K} \cup \widetilde{\mathbb{A}} \cup \widetilde{\mathbb{B}}|$ only requires to compute $|\mathbb{S}_1|$, $|\mathbb{S}_4|$ and $|\mathbb{S}_5|$ and we then check how it compares to $k$.

### 1.4.3 Results

We ran our algorithm for every $k$ such that we need at least 9 rounds to have full diffusion, according to Proposition 1.2.4. This corresponds to $14 \leq k \leq 21$, and we were able to find *all* optimal even-odd permutations for $k \in \{14, 15, 16, 18\}$. For $k \in \{17, 19, 20, 21\}$, our algorithm allowed us to prove that there is no even-odd permutation leading to a full diffusion after 9 rounds. Since 9 rounds correspond to the Fibonacci bound, we know that for these cases, we need at least 10 rounds to have full diffusion, and we give later in this section an optimal solution for $k = 17$ reaching full diffusion in 10 rounds, as well as good permutations for $k = 19, 20, 21$ with a diffusion round of 11. We can thus give the following theorem to summarize our results.

**Theorem 1.4.1.** *To build a Generalized Feistel Network $GFN_\pi^k$ with full diffusion where $\pi$ is an even-odd permutation, we have :*

- *For $k = 14, 15, 16$ and $18$, the optimal number of rounds for full diffusion is 9.*
- *For $k = 17$, the optimal number of rounds for full diffusion is 10.*
- *For $k = 19, 20$ and $21$, the optimal number of rounds for full diffusion is at least 10 and at most 11.*

We give in Table 1.2 an overview of our results. The first column gives the total time needed for our algorithm to either exhaust all optimal even-odd permutations, or prove that no such permutation exists. Note that this is the total CPU time, i.e. when using a single CPU, however our algorithm is highly parallelizable and thus the real time can be drastically reduced.[3] This

[3]Less than one hour for each $k$ using 72 threads.

| $k$ | Time | Structure of $p$ | Structure of $q$ | Number of solutions |
|---|---|---|---|---|
| 14 | 180 min | (6, 6, 1, 1) | (6, 6, 2) | 144 |
| | | (6, 6, 2) | (6, 6, 1, 1) | 144 |
| | | (6, 3, 2, 2, 1) | (6, 3, 2, 2, 1) | 144 |
| | | (12, 2) | (12, 1, 1) | 24 |
| | | (12, 1, 1) | (12, 2) | 24 |
| 15 | 480 min | (10, 2, 2, 1) | (10, 2, 2, 1) | 160 |
| 16 | 1023 min | (6, 6, 3, 1) | (6, 6, 3, 1) | 432 |
| | | (6, 6, 2, 2) | (6, 3, 3, 2, 1, 1) | 288 |
| | | (6, 3, 3, 2, 1, 1) | (6, 6, 2, 2) | 216 |
| 17 | 1700 min | - | - | 0 |
| 18 | 2213 min | (8, 8, 1, 1) | (8, 8, 2) | 256 |
| | | (8, 8, 2) | (8, 8, 1, 1) | 256 |
| 19 | 1913 min | - | - | 0 |
| 20 | 1116 min | - | - | 0 |
| 21 | 400 min | - | - | 0 |

Table 1.2: Results for optimal permutations with $DR_{max}(\pi) = 9$

| $(p, q)$ |
|---|
| $p = (1, 2, 3, 4, 5, 0, 7, 8, 9, 10, 11, 6, 13, 14, 12, 15)$<br>$q = (2, 6, 12, 10, 1, 13, 4, 15, 7, 9, 14, 5, 8, 3, 11, 0)$ |
| $p = (1, 2, 3, 4, 5, 0, 7, 8, 9, 10, 11, 6, 13, 14, 12, 15)$<br>$q = (5, 6, 13, 10, 4, 12, 9, 15, 2, 1, 14, 7, 11, 3, 8, 0)$ |
| $p = (1, 2, 3, 4, 5, 0, 7, 8, 9, 10, 11, 6, 13, 12, 15, 14)$<br>$q = (2, 1, 12, 8, 7, 14, 5, 4, 13, 11, 10, 15, 9, 3, 6, 0)$ |
| $p = (1, 2, 3, 4, 5, 0, 7, 8, 6, 10, 11, 9, 13, 12, 14, 15)$<br>$q = (7, 6, 14, 9, 11, 15, 1, 12, 2, 13, 5, 4, 10, 8, 3, 0)$ |

Table 1.3: Optimal equivalence classes with $k = 16$

shows that our algorithm is extremely efficient, as it can quickly solve the case $k = 16$ for which [39] were not able to give an optimal solution. The second (resp. third) column gives the possible cycle structures of $p$ (resp. $q$) in an optimal permutation, and the last column gives the number of solutions which have this structure. We can notice that not only the number of solutions is quite low, but also that the number of possible cycle structures is also quite limited. Moreover, we *always* have a fixed point in either $p$ or $q$.

The most important result in this table is that there are actually even-odd permutations which have full diffusion after 9 rounds for $k = 16$, while both [131] and [39] could only find a permutation with full diffusion after 10 rounds, leaving open the question of whether the theoretical bound of 9 rounds (from Proposition 1.2.4) could be reached. Our results shows that it is indeed possible, and thus this proves that our permutations are optimal when considering even-odd permutations. We will see in the next section that we can further regroup these permutations into more precise equivalence classes, leading for the case $k = 16$ to four equivalence classes, given in Table 1.3.

### 1.4.4 Security Analysis

Recall that our search space $\mathbb{P}_k$ defined in Proposition 1.2.6 contains *at least* one representative for each class. Hence, among all the permutations we found, some of them might actually be in the same equivalence class. We can thus go further and regroup all representatives that belong to the same class using the following proposition.

**Proposition 1.4.2.** *Let $\pi = (p, q)$ be a permutation over $2k$ elements. Then for any permutation $r$ such that $r \circ p \circ r^{-1} = p$, $(p, q)$ and $(p, r \circ q \circ r^{-1})$ are equivalent.*

*Proof.* Let $\pi = (p, q)$ and $\pi' = (p, r \circ q \circ r^{-1})$ where $r$ is a permutation such that $r \circ p \circ r^{-1} = p$. Recall that we have $\pi(2i) = 2p(i) + 1$ and $\pi(2i + 1) = 2q(i)$, for all $i \in \{0, \ldots, k - 1\}$. Now let $\varphi$ be the permutation over $2k$ elements defined as

$$\varphi(2i) = 2r(i),\ \varphi(2i + 1) = 2r(i) + 1,\ \forall i \in \{0, \ldots, k - 1\}.$$

Then we have $\pi' = \varphi \circ \pi \circ \varphi^{-1}$. Indeed, if we look at the image of an even number $2i$, we have

$$\begin{aligned} \varphi \circ \pi \circ \varphi^{-1}(2i) &= \varphi \circ \pi(2r^{-1}(i)) \\ &= \varphi(2(p \circ r^{-1})(i) + 1) \\ &= 2(r \circ p \circ r^{-1})(i) + 1 \\ &= 2p(i) + 1 = \pi'(2i). \end{aligned}$$

In the same way, the image of an odd number $2i + 1$ is

$$\begin{aligned} \varphi \circ \pi \circ \varphi^{-1}(2i + 1) &= \varphi \circ \pi(2r^{-1}(i) + 1) \\ &= \varphi(2(q \circ r^{-1})(i)) \\ &= 2(r \circ q \circ r^{-1})(i) \\ &= \pi'(2i + 1) \end{aligned}$$

We thus have $\pi' = \varphi \circ \pi \circ \varphi^{-1}$. Hence, $\pi$ and $\pi'$ are conjugate and thus equivalent, according to Definition 1.2.5. □

This leads us to the equivalence classes given in Table 1.4 to 1.7 for $k = 14, 15, 18$. The column $(p, q)$ gives both permutations $p$ and $q$. The column Imp. Diff. gives the number of rounds for the longest Impossible Differential distinguisher. Note that this is only considering *structural* Impossible Differentials, where we do not specify neither the size of the blocks nor the definition of the S-boxes, such that contradictions are obtained on blocks rather than bits. The columns $S_N^{s,\delta}$ give the minimal number of rounds to get at least $N$ active S-boxes, where each S-box is of size $s$ and the highest differential probability is $2^{-\delta}$. We chose to only consider three cases : $S_N^{4,2}$, $S_N^{8,6}$ and $S_N^{8,7}$. The first case represents the best case for 4-bit S-boxes. Indeed, we know that there is no APN bijective S-boxes of size 4 (which would lead to a highest differential probability of $2^{-3}$). As such, the best case is when the highest differential probability is $2^{-2}$. It is still unknown whether 8-bit APN bijective S-boxes exist, so we consider both cases. If such an APN 8-bit S-box exists, the column $S_N^{8,7}$ is relevant, otherwise it would be $S_N^{8,6}$ (for example the AES S-box). The last thing is that $N$ depends on the size of the key (as well as $\delta$). Indeed, if we have a key of size $\lambda$, then we want $N$ to verify $2^{-\delta N} < 2^{-\lambda}$, i.e. $N > \frac{\lambda}{\delta}$. As the evaluation of the minimal number of rounds to get at least

$N$ S-boxes can be quite expensive, we limited ourselves to $\lambda = 2ks$, where $k$ follows the notation in this chapter, i.e. we have $2k$ blocks of $s$ bits and the key is of the same size as the state. Finally, the last column $N_{20}$ shows the minimal number of active S-boxes for 20 rounds, as both [131] and [39] also gave this metric for the permutations they found. It is worth mentioning that while our permutations are optimal (w.r.t the diffusion round), for the case $k = 16$, they have a minimal number of active S-boxes over 20 rounds which is lower than for the permutations given in [39] in the same case, where those permutations have a diffusion round of 10 and at least 70 actives S-boxes over 20 rounds. However for all our permutations, the longest Impossible Differentials distinguisher we can build is over 17 rounds, which is at least one round lower than for the permutations with $k = 16$ given in [39].

Note that we were still able to find the following optimal even-odd permutation for $k = 17$, which thus has a diffusion round of 10 :

$$p = (7, 1, 4, 13, 8, 16, 2, 3, 12, 5, 0, 9, 15, 14, 10, 11, 6)$$
$$q = (8, 0, 9, 10, 3, 2, 16, 6, 14, 11, 7, 4, 1, 12, 5, 15, 13)$$

For this permutation, the longest Impossible Differential distinguisher is over 19 rounds, and $S_{69}^{4,2}, S_{46}^{8,6}, S_{39}^{8,7}, N_{20}$ are respectively 20, 16, 15 and 20. For $k = 19, 20, 21$, we easily found permutations reaching full diffusion after 11 rounds with a random search, leaving open the question to find one permutation with a diffusion round of 10. We give an example for these cases below

$$k = 19 : \quad \begin{aligned} p &= (18, 3, 5, 9, 13, 15, 10, 16, 11, 8, 6, 1, 0, 2, 14, 7, 17, 12, 4) \\ q &= (9, 14, 2, 6, 3, 8, 16, 4, 0, 13, 18, 15, 5, 11, 7, 17, 12, 1, 10) \end{aligned}$$

$$k = 20 : \quad \begin{aligned} p &= (14, 5, 15, 1, 17, 3, 11, 8, 4, 0, 6, 13, 19, 10, 2, 9, 18, 12, 16, 7) \\ q &= (1, 17, 5, 18, 12, 2, 0, 16, 13, 6, 3, 10, 14, 8, 11, 19, 9, 15, 7, 4) \end{aligned}$$

$$k = 21 : \quad \begin{aligned} p &= (19, 10, 7, 17, 2, 16, 20, 9, 6, 0, 3, 12, 18, 1, 4, 11, 15, 13, 14, 8, 5) \\ q &= (20, 12, 0, 8, 7, 1, 4, 2, 10, 13, 5, 6, 11, 14, 19, 15, 9, 16, 3, 17, 18) \end{aligned}$$

## 1.5 Conclusion

We solved a 10-year-old problem which was to find an optimal (w.r.t diffusion round) even-odd permutation for a Generalized Feistel Network with 32 blocks. More specifically, we showed that there exist permutations which have a diffusion round of 9, while the best permutation found before had a diffusion round of 10. To do so, we give a precise characterization for the permutation to have full diffusion after a given number of rounds. This characterization allowed us to get a very efficient exhaustive search for $k \leq 13$. Even if optimal permutations were already known for these sizes, this shows that our characterization is powerful, thus we have no need to use the elaborated techniques from [39] to treat all these cases. We then exploit this characterization to design a very efficient algorithm that allows us to exhibit *all* optimal even-odd permutations for 32 blocks, as well as for 28, 30 and 36 blocks, which also have an optimal diffusion round of 9 and were not given in the previous literature. For 34, 38, 40 and 42 blocks, our algorithm also allows us to prove that there is no even-odd permutation with a diffusion round of 9 (which is the lower bound), which is again a new result. However for these cases, we were able to give better optimality bounds when considering even-odd permutations, namely for $2k = 34$ the optimal number of rounds for full diffusion is exactly 10 rounds and for $2k = 38, 40, 42$, at most 11 rounds. We also give some

security evaluation for Impossible Differentials and Differentials (through the minimum number of active S-boxes). Especially for Impossible Differentials, for the 32 blocks case, all our permutations have their longest impossible differential distinguishers over 17 rounds, which is at least one round lower than every permutation given in [39] for this case.

| $(p, q)$ | Imp. Diff | $S_{57}^{4,2}$ | $S_{38}^{8,6}$ | $S_{33}^{8,7}$ | $N_{20}$ |
|---|---|---|---|---|---|
| $p = (1,2,3,4,5,0,7,8,9,10,11,6,12,13)$<br>$q = (10,7,13,11,9,8,4,1,12,5,3,2,6,0)$ | 17 | 19 | 14 | 13 | 66 |
| $p = (1,2,3,4,5,0,7,8,9,10,11,6,12,13)$<br>$q = (8,6,13,10,7,9,1,12,5,2,4,3,11,0)$ | 17 | 19 | 14 | 13 | 66 |
| $p = (1,2,3,4,5,0,7,8,9,10,11,6,13,12)$<br>$q = (9,1,13,5,2,10,3,7,12,11,8,4,6,0)$ | 17 | 19 | 14 | 13 | 66 |
| $p = (1,2,3,4,5,0,7,8,9,10,11,6,13,12)$<br>$q = (4,1,13,5,10,9,2,11,8,12,6,3,7,0)$ | 17 | 19 | 14 | 13 | 66 |
| $p = (1,2,3,4,5,6,7,8,9,10,11,0,13,12)$<br>$q = (3,5,2,13,0,10,9,11,8,12,6,4,7,1)$ | 17 | 22 | 14 | 13 | 52 |
| $p = (1,2,3,4,5,6,7,8,9,10,11,0,13,12)$<br>$q = (3,1,13,11,8,10,9,7,12,5,2,4,6,0)$ | 17 | 22 | 14 | 13 | 52 |
| $p = (1,2,3,4,5,6,7,8,9,10,11,0,12,13)$<br>$q = (3,11,8,13,6,10,9,5,2,12,0,4,7,1)$ | 17 | 22 | 14 | 13 | 52 |
| $p = (1,2,3,4,5,6,7,8,9,10,11,0,12,13)$<br>$q = (3,7,13,5,2,10,9,1,12,11,8,4,6,0)$ | 17 | 22 | 14 | 13 | 52 |
| $p = (1,2,3,4,5,0,7,8,6,10,9,12,11,13)$<br>$q = (4,9,6,11,13,12,10,2,8,1,5,3,7,0)$ | 17 | 23 | 19 | 18 | 46 |

Table 1.4: Security evaluation for the best equivalence classes with $k = 14$

| $(p, q)$ | Imp. Diff | $S_{61}^{4,2}$ | $S_{41}^{8,6}$ | $S_{35}^{8,7}$ | $N_{20}$ |
|---|---|---|---|---|---|
| $p = (1,2,3,4,5,6,7,8,9,0,11,10,13,12,14)$<br>$q = (12,5,10,3,11,1,13,9,14,7,4,6,2,8,0)$ | 17 | 20 | 16 | 14 | 61 |
| $p = (1,2,3,4,5,6,7,8,9,0,11,10,13,12,14)$<br>$q = (13,9,10,7,11,5,12,3,14,1,4,6,8,2,0)$ | 17 | 42 | 28 | 24 | 30 |

Table 1.5: Security evaluation for the best equivalence classes with $k = 15$

| $(p, q)$ | Imp. Diff | $S_{65}^{4,2}$ | $S_{43}^{8,6}$ | $S_{37}^{8,7}$ | $N_{20}$ |
|---|---|---|---|---|---|
| $p = (1, 2, 3, 4, 5, 0, 7, 8, 9, 10, 11, 6, 13, 14, 12, 15)$<br>$q = (2, 6, 12, 10, 1, 13, 4, 15, 7, 9, 14, 5, 8, 3, 11, 0)$ | 17 | 33 | 22 | 19 | 40 |
| $p = (1, 2, 3, 4, 5, 0, 7, 8, 9, 10, 11, 6, 13, 14, 12, 15)$<br>$q = (5, 6, 13, 10, 4, 12, 9, 15, 2, 1, 14, 7, 11, 3, 8, 0)$ | 17 | 33 | 22 | 19 | 40 |
| $p = (1, 2, 3, 4, 5, 0, 7, 8, 9, 10, 11, 6, 13, 12, 15, 14)$<br>$q = (2, 1, 12, 8, 7, 14, 5, 4, 13, 11, 10, 15, 9, 3, 6, 0)$ | 17 | 50 | 33 | 29 | 26 |
| $p = (1, 2, 3, 4, 5, 0, 7, 8, 6, 10, 11, 9, 13, 12, 14, 15)$<br>$q = (7, 6, 14, 9, 11, 15, 1, 12, 2, 13, 5, 4, 10, 8, 3, 0)$ | 17 | 50 | 33 | 29 | 26 |

Table 1.6: Security evaluation for the best equivalence classes with $k = 16$

| $(p, q)$ | Imp. Diff | $S_{73}^{4,2}$ | $S_{49}^{8,6}$ | $S_{42}^{8,7}$ | $N_{20}$ |
|---|---|---|---|---|---|
| $p = (1, 2, 3, 4, 5, 6, 7, 0, 9, 10, 11, 12, 13, 14, 15, 8, 16, 17)$<br>$q = (10, 9, 14, 12, 15, 11, 13, 17, 2, 1, 6, 4, 7, 3, 5, 16, 8, 0)$ | 17 | 31 | 22 | 19 | 44 |
| $p = (1, 2, 3, 4, 5, 6, 7, 0, 9, 10, 11, 12, 13, 14, 15, 8, 16, 17)$<br>$q = (14, 8, 12, 15, 13, 10, 9, 17, 7, 6, 16, 3, 5, 1, 4, 2, 11, 0)$ | 17 | 56 | 38 | 32 | 26 |
| $p = (1, 2, 3, 4, 5, 6, 7, 0, 9, 10, 11, 12, 13, 14, 15, 8, 17, 16)$<br>$q = (2, 1, 6, 12, 15, 3, 13, 16, 10, 9, 14, 4, 7, 11, 5, 17, 8, 0)$ | 17 | 31 | 22 | 19 | 44 |
| $p = (1, 2, 3, 4, 5, 6, 7, 0, 9, 10, 11, 12, 13, 14, 15, 8, 17, 16)$<br>$q = (11, 5, 9, 12, 2, 7, 6, 16, 3, 13, 1, 4, 10, 15, 14, 17, 8, 0)$ | 17 | 56 | 38 | 32 | 26 |

Table 1.7: Security evaluation for the best equivalence classes with $k = 18$

*"You must look beyond what you see."*

— Rafiki

# Chapter 2

# Linearly Equivalent S-boxes and the Division Property

In this chapter, we show that previous automatic search tools dedicated to division property are incomplete in the sense they do not exhaust all the search space. More precisely, propagating an initial division property through a block cipher requires decomposing the block cipher into small components for which we can compute division property propagation. However, contrary to differential and linear cryptanalysis, the result highly depends on how the block cipher is represented. Indeed, linearly equivalent Sboxes do not change the propagation of differentials, while it is not the case for the division property. Hence, given an S-box based block cipher, it is not clear which representation should be preferred since replacing any internal S-box with a linearly equivalent one could possibly lead to a different result. The main issue is that the number of distinguishers is significantly higher than one can be thinking and looking efficiently for the best distinguisher boils down to efficiently finding the best decomposition.

We solved a sub-case of this problem. Mounting an attack against a block cipher $E$ most often requires to split $E$ in three parts as $E = E_2 \circ E_1 \circ E_0$ and to find a distinguisher on $E_1$. Usually, $E_0$, $E_1$ and $E_2$ are round-reduced versions of $E$. However it is not the only way to split $E$ and, for any linear operations $L_{in}$ and $L_{out}$, $E$ can be split as $E = (E_2 \circ L_{out}^{-1}) \circ (L_{out} \circ E_1 \circ L_{in}) \circ (L_{in}^{-1} \circ E_0)$. One of the main problem we solve in this chapter is to answer the question of how to find $L_{in}$ and $L_{out}$ such that there exists a division property distinguisher through $L_{out} \circ E_1 \circ L_{in}$, focussing on block diagonal linear mappings $L_{in}, L_{out}$. In a nutshell, we first show how to highly reduce the number of candidates for both $L_{in}$ and $L_{out}$, and then present how to efficiently check the remaining candidates without performing a complete search on each of them. As a result we improve the best known division property distinguisher against RECTANGLE by one round and show that the previous best known distinguisher against PRESENT cannot be improved with this technique.

The second result presented in this chapter concerns the design of S-boxes that would offer maximal resistance against division property. In [37], Boura et al. provide new insights into the division property, and in particular they show several interesting results concerning the resistance of S-box-based block ciphers against division property. Here we prove that if an S-box satisfies a specific criteria (which is close to the one in [37]), then this S-box is *optimal* in term of resistance against classical division property (i.e. without our extension technique). To our knowledge, this is the first time that such a result is given for division property, and could be considered as a new criteria for designing S-boxes.

According to this criteria on S-boxes, we try to strengthen both RECTANGLE and PRESENT against our technique. Note that when considering our technique, the criteria mentioned above does not seem to guarantee optimality. However, in regards to our experiments, it still seems to be the best choice, as we were able to find linearly equivalent S-boxes for both RECTANGLE and PRESENT such that the resistance of both algorithm is improved by two rounds, even when considering our extension technique.

## 2.1 Introduction

Division property is a distinguishing property which was first presented by Todo at Eurocrypt'15 [136]. This cryptanalysis technique quickly became a hot topic in the community, especially since it led to the first theoretical attack against full MISTY1 [135]. This property can be seen as a generalization of integral and higher-order differential distinguishers. At Crypto'16, Boura et al. [37] provided a simpler formulation of the division property, especially for the construction of the division trails of S-boxes. Recently, division property was used to improve cube attacks and allowed to improve the best known results against several stream ciphers including ACORN, Trivium, Grain-128a and Kreyvium [139]. The idea of the division property is the same as in integral, higher-order differential and cube-attacks, namely, proving that if one encrypts a set of plaintexts with a certain structure, then the resulting set of ciphertexts will have some balanced bits, i.e. bits which sum to zero with probability 1 when going through the whole set of ciphertexts. The main difference between these different techniques comes from how one can prove this property. Division property is a more fine-grained property: it mainly comes down to tracking which monomials may or may not appear in the Algebraic Normal Form (ANF) of the whole block cipher, so that, for a set of ciphertexts $\mathbb{X}$, we can predict with probability 1 the result of $\bigoplus_{\boldsymbol{x}\in\mathbb{X}} \boldsymbol{x}^{\boldsymbol{k}}$, where $\boldsymbol{k}$ represents the indicator vector defining the value of the monomial $\boldsymbol{x}^{\boldsymbol{k}} = \prod_i x_i^{k_i}$. The distinguisher begins by generating a set of plaintexts where $c$ bits are fixed to an arbitrary constant, resulting in $n-c$ variables which then take all possible values (thus generating an affine space of dimension $n-c$). We want to know whether a monomial of degree $n-c$, i.e. implying all variables, exists in each component of the ANF of the block cipher. If no such monomials appear in the $i$-th component, this component is of degree $< n-c$. Consequently, when we sum the $i$-th bit through the whole corresponding set of ciphertexts, the sum will be zero, as we compute the sum of a function of degree $< n-c$ over a space of dimension $n-c$. Essentially, the division property defines a set $\mathbb{K} \subset \mathbb{F}_2^n$ of monomials which divides $\mathbb{F}_2^n$ into two parts. For one part $\overline{\mathbb{K}} = \{\boldsymbol{k} |\ \exists \bar{\boldsymbol{k}} \in \mathbb{K}\ s.t.\ \bar{\boldsymbol{k}} \preceq \boldsymbol{k}\}$, for $\preceq$ the usual preceding order, we cannot predict the result of this sum. However, for any $\boldsymbol{k} \in \mathbb{F}_2^n \setminus \overline{\mathbb{K}}$, we know that $\bigoplus_{\boldsymbol{x}\in\mathbb{X}} \boldsymbol{x}^{\boldsymbol{k}} = 0$, i.e. we track which monomials we know to be summing to zero.

**Automatic tools.** Studying the propagation of an initial division property through a block cipher is a challenging task requiring to be computer-aided. At Asiacrypt'16, Xiang et al. [143] showed how to model the division property propagation of the three basic operations **copy**, **AND** and **XOR**, as well as the propagation through an S-box, by a system of linear inequalities. Hence they built MILP models for several block ciphers which they efficiently solved using a third-party MILP solver. As a result they obtained the best known division property distinguishers on SIMON, SIMECK, PRESENT and RECTANGLE. In [146], Zhang et al. gave a new way to model the propagation of division property through linear diffusion layers by the smallest amount of inequalities which are generated from linear combinations of row vectors of the diffusion matrix. Using this new description, they found the best known distinguishers for both PRINCE and MIDORI. Finally, at Asiacrypt'17, Sun et al. [129] presented two new automatic search tools: one dedicated to ARX ciphers based on a SAT solver and one dedicated to word-based division property based on SMT (Satisfiability Modulo Theories) solver. Those tools are much faster than previous MILP-based works and were able to study primitives with large internal state such as CLEFIA, WHIRLPOOL and RIJNDAEL.

**Our contributions.** In this chapter we show that previous automatic search tools dedicated to division property are incomplete in the sense they do not exhaust all the search space. More precisely, propagating an initial division property through a block cipher requires decomposing the block cipher into small components such as **AND**, **XOR**, S-boxes, ... for which we can compute

division property propagation. However, division property behaves very differently than some others cryptanalysis techniques like differential and linear cryptanalysis, for which the representation of the block cipher does not matter. For instance, in Section 2.3.1 we give two S-boxes $S_1$ and $S_2$ such that $S_2 = S_1 \circ L$, where $L$ is linear, such that propagating division property through $L$ then $S_1$ leads to a completely different result than propagating it directly through $S_2$. This would not be the case for e.g. differentials, where propagating either through $L$ and $S_1$ leads to the same result as propagating through $S_2$. As such, and considering that there are many ways to represent a block cipher, it is not clear which representation should be used to get the best results, and there may be much more possible distinguishers when considering all these representations. Thus, the main problem is to efficiently find the best representation to analyze a specific block cipher.

In this chapter we solved a sub-case of this problem. Mounting an attack against a block cipher $E$ most often requires to split $E$ in three parts as $E = E_2 \circ E_1 \circ E_0$ and to find a distinguisher on $E_1$. Usually, $E_0$, $E_1$ and $E_2$ are round-reduced versions of $E$. However it is not the only way to split $E$ and, for any linear operations $L_{in}$ and $L_{out}$, $E$ can be split as:

$$E = (E_2 \circ L_{out}^{-1}) \circ (L_{out} \circ E_1 \circ L_{in}) \circ (L_{in}^{-1} \circ E_0).$$

This kind of carving was for instance used in [51] by Derbez et al. to provide several new meet-in-the-middle attacks against AES. However, the division property is different from differential and linear cryptanalysis. Hence, one of the main problem we solved in this chapter is to answer the question of how to find $L_{in}$ and $L_{out}$ such that there exists a division property distinguisher through $L_{out} \circ E_1 \circ L_{in}$. We focused on linear mappings $L_{in}, L_{out}$ which are block diagonal, of block size $m$, where $m$ is the size of the S-box. In a nutshell, we first show how to highly reduce the number of candidates for both $L_{in}$ and $L_{out}$, and then present how to efficiently check the remaining candidates without performing a complete search on each of them. We severely reduce the complexity of the search. Indeed, to search for a distinguisher over $r$ rounds, a naive algorithm would need about $ms^3 2^{2m^2}$ (where $s$ in the number of S-boxes) calls to the MILP solver with a model representing $r$ rounds of the block-cipher. However in our case, we only need about $s^2 2^{2m}$ calls to the solver, on a model representing $r-2$ rounds which is thus much more efficient to solve. As a result we improve the best known division property distinguisher against RECTANGLE by one round and show that the previous best known distinguisher against PRESENT cannot be improved with this technique. We emphasize that this is an advantage to our algorithm, as it allows us to *prove* that a given cipher is resistant to our technique, as proving negative results is in general harder than findings attacks since we have to check all such attacks.

The second result presented in this chapter concerns the design of S-boxes that would offer maximal resistance against division property. In [37], Boura et al. provide new insights into the division property, presenting a new approach to it. In particular they show several interesting results concerning the resistance of S-box-based block ciphers against division property. Here we prove that if an S-box satisfies a specific criteria (which is close to the one in [37]), then this S-box is *optimal* in term of resistance against classical division property (i.e. without our extension technique). We define optimality in the sense that if one uses such a *perfect* S-box and can find a distinguisher on at most $r^\star$ rounds, then using any other S-box will lead to a distinguisher on $r$ rounds with $r^\star \leq r$. To our knowledge, this is the first time that such a result is given for division property, and could be considered as a new criteria for designing S-boxes. Our criteria is the following : if each component of the ANF of an $n$-bit S-box contains *all* monomials of degree $n-1$, then this S-box is optimal. Note that our criteria is equivalent to a very specific structure for the division property propagation table, and this table is a major component in the existing search algorithms [146, 143]. Compared to the criteria in [37], we have two major differences. The first is that any S-box satisfying our criteria

*does not* satisfies the one in [37]. Indeed, their criteria is that *any* non-trivial linear combination of the components of the ANF must be of degree $n-1$. In our case, since all monomials of degree $n-1$ appear in each component, the sum of any two components will be of degree $n-2$. Nonetheless, the second difference is that our criteria leads to an optimality proof, whereas their criteria is more of an indication that an S-box satisfying it *should* be good enough.

According to this criteria on S-boxes, we try to strengthen both RECTANGLE and PRESENT against our technique. Note that when considering our technique, the criteria mentioned above does not seem to guarantee optimality. However, in regards to our experiments, it still seems to be the best choice. Indeed, to preserve some differential and linear property of the original S-boxes, we chose to only consider S-boxes which are linearly equivalent to the original ones. Unfortunately, for both RECTANGLE and PRESENT, it was not possible to generate a *perfect* S-box from linearly equivalent S-boxes, however, we found many *almost perfect* S-boxes. Trying all of them allowed us to find a linearly equivalent S-box for RECTANGLE such that the best distinguisher is over 7 rounds for classical division property, and 8 rounds when using our technique, while the best distinguisher we found with our technique is over 10 rounds of RECTANGLE when using the original S-box. Doing the same for PRESENT, we found an S-box such that the classical division property could only lead to a distinguisher on up to 6 rounds, and up to 7 rounds with our technique, while the best know division property based distinguisher for PRESENT is over 9 rounds. Furthermore, on non-table-based implementations, the extra cost of the new S-boxes is only 2 extra XORs per S-box for PRESENT and 5 extra XORs per S-box for RECTANGLE. These experiments show that our new search process finds distinguishers against one extra round than classical search, highlighting again its interest, and also confirms that our strategy to choose these new S-boxes seems promising, as it improves the resistance of both algorithms by 2 rounds. We made our implementation available at `https://github.com/ExtendDivProp/ExtendDivProp`.

## 2.2 Background

### 2.2.1 Notations

We will use the following notations in the chapter. We denote $\boldsymbol{x} = (x_0, \dots, x_{n-1}) \in \mathbb{F}_2^n$ an $n$-bit vector over $\mathbb{F}_2$, where $x_0$ is the least significant bit. We will often write $x_0 x_1 \dots x_{n-1}$ instead of $(x_0, \dots, x_{n-1})$. We denote $w(\boldsymbol{x})$ the hamming weight of $\boldsymbol{x} \in \mathbb{F}_2^n$. We denote $\boldsymbol{e}_i$ the $i$-th unit vector, and $\mathbb{E}_m$ denotes the set of all unit vectors of size $m$, i.e. vectors of hamming weight 1. For $\boldsymbol{x}, \boldsymbol{u} \in \mathbb{F}_2^n$, we denote by $\boldsymbol{x}^{\boldsymbol{u}}$ the bit product

$$\boldsymbol{x}^{\boldsymbol{u}} = \prod_{i=0}^{n-1} x_i^{u_i}.$$

For $\boldsymbol{x}, \boldsymbol{y} \in \mathbb{F}_2^n$, we define $\boldsymbol{x} \succeq \boldsymbol{y}$ if $x_i \geq y_i$ for all $i$, where $x_i$ and $y_i$ are considered as integers. We denote $\mathcal{P}_m$ the set of all permutations over $m$ elements. We denote $GL_m(\mathbb{F}_2)$ the set of all invertible matrices of size $m \times m$ over $\mathbb{F}_2$.

### 2.2.2 Division Property and Division Trails

The division property was introduced by Todo [136] as a generalization of integral cryptanalysis, and later at FSE'16 [137], Todo and Morii defined a more refined version of it, called bit-based division property. Here, we only consider the bit-based division property, and will often refer to it directly as *division property*. As it is not relevant for this chapter, we refer the reader to the original articles for further details about the differences.

**Definition 2.2.1** (Bit-based Division Property [137]). *A set $\mathbb{X} \subset \mathbb{F}_2^n$ has the division property $D_{\mathbb{K}}^n$, where $\mathbb{K} \subset \mathbb{F}_2^n$ is a set, if for all $\boldsymbol{u} \in \mathbb{F}_2^n$, we have*

$$\bigoplus_{\boldsymbol{x} \in \mathbb{X}} \boldsymbol{x}^{\boldsymbol{u}} = \begin{cases} \textit{unknown} & \textit{if there is } \boldsymbol{k} \in \mathbb{K} \textit{ s.t. } \boldsymbol{u} \succeq \boldsymbol{k} \\ 0 & \textit{otherwise} \end{cases}$$

Note that if there are some vectors $\boldsymbol{k}, \boldsymbol{k'} \in \mathbb{K}$ such that $\boldsymbol{k} \succeq \boldsymbol{k'}$, then $\boldsymbol{k}$ can be removed from $\mathbb{K}$ because it is redundant.

A common way to study division property for a block cipher is to study the *division trails* of this cipher, which show the propagation of the division property through the basic operations composing the block cipher.

**Definition 2.2.2** (Division Trails [143]). *Let $f$ denote the round function of an iterated block cipher. Assume the input set to the block cipher has initial division property $D_{\{\boldsymbol{k}\}}^n$, and denote the division property after propagating through $i$ rounds of the block cipher (i.e. $i$ applications of $f$) by $D_{\mathbb{K}^i}^n$. Thus, we have the following chain of division property propagations :*

$$\{\boldsymbol{k}\} \triangleq \mathbb{K}^0 \xrightarrow{f} \mathbb{K}^1 \xrightarrow{f} \mathbb{K}^2 \xrightarrow{f} \ldots \xrightarrow{f} \mathbb{K}^r.$$

*Moreover, for any vector $\boldsymbol{k}^i$ in $\mathbb{K}^i (i \geq 1)$, there must exist a vector $\boldsymbol{k}^{i-1}$ in $\mathbb{K}^{i-1}$ such that $\boldsymbol{k}^{i-1}$ can propagate to $\boldsymbol{k}^i$ by the division property propagation rules. Furthermore, for $(\boldsymbol{k}^0, \boldsymbol{k}^1, \ldots, \boldsymbol{k}^r) \in \mathbb{K}^0 \times \mathbb{K}^1 \times \cdots \times \mathbb{K}^r$, if $\boldsymbol{k}^{i-1}$ can propagate to $\boldsymbol{k}^i$ for all $i \in \{1, 2, \ldots, r\}$, we call $(\boldsymbol{k}^0, \boldsymbol{k}^1, \ldots, \boldsymbol{k}^r)$ an $r$-round division trail.*

In the rest of the chapter, we will denote $\boldsymbol{k} \xrightarrow{f} \boldsymbol{k'}$ if the vector $\boldsymbol{k} \in \mathbb{F}_2^n$ can propagate to a vector $\boldsymbol{k'} \in \mathbb{F}_2^n$ through the function $f$. In the same way, $\boldsymbol{k} \xrightarrow{f} \mathbb{K}$ denotes that for all $\boldsymbol{k'} \in \mathbb{K}$, we have $\boldsymbol{k} \xrightarrow{f} \boldsymbol{k'}$.

Given the set $\mathbb{K}^r$ resulting of the propagation of an initial division property $D_{\{\boldsymbol{k}\}}^n$, we can find whether $D_{\{\boldsymbol{k}\}}^n$ allows to build an integral distinguisher using the following proposition.

**Proposition 2.2.3** ([143]). *Assume $\mathbb{X}$ is a set with division property $D_{\mathbb{K}}^n$, then $\mathbb{X}$ does not have integral property if and only if $\mathbb{K}$ contains all the $n$ unit vectors. As a result, if $\boldsymbol{e}_i \notin \mathbb{K}$, then the $i$-th bit is balanced.*

*Proof.* Suppose that the vector $\boldsymbol{e}_i$ belongs to $\mathbb{K}$. Then according to the definition of the division property, this implies that the result of the sum

$$\bigoplus_{\boldsymbol{x} \in \mathbb{X}} \boldsymbol{x}^{\boldsymbol{e_i}} = \bigoplus_{\boldsymbol{x} \in \mathbb{X}} \boldsymbol{x_i}$$

is unknown since $\boldsymbol{e}_i \succeq \boldsymbol{e}_i$ and $\boldsymbol{e}_i \in \mathbb{K}$, i.e. the $i$-th bit is not balanced. On the other hand, if we suppose that the $i$-th bit is balanced, we cannot have $\boldsymbol{e}_i \in \mathbb{K}$ as it would mean that the $i$-th bit is in an unknown state, which contradicts the definition of the division property. □

For example, we can make a parallel with the well known Square attack on AES [46]. In this attack, the set of plaintexts has one byte taking all possible values while the others are constant. In term of division property, this would translate to the set of plaintexts having a division property $D_{\boldsymbol{k}}^{128}$, where

$$\boldsymbol{k} = \underbrace{11111111}_{8 \text{ bits}} 0 \ldots 0.$$

Then, it is shown in [46] that after 3 rounds of AES, such a set of plaintexts has all its bits balanced. According to Proposition 2.2.3, this means that the resulting set has a division property $D_{\mathbb{K}}^{128}$, where $\mathbb{K}$ does not contain any unit vector.

Hence, to study whether we can build an integral distinguisher over a block cipher from a given initial division property $\mathbb{K}^0$, we need to propagate $\mathbb{K}^0$ through the different operations of the block cipher. Fortunately, propagation rules were defined in [137] for most basic operations in a block cipher, namely Copy, AND and XOR. However, for SPN block ciphers, there are two main components that, while they can be described using only these operations, should have their own way to propagate the division property vectors. These components are linear layers and S-boxes. For linear layers, while [128] proposed to use only the Copy and XOR operations to propagate division property vectors, it has been shown in [146] that this is actually not the right way to propagate through linear layers, as it looses some information and is not able to recover all possible integral distinguishers. We thus refer the reader to [146] for the correct way to propagate division property vectors through a given linear layer.

For S-boxes, again using only the basic operations might result in a loss of information. Hence, [143] proposed an algorithm of complexity $\mathcal{O}\left(2^{2m}\right)$ to compute all possible pairs $\boldsymbol{k} \xrightarrow{S} \boldsymbol{k'}$ for a given $m$-bit S-box $S$.

### 2.2.3 Searching for division property based integral distinguishers

While Todo and Morii proposed a way to search for integral distinguishers based on the division property [137], its complexity is quite hard to estimate, and the authors gave an upper bound of $2^n$, where $n$ is the block size of the block cipher. In practice, they said that their algorithm is not suitable for block ciphers with block size over 32 bits, and thus especially for standard block size of 64 and 128 bits. However, a lot of work has been done towards efficiently searching such distinguishers, based on either MILP [127, 143, 146] or SAT/SMT solver [61, 129]. We refer the reader to these papers for further details about the modeling, and will only give a brief description of the idea behind it for MILP. Note that using SAT/SMT solvers is very similar to using MILP, and mostly differs in efficiency when considering different primitives. For example searching division property based integral distinguishers on ARX ciphers seems to be easier with SAT solvers. First we briefly recall what is MILP.

**Definition 2.2.4.** *An MILP problem is formulated as follows. Given a matrix $A \in \mathbb{R}^{m \times n}, b \in \mathbb{R}^m$ and $c \in \mathbb{R}^n$, find a vector $x \in \mathbb{Z}^k \times \mathbb{R}^{n-k}$ with $Ax \leq b$ which minimize (or maximize) the value of*

$$f(x) = c_1 x_1 + c_2 x_2 + \cdots + c_n x_n.$$

*Here, $f$ is called the objective function of the MILP problem.*

#### Modelizing division property propagation with MILP.

The idea of using MILP to search for integral distinguishers is first to modelize the set of all possible division trails by an MILP problem. That is, building a set of linear inequalities such that [143] :

1. each division trail must satisfy all linear inequalities in the linear equality system, i.e. each division trail corresponds to a feasible solution of the linear inequality system ;

2. each feasible solution of the linear inequality system corresponds to a division trail, i.e. the set of all feasible solutions of the linear inequality system does not contain any impossible division trail.

We can thus build an MILP model satisfying the previous conditions using [143] for basic operations and S-boxes, [146] for linear layers and [127] for ARX block ciphers. Note that this step is not totally free.

For S-boxes, we first compute the set of all possible propagations through a given $m$-bit S-box, which has complexity $\mathcal{O}\left(2^{2m}\right)$. Then, we need to compute a set of linear inequalities which represents these possible propagations, according to the two previous rules. To do so, [143] proposed to first use the function `inequality_generator()` from the Sagemath [134] software to get such a set of inequalities, and then use a greedy algorithm to reduce their number. While this works for small S-boxes (e.g. 4-bit S-boxes), this approach fails when considering bigger S-boxes (e.g. 8-bit S-boxes) as the complexity of generating the initial set of inequalities is too high. However, Abdelkhalek et al. showed a new method in [1] to tackle this problem, and thus proposed a way to modelize 8-bit S-boxes in MILP. Note that while this allows us to modelize 8-bit S-boxes, it often leads to a lot of inequalities, thus the resulting model can be quite huge and this can result in a high solving time.

For linear layers, Zhang et al. [146] showed that the previous method [128] proposed to modelize linear layers does not actually fulfill the above rules, as it introduces some impossible propagations, resulting in some integral distinguishers being omitted. Hence, they proposed a new way to modelize such layers, and proved that their way was optimal, i.e. removing any one inequality will result in some fraudulent propagations. To modelize a given linear layer $L$, the number of inequalities generated is given by $n(2^s - 1)$, where $s$ is the size of the smallest square matrix $M$ such that $M$ is the representation of $L$ over the field $\mathbb{F}_{2^n}$ and $M$ is *binary*. For example, the matrix used in `SKINNY64` [10] is a binary matrix of size 4 over $\mathbb{F}_{2^4}$, thus needs $4(2^4-1) = 60$ inequalities. However, if we take the matrix used in AES, which is described as a *non-binary* matrix of size 4 over $\mathbb{F}_{2^8}$, the amount of inequalities is much higher. Indeed, since the multiplication over $\mathbb{F}_{2^8}$ corresponds to a linear operation over $\mathbb{F}_2^8$, the matrix used in AES can be represented as a matrix of size 32 over $\mathbb{F}_2$, which is obviously binary. This is the smallest way to represent this operation with a binary matrix, and thus, it would need $2^{32} - 1$ inequalities to modelize only *one* propagation through this linear layer, which would result in a very huge model which cannot be solved in practical time. Hence, not all linear layers can be modelized in an exact way, and complex linear layers may lead to a model which is much harder to solve. Note however that if the linear layer is only a permutation, such as in PRESENT [33] or RECTANGLE [145], then the above formula does not apply, as we can just reorder the different variables, and thus we can always modelize such kind of linear layers.

**Searching for a distinguisher.**

As a result, we have a set of variables $\{\mathbf{k}_i^j, i \in \{0, \dots, n-1\}, j \in \{0, \dots, r\}\}$ such that, for a given solution of the MILP problem, the corresponding values of these variables give a division trail $(\boldsymbol{k}^0, \boldsymbol{k}^1, \dots, \boldsymbol{k}^r)$ with $\boldsymbol{k}^i = (\mathbf{k}_0^i, \dots, \mathbf{k}_{n-1}^i)$. In particular, this allows us to see whether each unit vector belongs to $\mathbb{K}^r$. Indeed, once we have the MILP model for $r$ rounds of a given block cipher, we can set the objective function to $\mathbf{k}_0^r + \cdots + \mathbf{k}_{n-1}^r$. Then we set the initial division property using equality constraints, i.e. if the initial division property is $\boldsymbol{a} \in \mathbb{F}_2^n$, we add the constraints

$$\forall i \in \{0, \dots, n-1\}, \mathbf{k}_i^0 = a_i,$$

and then ask the solver (e.g. Gurobi [74]) to solve this problem by minimizing the value of the objective function. If the solver finds a solution of value 1, there is a vector $\boldsymbol{k}^r$ of weight 1 (i.e. a unit vector) that belongs to $\mathbb{K}^r$. We can then add a linear constraint to remove this vector $\boldsymbol{k}^r$ from the set of solutions, and solve the problem again. Once there are no more solutions of value 1, we found all unit vectors belonging to $\mathbb{K}^r$, hence we can easily see whether or not there are some

balanced bits using Proposition 2.2.3. Note that we do not need to stop after finding all solutions of value 1. Indeed, we can keep going until the problem does not have any remaining solutions, and we will thus have computed the whole $\mathbb{K}^r$ set. This will be useful later in the chapter, and will be accompanied with a bit more details.

## 2.3 Extended Division Property Using Linear Mappings

### 2.3.1 First observations

Several integral distinguishers were found using the previously described method. However, we claim that this method does not actually search through the whole space of all possible integral distinguishers based on the division property. Indeed, we show that for a given block cipher $E$, we can instead consider $L_{out} \circ E \circ L_{in}$, where both $L_{out}$ and $L_{in}$ are linear mappings, and this results in integral distinguishers previously unknown. We now explain the main idea behind using $L_{out}$ and $L_{in}$. For $L_{out}$, while all bits could be unbalanced after $E$, it might occur however that a linear combination of some bits is balanced. This was already mentioned by Todo and Morii in [137] when they introduced the *division property using three subsets*.

For $L_{in}$, the idea is very close. The initial division property $\boldsymbol{k}^0$ basically sets some constant bits. That is, if the set $\mathbb{X}$ has division property $D^n_{\boldsymbol{k}^0}$, through all the set, each bit $i$ such that $\boldsymbol{k}^0_i = 0$ has a constant value, and if $\boldsymbol{k}^0_i = 1$, the bit $i$ takes all possible values through the set. For example, the following set has division property $D^4_{0011}$

$$\mathbb{X} = \{0100, 0101, 0110, 0111\}.$$

Hence, the idea behind $L_{in}$ is to get a set such that a linear combination of some bits is constant, while those bits are not necessarily constant.

Finally, we can see that considering $L_{out} \circ E \circ L_{in}$ instead of $E$ is still meaningful. Classically, when an attacker uses a distinguisher to mount an attack, he basically splits the cipher $E$ into $E = E_2 \circ E_1 \circ E_0$, where he has a distinguisher over $E_1$. In that case, $E_1$ can be seen as a reduced version of $E$, containing only a certain number of rounds of $E$. However, we could also rewrite $E$ as

$$E = (E_2 \circ L_{out}^{-1}) \circ (L_{out} \circ E_1 \circ L_{in}) \circ (L_{in}^{-1} \circ E_0).$$

In that case, the attacker would search a distinguisher over $L_{out} \circ E_1 \circ L_{in}$, and could still use it to mount an attack. Indeed, the attacker starts with a set $\mathbb{X}$ respecting a given initial division property (according to the distinguisher over $L_{out} \circ E_1 \circ L_{in}$) and compute $\mathbb{X}' = E_0^{-1} \circ L_{in}(\mathbb{X})$ by guessing part of the key. He then asks for the encryption of $\mathbb{X}'$ through $E$ to get a set of ciphertexts $\mathbb{Y}$, compute $\mathbb{Y}' = L_{out} \circ E_2^{-1}(\mathbb{Y})$ using some other key guesses and check whether $\mathbb{Y}'$ has some balanced bits (according to the distinguisher over $L_{out} \circ E_1 \circ L_{in}$). If that is the case, the key guesses are supposed to be correct. Note that this idea was already successfully used in the past, for example in [51].

So considering $E' = L_{out} \circ E \circ L_{in}$ instead of $E$ could lead to some new integral distinguishers. In the following, $E$ is an SPN block cipher, i.e. the round function of $E$ is $f = \mathcal{L} \circ \mathcal{S}$, where $\mathcal{L}$ is linear and $\mathcal{S}$ is the parallel application of an S-box $S$ over the state. Note that we omit $\mathcal{L}$ in the last round. Now our goal is to search if $E'$ has an integral distinguisher based on the division property using MILP. Classically, we study the following propagation chain

$$\mathbb{K}_0 \xrightarrow{L_{in}} \widehat{\mathbb{K}}^0 \xrightarrow{\mathcal{S}} \widetilde{\mathbb{K}}^0 \xrightarrow{\mathcal{L}} \mathbb{K}^1 \xrightarrow{\mathcal{S}} \ldots \xrightarrow{\mathcal{S}} \widehat{\mathbb{K}}^r \xrightarrow{L_{out}} \mathbb{K}^r.$$

Basically, we model independently the propagation through the linear layers and the S-box layers, especially for $L_{in}$ and the first S-box layer, and for $L_{out}$ and the last S-box layer. However, this might actually not be the best way to modelize this, and we see this through an example.

**Merging linear mappings and S-boxes**

Let $S_1$ and $S_2$ be two S-boxes over $\mathbb{F}_2^4$ such that

| $x$ | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| $S_1(x)$ | 12 | 13 | 11 | 9 | 6 | 0 | 5 | 10 | 3 | 2 | 8 | 4 | 15 | 7 | 14 | 1 |
| $S_2(x)$ | 12 | 11 | 14 | 15 | 1 | 7 | 13 | 9 | 10 | 0 | 2 | 4 | 3 | 8 | 5 | 6 |

where $S_2$ is obtained as $S_2 = S_1 \circ L$ with

$$L = \begin{bmatrix} 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 \\ 0 & 1 & 1 & 0 \end{bmatrix}.$$

We can use the algorithm from [143] to compute all possible propagations through $S_1, S_2$ and $L$. Using this, if we look at the propagation of $\boldsymbol{x} = 0111$ through $L$ and $S_1$ independently, we have the following trail

$$0111 \xrightarrow{L} \{1101, 1011\} \xrightarrow{S_1} \{0100, 0010, 0001\} = \mathbb{K}.$$

However, if we now consider $L$ and $S_1$ together, i.e. by looking at the propagation of 0111 through $S_2 = S_1 \circ L$, then we have the trail

$$0111 \xrightarrow{S_2} \{1100, 1001, 0110, 0011\} = \mathbb{K}'.$$

As we can see, the resulting division property set is completely different, yet comes from the same initial division property, and goes through the same function. Moreover, this is not just a local change, and not only $\mathbb{K}'$ is a set which was not reachable through only $S_1$, but the whole propagation tables of $S_1$ and $S_2$ are different, as we can see in Figure 2.1.

This clearly shows that considering both the S-box and the linear mapping *together* gives way more information about the propagation of the division property. Note that we give this example by putting a linear mapping at the input of the S-box, but similar observations can be made when considering $S$ and $L \circ S$ for some S-box $S$ and linear mapping $L$. Moreover, not only this gives more information about the propagation, but this could, and will, actually help us to find new distinguishers when considering $L_{out} \circ E \circ L_{in}$ instead of $E$.

We can see that, except when we have either the full zero or the full one vector, if we consider a division property chain $\mathbb{K}^0 \to \cdots \to \mathbb{K}^r$ of a block cipher, the weight of the vectors in each $\mathbb{K}^i$ can only decrease (or remain constant, but in practice, this is rarely the case, see Figure 2.1). Recall that if the set $\mathbb{K}^r$ contains all unit vectors (i.e. of weight 1), no integral distinguisher can be built from it. Thus, intuitively, if we want to find an integral distinguisher, we would like to have vectors of relatively high weight in each set $\mathbb{K}^i$ as long as possible.

Now consider a block cipher $E$ such that the first layer of S-boxes contains only $S_1$ as defined previously. Then from the propagation table in Figure 2.1, we can see that the output of each S-box will always be of weight 1 (except for 0000 and 1111). So after the first round, if the weight at

| Propagations of $S_1$ | |
|---|---|
| 0000 | 0000 |
| 1000 | 1000, 0100, 0010, 0001 |
| 0100 | 1000, 0100, 0010, 0001 |
| 0010 | 1000, 0100, 0010, 0001 |
| 0001 | 1000, 0100, 0010, 0001 |
| 1100 | 1000, 0100, 0010, 0001 |
| 1010 | 1000, 0100, 0010, 0001 |
| 1001 | 0100, 0010, 0001 |
| 0110 | 0100, 0010, 0001 |
| 0101 | 0100, 0010, 0001 |
| 0011 | 0100, 0010, 0001 |
| 1110 | 0100, 0001 |
| 1101 | 0100, 0010, 0001 |
| 1011 | 0100, 0010, 0001 |
| 0111 | 0100, 0010, 0001 |
| 1111 | 1111 |

| Propagations of $S_2$ | |
|---|---|
| 0000 | 0000 |
| 1000 | 1000, 0100, 0010, 0001 |
| 0100 | 0100, 0010, 0001 |
| 0010 | 1000, 0100, 0010, 0001 |
| 0001 | 1000, 0100, 0010, 0001 |
| 1100 | 0100, 0010, 0001 |
| 1010 | 1000, 0010, **0101** |
| 1001 | 1000, 0100, 0010, 0001 |
| 0110 | 0100, 0010, 0001 |
| 0101 | 0100, 0001, **1010** |
| 0011 | 0010, **1100**, **1001**, **0101** |
| 1110 | 0010, **1100**, **0101** |
| 1101 | 0100, 0001, **1010** |
| 1011 | **1100**, **1010**, **1001**, **0110**, **0011** |
| 0111 | **1100**, **1001**, **0110**, **0011** |
| 1111 | 1111 |

Figure 2.1: Propagation tables of $S_1$ and $S_2$. Vectors of weight 2 are in bold.

the input of any S-box is different from 0 and 4, we will already only have vectors of weight 1 at the output of the S-box. However, if we now consider $E \circ \mathcal{M}$, where $\mathcal{M} = (M, \ldots, M)$ apply the linear mapping $M$ on all S-box's input before the first round, then this is the same as considering the first layer of S-boxes to be built as $(S_2, \ldots, S_2)$. This time, if one carefully chooses the input division property of the S-box, he can now only have vectors of weight 2, which *could* result in a better propagation through the remaining layers of the cipher.

Clearly, considering $L_{out} \circ E \circ L_{in}$ instead of $E$, and considering the propagation of the division property vectors through $M \circ S$ (or $S \circ M$) as a whole instead of independently through $M$ and $S$, could result in better distinguishers, and thus in the next section, we focus on the search of such distinguishers.

### 2.3.2 Searching for Extended Division Property

In this chapter, we will only consider SPN block ciphers, i.e. the round function is $f = \mathcal{L} \circ \mathcal{S}$, where $\mathcal{L}$ is linear and $\mathcal{S}$ is built as the concatenation of $s$ S-boxes of size $m$ applied in parallel on the state, hence the block cipher has block size $n = s.m$. Moreover, we will consider that all S-boxes are the same. This is to get an easier analysis, but we can extend this with different S-boxes.

#### Reducing the search space of $L_{in}$ and $L_{out}$.

Given a block cipher $E$ which does not have any integral distinguisher based on the division property, we want to find two linear mappings $L_{in}$ and $L_{out}$ such that $L_{out} \circ E \circ L_{in}$ has an integral distinguisher based on the division property which is supported by the previous observations. Moreover, we also would like to exploit the fact that we have a more precise propagation when considering the propagation of division property vectors through $S \circ M$ as a single function, instead of independently through $M$ then $S$. Note that, theoretically, we could consider the whole round function of the block cipher as a single function (or even the whole block cipher), and thus get more precise information about the propagation of division property vectors. However, computing the propagation table of

division vectors needs $\mathcal{O}(2^{2n})$ operations, where $n$ is the size of the function. Hence for classical block ciphers with 64 or 128 bits block size, this is clearly impractical.

This also means that we cannot choose any $L_{in}$ and $L_{out}$, as we want to somehow merge $L_{in}$ with the first S-box layer and, respectively, merge $L_{out}$ with the last S-box layer. Hence, we will focus our search on linear maps $L_{in}$ and $L_{out}$ which are block diagonal, of block size $m$. Consequently we want to put an invertible linear map $L^i_{in}$ (resp. $L^i_{out}$) before (resp. after) each S-box of the first (resp. last) round. By doing so, we will denote by $S^i_{in} = S \circ L^i_{in}$ and $S^i_{out} = L^i_{out} \circ S$ the modified S-boxes.

First, we give the following proposition to show that we do not need to consider every possible choice for each block $L^i_{in}$ and $L^i_{out}$.

**Proposition 2.3.1.** *Let $S$ be an invertible $m$-bit S-box and $P$ an $m$-bit permutation. Let $S_1 = S \circ P$ and $S_2 = P \circ S$, and $\boldsymbol{k} \xrightarrow{S} \boldsymbol{k}'$ be any valid division property propagation through $S$ with $\boldsymbol{k}, \boldsymbol{k}' \in \mathbb{F}_2^m$. Then both propagations $P^{-1}(\boldsymbol{k}) \xrightarrow{S_1} \boldsymbol{k}'$ and $\boldsymbol{k} \xrightarrow{S_2} P(\boldsymbol{k}')$ are always valid.*

*Proof.* This directly comes from the fact that $S_1$ is obtained by just permuting the input variables of $S$, and respectively $S_2$ is obtained by permuting the output bits of $S$. □

Hence, if we search an integral distinguisher for any given block $L^i_{in}$, we do not need to do the search for all $L^i_{in} \circ P$ where $P$ goes through all possible permutations, as we could obtain the same result from the search using $L^i_{in}$ by just permuting the initial division property with $P$. For example, if we have the set $\mathbb{K}^r$ from a given initial division property $\boldsymbol{k}$ through $L_{out} \circ E \circ L_{in}$, and we consider $L'_{out} \circ E \circ L'_{in}$ where $L'_{in} = L_{in} \circ (P^0_{in}, \ldots, P^{s-1}_{in})$ and $L'_{out} = (P^0_{out}, \ldots, P^{s-1}_{out}) \circ L_{out}$, where each $P^i_{in}$ and $P^i_{out}$ is a permutation over $m$ bits, we directly have that the initial division property $(P^0_{in}, \ldots, P^{s-1}_{in})^{-1}(\boldsymbol{k})$ propagates to the set $(P^0_{out}, \ldots, P^{s-1}_{out})(\mathbb{K}^r)$. In particular, if we have an integral distinguisher for $L_{out} \circ E \circ L_{in}$, so do we for $L'_{out} \circ E \circ L'_{in}$ (and vice-versa if $L_{out} \circ E \circ L_{in}$ does not have any integral distinguisher).

This allows us to restrict the search space for each block $L^i_{in}$ to a set $\mathbb{L}_{in}$ containing a representative of each equivalence class

$$\mathcal{E}_{in}(L) = \{L' \in GL_m(\mathbb{F}_2) \mid \exists P \in \mathcal{P}_m \ s.t. \ L' = L \circ P\},$$

and in the same way, to restrict the search space of each $L^i_{out}$ to a set $\mathbb{L}_{out}$ containing a representative of each equivalence class

$$\mathcal{E}_{out}(L) = \{L' \in GL_m(\mathbb{F}_2) \mid \exists P \in \mathcal{P}_m \ s.t. \ L' = P \circ L\}.$$

The size of these spaces $\mathbb{L}_{in}$ and $\mathbb{L}_{out}$ can be obtained by

$$\frac{\prod_{i=0}^{m-1} 2^m - 2^i}{m!},$$

as it is the total number of invertible matrices of size $m$ divided by the number of permutations over $m$ elements. Note that this is much lower than the total number of matrices of size $m \times m$ over $\mathbb{F}_2$ which is $2^{m^2}$, and for example if $m = 4$, then there are only 840 matrices to consider.

**Reducing the amount of work for $L_{in}$**

Let us focus on finding a distinguisher over $E \circ L_{in}$. We will see later that we can use the idea of this section together with the next section to search for a distinguisher over $L_{out} \circ E \circ L_{in}$. Note that our goal is to exhibit a distinguisher on $E \circ L_{in}$, not necessarily the best one. As such, we focus on finding a distinguisher requiring $2^{n-1}$ data, i.e. the initial division property will be $\mathbb{K}^0 = \boldsymbol{k}^0$ with $w(\boldsymbol{k}^0) = n-1$. By doing so, we focus our search on only one modified S-box $S^i_{in}$ and set the others to $S$. Indeed, if $w(\boldsymbol{k}^0) = n-1$, there is only one specific S-box $S^i_{in}$ which has an input of weight $m-1$, while all the others S-boxes $S^j_{in}$ with $j \neq i$ have $1 \dots 1$ has input. Note that if a set $\mathbb{X}$ has division property $\boldsymbol{k} = 1 \dots 1$, all bits takes all possibles values through the set, i.e. $\mathbb{X} = \mathbb{F}_2^m$. Hence, since we are considering bijective S-boxes, we have $S^j_{in}(\mathbb{X}) = \mathbb{F}_2^m$ for all $j \neq i$, and thus the resulting division property set is $\mathbb{K} = \{1 \dots 1\}$.

From the previous remark, we only need to look at each matrix from $\mathbb{L}_{in}$. However, we can reduce even further the amount of propagation we need to compute. Since the input of the S-box $S^i_{in}$ is $\boldsymbol{k}^0_i$ with $w(\boldsymbol{k}^0_i) = m-1$, we know that this can only result in at most $2^m - 2$ possible vectors (by excluding the full-zero and full-one vectors) after the application of $S^i_{in}$. Thus, to search for a distinguisher over $E' = E \circ L_{in}$ with $E$ containing $r$ rounds with round function $f = \mathcal{L} \circ \mathcal{S}$, we first decompose $E'$ as

$$E' = f \circ f \circ \dots \circ f \circ \mathcal{L} \circ \mathcal{S}_{in}, \text{ with } \mathcal{S}_{in} = (S, \dots, S^i_{in}, \dots, S).$$

This leads to the following chain of division property propagation

$$\boldsymbol{k}^0 \xrightarrow{\mathcal{S}_{in}} \widetilde{\mathbb{K}}^0 \xrightarrow{\mathcal{L}} \mathbb{K}^1 \xrightarrow{f} \dots \xrightarrow{f} \mathbb{K}^r$$

where

$$\boldsymbol{k}^0 = \overbrace{1 \dots 1}^{m} | \overbrace{1 \dots 1}^{m} | \dots | \boldsymbol{k}^0_i | \dots | \overbrace{1 \dots 1}^{m}.$$

We first define the set $\mathcal{K}^S_{in}$ as

$$\mathcal{K}^S_{in} := \{\mathbb{K} \mid \exists L_{in} \in \mathbb{L}_{in}, \boldsymbol{k} \in \mathbb{F}_2^m \text{ s.t. } \boldsymbol{k} \xrightarrow{S^i_{in}} \mathbb{K} \text{ and } w(\boldsymbol{k}) = m-1\}.$$

Computing $\mathcal{K}^S_{in}$ allows to build all possible $\widetilde{\mathbb{K}}^0$ since there exists a set $\mathbb{K} \in \mathcal{K}^S_{in}$ such that every vector $\tilde{\boldsymbol{k}}^0$ of $\widetilde{\mathbb{K}}^0$ is of the form

$$\tilde{\boldsymbol{k}}^0 = \overbrace{1 \dots 1}^{m} | \overbrace{1 \dots 1}^{m} | \dots | \tilde{\boldsymbol{k}}^0_i | \dots | \overbrace{1 \dots 1}^{m}, \text{ with } \tilde{\boldsymbol{k}}^0_i \in \mathbb{K}.$$

Hence, instead of trying all possible $L^i_{in} \in \mathbb{L}_{in}$, we skip the first propagation through $\mathcal{S}_{in}$ and directly consider that the propagation starts at $\widetilde{\mathbb{K}}^0$.

We now need to test each set in $\mathcal{K}^S_{in}$. Recall that $\widetilde{\mathbb{K}}^0$ can only be built from $2^m - 2$ vectors $\tilde{\boldsymbol{k}}^0$. We propagate each of those vectors through the remaining layers of the cipher, i.e. the following chain of propagation

$$\tilde{\boldsymbol{k}}^0 \xrightarrow{\mathcal{L}} \mathbb{K}^1 \xrightarrow{f_1} \dots \xrightarrow{f_{r-1}} \mathbb{K}^r.$$

Thus, for each $\tilde{\boldsymbol{k}}^0$, we deduce a set $\mathbb{S}_{\tilde{\boldsymbol{k}}^0}$ of balanced bits using MILP. We then consider each set $\widetilde{\mathbb{K}}^0 \in \mathcal{K}^S_{in}$, and compute

$$\mathbb{S}_{\widetilde{\mathbb{K}}^0} = \bigcap_{\tilde{\boldsymbol{k}}^0 \in \widetilde{\mathbb{K}}^0} \mathbb{S}_{\tilde{\boldsymbol{k}}^0}.$$

If there is one non-empty $\mathbb{S}_{\widetilde{\mathbb{K}}^0}$, $\widetilde{\mathbb{K}}^0$ will lead to a set of balanced bits, given by $\mathbb{S}_{\widetilde{\mathbb{K}}^0}$.

Finally, using a precomputed table $\mathcal{T}_{in}^{S}$ defined as

$$\mathcal{T}_{in}^{S}[\mathbb{K}] := \{(L_{in}, \boldsymbol{k}) \in \mathbb{L}_{in} \times \mathbb{F}_2^m \mid \boldsymbol{k} \xrightarrow{S_{in}^i} \mathbb{K} \text{ and } w(\boldsymbol{k}) = m-1\},$$

we deduce a linear map $L_{in}^i \in \mathbb{L}_{in}$ and a vector $\boldsymbol{k}^0$ such that we get an integral distinguisher over $E \circ L_{in}$ starting from the initial division property $\boldsymbol{k}^0$.

In summary, we first propagate each of the $2^m - 2$ vectors through $f \circ \cdots \circ f \circ \mathcal{L}$. Then, for each set $\widetilde{\mathbb{K}}^0 \in \mathcal{K}_S$, we check if each vector of $\widetilde{\mathbb{K}}^0$ lead to the same balanced bits through $f \circ \cdots \circ f \circ \mathcal{L}$. If so, then using $\mathcal{T}_{in}^{S}$ we can easily deduce a linear map $L_{in}$ and an initial division property which results in an integral distinguisher.

**Reducing the amount of work for $L_{out}$.**

Again, we first only consider $L_{out} \circ E$, and will see in the next part how to combine this with the previous section to get a distinguisher over $L_{out} \circ E \circ L_{in}$. For $L_{out}$, if we search naively, we need to try each possible matrix from $\mathbb{L}_{out}$. However, this is actually not necessary. Indeed, recall that there is an integral distinguisher if and only if the last division property set $\mathbb{K}^r$ does not contain all unit vectors, and thus we only need to check if each unit vector belongs to $\mathbb{K}^r$. Now consider a division property vector $\boldsymbol{k}$ which is sent to such a unit vector $\boldsymbol{e}_i$ through the last (modified) S-box layer. That is, we have $\boldsymbol{k} \xrightarrow{\mathcal{S}_{out}} \boldsymbol{e}_i$ where $\mathcal{S}_{out} = (S_{out}^0, \ldots, S_{out}^{s-1})$. In that case, all S-boxes except one have an output division property vector equal to $0 \ldots 0$. Again, since we are using bijective S-boxes, this means that the output set is constant, and thus the input set is also constant, leading to a corresponding input vector $0 \ldots 0$. Hence, $\boldsymbol{k}$ will be of the form

$$\overbrace{0 \ldots 0}^{m} | \overbrace{0 \ldots 0}^{m} | \ldots | \tilde{\boldsymbol{k}} | \ldots | \overbrace{0 \ldots 0}^{m}$$

where $\tilde{\boldsymbol{k}}$ is a non-zero vector of $\mathbb{F}_2^m$.

Consequently, we first compute, for each $L_{out} \in \mathbb{L}_{out}$, all possible sets $\mathbb{K}$ such that $\mathbb{K} \xrightarrow{S_{out}} \mathbb{K}'$, with $S_{out} = L_{out} \circ S$ and $\mathbb{K}'$ does not contain all unit vectors over $m$ bits. According to those notations, denote by $\mathcal{K}_{out}^{S}$ the set

$$\mathcal{K}_{out}^{S} = \{\mathbb{K} \mid \exists L_{out} \in \mathbb{L}_{out} \text{ and } \mathbb{K}' \text{ s.t. } \mathbb{K} \xrightarrow{S_{out}} \mathbb{K}' \text{ and } \mathbb{E}_m \not\subset \mathbb{K}'\}.$$

We can write the division property propagation chain

$$\boldsymbol{k^0} \xrightarrow{f} \mathbb{K}^1 \xrightarrow{f} \ldots \mathbb{K}^{r-1} \xrightarrow{\mathcal{S}_{out}} \mathbb{K}^r.$$

However, we do not know which $L_{out}$ to use, and thus cannot propagate through $S_{out}$. But instead, we compute a subset $\widetilde{\mathbb{K}}$ of $\mathbb{K}^{r-1}$ such that for every vector $\boldsymbol{k}$ of $\widetilde{\mathbb{K}}$, the non-zeros elements of $\boldsymbol{k}$ all belong to a single S-box block, i.e. $\boldsymbol{k}$ is of the form

$$\overbrace{0 \ldots 0}^{m} | \overbrace{0 \ldots 0}^{m} | \ldots | \tilde{\boldsymbol{k}} | \ldots | \overbrace{0 \ldots 0}^{m}$$

with $\tilde{\boldsymbol{k}}$ a non-zero vector of $\mathbb{F}_2^m$. Thus, if there is a propagation $\boldsymbol{k} \xrightarrow{\mathcal{S}_{out}} \boldsymbol{e}$ where $\boldsymbol{e}$ is a unit vector, then we must have $\boldsymbol{k} \in \widetilde{\mathbb{K}}$. Now from $\widetilde{\mathbb{K}}$, build the following sets for each $i \in \{0, \ldots, s-1\}$

$$\mathbb{K}_i^{r-1} = \{\tilde{\boldsymbol{k}} \text{ s.t. } 0 \ldots 0 | \tilde{\boldsymbol{k}} | 0 \ldots 0 \in \widetilde{\mathbb{K}} \text{ where } \tilde{\boldsymbol{k}} \text{ is on the } i\text{-th S-box}\}.$$

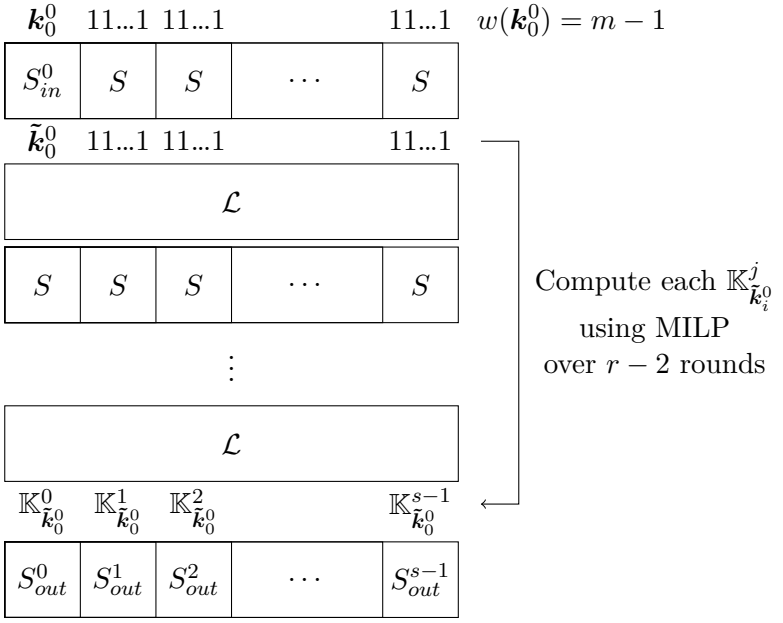


Figure 2.2: Overall framework of our search algorithm, where we search for $L_{in}^0$

These sets $\mathbb{K}_i^{r-1}$ allow us to see if we can get a distinguisher. Indeed, if for at least one $i \in \{0, \dots, s-1\}$ we have $\mathbb{K}_i^{r-1} \in \mathcal{K}_{out}^S$, then we can get a distinguisher over $L_{out} \circ E$. Then, using a precomputed table $\mathcal{T}_{out}^S$ defined as

$$\mathcal{T}_{out}^S[\mathbb{K}] = \{L_{out} \in \mathbb{L}_{out} \mid \exists \mathbb{K}' \text{ s.t. } \mathbb{K} \xrightarrow{S_{out}} \mathbb{K}' \text{ and } \mathbb{E}_m \not\subset \mathbb{K}'\},$$

we know that there exists a linear map $L_{out}^i \in \mathcal{T}_{out}^S[\mathbb{K}_i^{r-1}]$ and a unit vector $\boldsymbol{e} \in \mathbb{F}_2^m$ such that $\mathbb{K}_i^{r-1} \xrightarrow{S_{out}^i} \mathbb{K}'$ where $\boldsymbol{e} \notin \mathbb{K}'$. Hence, the unit vector $0 \dots 0|\boldsymbol{e}|0 \dots 0 \in \mathbb{F}_2^n$ will not belong to $\mathbb{K}^r$, which means that we have at least one balanced bit. In summary, to search for each block $L_{out}^i$, we just need to compute all sets $\mathbb{K}_i^{r-1}$ and check if at least one of them belongs to $\mathcal{K}_{out}^S$. If so, we can deduce from $\mathcal{T}_{out}^S$ which block $L_{out}^i$ to use such that this results in an integral distinguisher.

**Putting everything together.**

We can now combine the two previous sections to search for a distinguisher over $L_{out} \circ E \circ L_{in}$. The overall idea is given in Figure 2.2. We first write $L_{out} \circ E \circ L_{in}$ as

$$\mathcal{S}_{out} \circ \underbrace{f \circ \cdots \circ f}_{r-2 \text{ rounds}} \circ \mathcal{L} \circ \mathcal{S}_{in},$$

and get the following propagation chain

$$\boldsymbol{k}^0 \xrightarrow{\mathcal{S}_{in}} \widetilde{\mathbb{K}}^0 \xrightarrow{\mathcal{L}} \mathbb{K}^1 \xrightarrow{f} \dots \xrightarrow{f} \mathbb{K}^{r-1} \xrightarrow{\mathcal{S}_{out}} \mathbb{K}^r,$$

where $w(\boldsymbol{k}^0) = n-1$. According to the two previous sections, we first start by computing $\mathcal{K}_{in}^S, \mathcal{T}_{in}^S, \mathcal{K}_{out}^S$ and $\mathcal{T}_{out}^S$. Then, for each S-box block $i$ of the first layer, and for each of the $2^m-2$ initial division property vectors $\tilde{\boldsymbol{k}}_i^0$, we use an MILP solver to compute all the sets $\mathbb{K}_j^{r-1}, j \in \{0, \dots, s-1\}$ through $f \circ \cdots \circ f \circ \mathcal{L}$, where there are $r-2$ applications of $f$. We denote by $\mathbb{K}^j_{\tilde{\boldsymbol{k}}_i^0}$ these sets to tie them with $\tilde{\boldsymbol{k}}_i^0$.

Next for each set $\widetilde{\mathbb{K}}^0 \in \mathcal{K}_{in}^S$, we compute the following union for each $j \in \{0, \ldots, s-1\}$ :

$$\mathbb{K}^j_{\widetilde{\mathbb{K}}^0} = \bigcup_{\tilde{\boldsymbol{k}}_i^0 \in \widetilde{\mathbb{K}}_0} \mathbb{K}^j_{\tilde{\boldsymbol{k}}_i^0}.$$

Now if at least one $\mathbb{K}^j_{\widetilde{\mathbb{K}}^0}$ belongs to $\mathcal{K}_{out}^S$, then we can get a distinguisher. Indeed, $\mathbb{K}^j_{\widetilde{\mathbb{K}}^0}$ is the set of division property vectors that can lead to a unit vectors after the application of $S^j_{out}$. Thus by definition of $\mathcal{K}_{out}^S$, if $\mathbb{K}^j_{\widetilde{\mathbb{K}}^0} \in \mathcal{K}_{out}^S$ we know that at least one unit vector will not appear after the application of $S^j_{out}$, i.e. $\mathbb{K}^r$ does not contains all unit vectors. We then put any map from $\mathcal{T}^S_{out}[\mathbb{K}^j_{\widetilde{\mathbb{K}}^0}]$ after the $j$-th S-box in the last layer, and any map from $\mathcal{T}^S_{in}[\widetilde{\mathbb{K}}^0]$ before the $i$-th S-box in the first layer, which thus gives us our new distinguisher. Note that we can easily see that

$$\mathcal{T}^S_{out}[\mathbb{K}^j_{\widetilde{\mathbb{K}}^0}] = \bigcap_{\tilde{\boldsymbol{k}}_i^0 \in \widetilde{\mathbb{K}}_0} \mathcal{T}^S_{out}[\mathbb{K}^j_{\tilde{\boldsymbol{k}}_i^0}].$$

Indeed, a linear mapping lead to at least one balanced bit from $\widetilde{\mathbb{K}}^0$ if and only if it lead to at least one balanced bit from each $\tilde{\boldsymbol{k}}_i^0 \in \widetilde{\mathbb{K}}_0$. This will be used later on to even further reduce the work needed with an early-abort strategy. The whole procedure is summarized in Algorithm 2.

**Complexity.** Overall, the number of calls to the MILP solver can be upper bounded as follow. First, we need to compute all $\mathbb{K}^j_{\tilde{\boldsymbol{k}}_i^0}$ for each of the $s(2^m-2)$ possible $\tilde{\boldsymbol{k}}^0$. Then, each set $\mathbb{K}^j_{\tilde{\boldsymbol{k}}_i^0}$ can contain at most $2^m$ vectors, and getting one vector of any of these sets cost one call to the MILP solver. Since there are $s$ of those sets, we need $s2^m$ calls to the MILP solver. Note however that in practice, this is much lower, as we do not need to recover the redundant vectors. This means that for example, the sets $\{0001, 0011\}$ and $\{0001\}$ are considered to be equivalent, as 0011 is redundant in the first set and thus can be removed. If we go through all sets with $m = 4$, the maximum size of any set $\mathbb{K}^j_{\tilde{\boldsymbol{k}}_i^0}$ is 6, and there are only 167 possible sets (compared to, in theory, a maximum size of 16, and $2^{16}$ possible sets). In total, we need at most $s^2(2^m-2)2^m$ calls to the MILP solver for a model over $r-2$ rounds, and the factor $2^m$ is actually much lower in practice.

This can be compared to the complexity of a naive algorithm. In such an algorithm, one would need to try every possible invertible matrix for each S-box at the first round, so about $s2^{m^2}$ cases (a bit less as there are less than $2^{m^2}$ invertible matrices). For each of those case, we need to try again every possible matrix for each S-box at the last round, so this add another factor $s2^{m^2}$. This generate $s^2 2^{2m^2}$ models, and then for each of those, we need to check if there is a distinguisher. At most, it costs $n = sm$ calls to the MILP solver, as one call can retrieve one vector of weight 1, and there are $n$ of them. So in total, a naive algorithm would need about $ms^3 2^{2m^2}$ calls to the MILP solver, and each model is over $r$ rounds which is much more expensive to solve.

Moreover for our technique, if we go through each of the $2^m - 2$ vectors $\tilde{\boldsymbol{k}}_i^0$ in a clever way, we can often reduce further the number of calls to the MILP solver. Indeed, if we first go through all vectors of weight $m-1$ and compute all corresponding $\mathbb{K}^j_{\tilde{\boldsymbol{k}}_i^0}$, we are left with two cases :

- All sets $\mathcal{T}^S_{out}[\mathbb{K}^j_{\tilde{\boldsymbol{k}}_i^0}]$ for all vectors $\tilde{\boldsymbol{k}}_i^0$ of weight $m-1$ are empty, and thus we do not need to go further. Indeed, this means that no linear mapping lead to at least one balanced bit from any initial vector of weight $m-1$. Moreover, for any vector $\boldsymbol{k}$ such that $w(\boldsymbol{k}) < m-1$, we know that there is a vector $\tilde{\boldsymbol{k}}_i^0$ of weight $m-1$ such that $\tilde{\boldsymbol{k}}_i^0 \succeq \boldsymbol{k}$. Hence, since there is no balanced bit from all vectors $\tilde{\boldsymbol{k}}_i^0$ of weight $m-1$, then we cannot have any balanced bit from any vector of weight strictly lower than $m-1$ (see [129] Proposition 2).

- Otherwise, we first check if there is any set $\widetilde{\mathbb{K}}^0 \in \mathcal{K}_{in}$ built only from vectors of weight $m-1$. If so, we apply Algorithm 2 from line 11 to line 22 to check if we can find a distinguisher. If no distinguisher exists, or if none of the set of $\mathcal{K}_{in}$ are built only from vectors of weight $m-1$, then we go through all vectors of weight $m-2$ and do the same procedure and so on.

We can even go further by looking at all the possible transitions $\boldsymbol{k} \xrightarrow{\mathcal{S}^i_{in}} \mathbb{K}$ with $w(\boldsymbol{k}) = m-1$ when we go through all linear mappings in $\mathbb{L}_{in}$. Suppose that the two following transitions are possible (and possibly with different linear mappings), $\boldsymbol{k} \xrightarrow{\mathcal{S}^i_{in}} \mathbb{K}$ and $\boldsymbol{k'} \xrightarrow{\mathcal{S}'^i_{in}} \mathbb{K}'$, with $w(\boldsymbol{k}) = w(\boldsymbol{k'}) = m-1$. If for all vectors $\widetilde{\boldsymbol{k'}} \in \mathbb{K}'$, there exists a vector $\widetilde{\boldsymbol{k}} \in \mathbb{K}$ such that $\widetilde{\boldsymbol{k'}} \preceq \widetilde{\boldsymbol{k}}$, it is not useful to consider $\mathcal{S}'^i_{in}$. Indeed, in that case, if $\mathcal{S}'^i_{in}$ would lead to a distinguisher, then so would $\mathcal{S}^i_{in}$. Such a transition $\boldsymbol{k'} \xrightarrow{\mathcal{S}'^i_{in}} \mathbb{K}'$ is thus redundant and does not need be examined. We can thus build all possible transitions $\boldsymbol{k} \xrightarrow{\mathcal{S}^i_{in}} \mathbb{K}$ which are not redundant. If there is a vector $\widetilde{\boldsymbol{k}}$ which never belongs to $\mathbb{K}$ among all such non-redundant transitions, we never have to examine the propagation of this vector. This essentially reduces even further the space of all the vectors $\tilde{\boldsymbol{k}}^0_i$ we need to consider. In practice, this allows to significantly reduce the time required to find a distinguisher, or even prove that no such distinguisher exists, and this will be detailed in the next section.

**Algorithm 2** Searching $L_{in}$ and $L_{out}$

1: Compute $\mathcal{K}^S_{in}, \mathcal{T}^S_{in}, \mathcal{K}^S_{out}$ and $\mathcal{T}^S_{out}$
2: **for** $i = 0 \dots s-1$ **do**
3:  **for** each of the $2^m - 2$ vectors $\tilde{\boldsymbol{k}}^0_i$ **do**
4:  Generate a MILP model for $f \circ \cdots \circ f \circ \mathcal{L}$ — *$r-2$ application of $f$*
5:  Set the initial division property to $1 \dots \tilde{\boldsymbol{k}}^0_i \dots 1$ — *$\tilde{\boldsymbol{k}}^0_i$ on the $i$-th block*
6:  Compute each set $\mathbb{K}^j_{\tilde{\boldsymbol{k}}^0_i}, j \in \{0, \dots, s-1\}$ using the MILP model
7:  **end for**
8:  **for** $\widetilde{\mathbb{K}}^0 \in \mathcal{K}^S_{in}$ **do**
9:  $L^0_{out}, \dots, L^{s-1}_{out} \leftarrow I_m$ — *$I_m$ : identity matrix of size $m$*
10:  *found_distinguisher* $\leftarrow$ *false* — *boolean to check if we found a distinguisher*
11:  **for** $j = 0 \dots s-1$ **do**
12:  Compute $\mathbb{K}^j_{\widetilde{\mathbb{K}}^0}$
13:  **if** $\mathbb{K}^j_{\widetilde{\mathbb{K}}^0} \in \mathcal{K}^S_{out}$ **then**
14:  $L^j_{out} \leftarrow$ any element from $\mathcal{T}^S_{out}[\mathbb{K}^j_{\widetilde{\mathbb{K}}^0}]$
15:  *found_distinguisher* $\leftarrow$ *true*
16:  **end if**
17:  **end for**
18:  **if** *found_distinguisher* **then**
19:  $L^i_{in} \leftarrow$ any element in $\mathcal{T}^S_{in}[\widetilde{\mathbb{K}}^0]$
20:  **return** $Diag(I_m, \dots, L^i_{in}, \dots, I_m), Diag(L^0_{out}, \dots, L^{s-1}_{out})$
21:  **end if**
22:  **end for**
23: **end for**

## 2.4 Application

### 2.4.1 Division Property against 10-round RECTANGLE

RECTANGLE [145] is a lightweight block cipher designed for fast implementation using bit-slice techniques. It is a 64-bit block cipher, using 4-bit S-boxes and a permutation as the linear layer. There are 80-bit and 128-bit key sizes, and the total number of round is 25 in both cases. The best known division property based integral distinguisher is from [143] over 9 rounds, using $2^{60}$ data and resulting in 16 balanced bits. By applying the previous algorithm, we were able to find a distinguisher over 10 rounds, using $2^{63}$ data and resulting in 1 balanced bit. The distinguisher is built on $L_{out} \circ E \circ L_{in}$, where the block 0 of $L_{in}$ is

$$L_{in}^0 = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

and $L_{out}$ is the identity. This results in the following distinguisher, where c denotes a constant bit, a denotes a bit taking all possible values through the set, b denotes a balanced bit and ? denotes a bit in an unknown state.

$$\text{Input : } \begin{pmatrix} \texttt{aaaaaaaaaaaaaaaa} \\ \texttt{aaaaaaaaaaaaaaaa} \\ \texttt{aaaaaaaaaaaaaaa}\mathbf{\texttt{c}} \\ \texttt{aaaaaaaaaaaaaaaa} \end{pmatrix} \rightarrow \text{Output : } \begin{pmatrix} \texttt{??????????}\mathbf{\texttt{b}}\texttt{?????} \\ \texttt{????????????????} \\ \texttt{????????????????} \\ \texttt{????????????????} \end{pmatrix}$$

Overall, the time needed to compute all $\mathbb{K}^j_{\tilde{\boldsymbol{k}}_i^0}$ for a given $\tilde{\boldsymbol{k}}_i^0$ is about 400 seconds in average. The reason this distinguisher exists is that when considering $S' = S \circ L_{in}^0$ where $S$ is the S-box of RECTANGLE, the transition $1101 \xrightarrow{S'} \{0101, 1110\}$ is now possible, while the set $\{0101, 1110\}$ was not reachable from the original S-box $S$. Note that this distinguisher does not depend on the key size, and thus is applicable to both the 80-bit and the 128-bit key variants.

### 2.4.2 Strengthening RECTANGLE

According to our observations in Section 2.3.1, it is natural to think that the resistance of an S-box-based cipher against division property is highly related to the number of weight 1 vectors in the division property propagation table of the S-box. As such we study how the choice of the S-box affects the resistance of RECTANGLE against division property. We first give some generic insights about the design of an S-box to resist classical division property (i.e. without using our extension technique). Before going further, let us recall how the division property propagation table is built.

**Proposition 2.4.1** ([143])**.** *Let $S$ be an $n$-bit S-box with $y = (y_0, \ldots, y_{n-1})$ the ANF of $S$ , where each $y_i$ is a polynomial in the input variables $x = (x_0, \ldots, x_{n-1})$ of $S$. For some $\boldsymbol{k} \in \mathbb{F}_2^n$, let $\mathbb{U}_{\boldsymbol{k}} = \{\bar{\boldsymbol{k}} \in \mathbb{F}_2^n | \boldsymbol{k} \preceq \bar{\boldsymbol{k}}\}$ and $F_{\boldsymbol{k}} = \{x^{\bar{\boldsymbol{k}}} | \bar{\boldsymbol{k}} \in \mathbb{U}_{\boldsymbol{k}}\}$. Then we have the transition $\boldsymbol{k} \xrightarrow{S} \boldsymbol{k'}$ if and only if $y^{\boldsymbol{k'}}$ contains a monomial in $F_{\boldsymbol{k}}$.*

Intuitively, an S-box such that all vectors in the propagation table are of weight 1 should provide a good resistance against division property. This leads us to define a *perfect* S-box, where the choice of the word *perfect* will be justified in Theorem 2.4.6.

**Definition 2.4.2.** *Let $S$ be an $n$-bit S-box. We say that $S$ is perfect (w.r.t division property) if its division property propagation table is of the following form :*

- $0\dots0 \xrightarrow{S} \mathbb{K} = \{0\dots0\}$,
- $1\dots1 \xrightarrow{S} \mathbb{K} = \{1\dots1\}$,
- *For any other* $\boldsymbol{k} \in \mathbb{F}_2^n$, $\boldsymbol{k} \xrightarrow{S} \mathbb{E}_n$.

Note that, from Proposition 2.4.1 this also means that if $S$ is a perfect S-box, for any $\boldsymbol{k} \in \mathbb{F}_2^n \setminus \{0\dots0, 1\dots1\}$, the transition $\boldsymbol{k} \xrightarrow{S} \boldsymbol{k'}$ is always valid for any $\boldsymbol{k'} \in \mathbb{F}_2^n \setminus \{0\dots0\}$. However, most vectors will be redundant, i.e. vectors $\boldsymbol{k'}, \boldsymbol{k''} \in \mathbb{F}_2^n$ such that $\boldsymbol{k} \xrightarrow{S} \boldsymbol{k'}$, $\boldsymbol{k} \xrightarrow{S} \boldsymbol{k''}$ and $\boldsymbol{k'} \preceq \boldsymbol{k''}$. Since we do not need to consider redundant vectors in the division property propagation table, we still write $\boldsymbol{k} \xrightarrow{S} \mathbb{E}_n$.

One can wonder wether such S-box exists, and consequently, we give a clear characterization for perfect S-boxes.

**Proposition 2.4.3.** *Let $S$ be an $n$-bit S-box with $y = (y_0, \dots, y_{n-1})$ the ANF of $S$, where each $y_i$ is a polynomial in the input variables $x = (x_0, \dots, x_{n-1})$ of $S$. $S$ is perfect if and only if each $y_i$ contains* all *monomials of degree $n-1$. An example of such an S-box over 4 bits is the following :*

$$S = [1, 4, 3, 5, 13, 7, 12, 10, 8, 0, 11, 15, 6, 14, 9, 2]$$

*Proof.* Let $S$ be an $n$-bit S-box satisfying the characterization above. Since $S$ is invertible, we know that we already have

$$0\dots0 \xrightarrow{S} \mathbb{K} = \{0\dots0\} \text{ and } 1\dots1 \xrightarrow{S} \mathbb{K} = \{1\dots1\}.$$

We first study the case $\boldsymbol{k} \xrightarrow{S} \mathbb{K}$ where $w(\boldsymbol{k}) = n-1$. In that case, we have $\mathbb{U}_{\boldsymbol{k}} = \{\boldsymbol{k}, 1\dots1\}$ and thus $F_{\boldsymbol{k}} = \{x^{\boldsymbol{k}}, x_0\dots x_{n-1}\}$. Then if $\boldsymbol{k'} \in \mathbb{K}$, from Proposition 2.4.1 this means that either $x^{\boldsymbol{k}}$ or $x_0\dots x_{n-1}$ appears in the expression of $y^{\boldsymbol{k'}}$. Especially, for $S$ to be perfect, this needs to hold for every $\boldsymbol{k'} \in \mathbb{E}_n$, thus for every $i \in \{0, \dots, n-1\}$, $x^{\boldsymbol{k}}$ or $x_0\dots x_{n-1}$ must appear in the expression of $y_i$.

However, since $S$ must be invertible, it is well known that each component of its ANF must have a degree at most $n-1$, hence $x_0\dots x_{n-1}$ cannot appear in any $y_i$. To summarize, to have $\boldsymbol{k} \xrightarrow{S} \mathbb{K} = \mathbb{E}_n$ for every $\boldsymbol{k}$ such that $w(\boldsymbol{k}) = n-1$, then for every such $\boldsymbol{k}$, $x^{\boldsymbol{k}}$ must appear in the expression of each and every $y_i, i \in \{0, \dots, n-1\}$, which exactly means that each $y_i$ contains all monomials of degree $n-1$.

Now for every remaining case, i.e. $1 \leq w(\boldsymbol{k}) \leq n-2$, $\mathbb{U}_{\boldsymbol{k}}$ always contains at least one $\bar{\boldsymbol{k}}$ such that $w(\bar{\boldsymbol{k}}) = n-1$, and thus $F_{\boldsymbol{k}}$ contains at least one monomial of degree $n-1$. If we want to have $\boldsymbol{k} \xrightarrow{S} \mathbb{E}_n$, this means that every $y_i$ must contain at least one monomial from $F_{\boldsymbol{k}}$. However, each $y_i$ contains all monomials of degree $n-1$, and $F_{\boldsymbol{k}}$ contains at least one such monomial, and thus $\boldsymbol{k} \xrightarrow{S} \mathbb{E}_n$ holds, which leads to the fact that $S$ is then a perfect S-box. □

This characterization is very similar to the property that an S-box should verify to have a good resistance against division property given in [37]. However, their representation is a bit different, and we show that choosing a perfect S-box for an SPN block cipher is actually the optimal choice when considering classical division property. First, we need the two following lemmas.

**Lemma 2.4.4.** *Let $S$ be an $n$-bit S-box, and $\boldsymbol{k}, \boldsymbol{k'} \in \mathbb{F}_2^n$ such that $\boldsymbol{k} \preceq \boldsymbol{k'}$. Let $\widetilde{\boldsymbol{k}} \in \mathbb{F}_2^n$ such that $\boldsymbol{k'} \xrightarrow{S} \widetilde{\boldsymbol{k}}$. Then we have $\boldsymbol{k} \xrightarrow{S} \widetilde{\boldsymbol{k}}$.*

*Proof.* From $\boldsymbol{k} \preceq \boldsymbol{k'}$, we know that $\mathbb{U}_{\boldsymbol{k'}} \subseteq \mathbb{U}_{\boldsymbol{k}}$, and as such, $F_{\boldsymbol{k'}} \subseteq F_{\boldsymbol{k}}$. Since $\boldsymbol{k'} \xrightarrow{S} \widetilde{\boldsymbol{k}}$, we know that $y^{\widetilde{\boldsymbol{k}}}$ contains a monomial in $F_{\boldsymbol{k'}}$. However, since $F_{\boldsymbol{k'}} \subseteq F_{\boldsymbol{k}}$, we also have that $y^{\widetilde{\boldsymbol{k}}}$ contains a monomial in $F_{\boldsymbol{k}}$, which exactly means $\boldsymbol{k} \xrightarrow{S} \widetilde{\boldsymbol{k}}$. □

**Lemma 2.4.5.** *Let $\mathcal{S}^\star$ be an S-box layer such that $\mathcal{S}^\star = (S^\star, \ldots, S^\star)$ where $S^\star$ is a perfect S-box. Let $\mathcal{S}$ be another S-box layer such that $\mathcal{S} = (S, \ldots, S)$ where $S$ is any non-perfect S-box. Let $\boldsymbol{k}, \widetilde{\boldsymbol{k'}}$ such that $\boldsymbol{k} \xrightarrow{\mathcal{S}} \widetilde{\boldsymbol{k'}}$. Then we can always find $\widetilde{\boldsymbol{k}} \in \mathbb{F}_2^n$ such that $\boldsymbol{k} \xrightarrow{\mathcal{S}^\star} \widetilde{\boldsymbol{k}}$ and $\widetilde{\boldsymbol{k}} \preceq \widetilde{\boldsymbol{k'}}$.*

*Proof.* Denote by $\boldsymbol{k}_i$ the $i$-th block of $\boldsymbol{k}$ which goes through the $i$-th S-box, *i.e.* $\boldsymbol{k}_i \xrightarrow{S} \widetilde{\boldsymbol{k'}}_i$. Note that $\boldsymbol{k} \preceq \boldsymbol{k'}$ is equivalent to $\boldsymbol{k}_i \preceq \boldsymbol{k'}_i$ for all $i$. We can build each block $\widetilde{\boldsymbol{k}}_i$ of $\widetilde{\boldsymbol{k}}$ such that $\boldsymbol{k} \xrightarrow{\mathcal{S}^\star} \widetilde{\boldsymbol{k}}$ and $\widetilde{\boldsymbol{k}} \preceq \widetilde{\boldsymbol{k'}}$ as follow :

- If $\widetilde{\boldsymbol{k'}}_i = 0 \ldots 0$, then $\widetilde{\boldsymbol{k}}_i = 0 \ldots 0$,
- If $\widetilde{\boldsymbol{k'}}_i = 1 \ldots 1$, then $\widetilde{\boldsymbol{k}}_i = 1 \ldots 1$,
- Otherwise, since $S^\star$ is perfect, we can choose $\widetilde{\boldsymbol{k}}_i = \boldsymbol{e}$, where $\boldsymbol{e}$ is a unit vector such that $\boldsymbol{e} \preceq \widetilde{\boldsymbol{k'}}_i$.

By building $\widetilde{\boldsymbol{k}}_i$ as described, it is clear that for all $i$ we have $\boldsymbol{k}_i \xrightarrow{\mathcal{S}^\star} \widetilde{\boldsymbol{k}}_i$ and $\widetilde{\boldsymbol{k}}_i \preceq \widetilde{\boldsymbol{k'}}_i$. As such, $\boldsymbol{k} \xrightarrow{\mathcal{S}^\star} \widetilde{\boldsymbol{k}}$ and $\widetilde{\boldsymbol{k}} \preceq \widetilde{\boldsymbol{k'}}$. □

We are now ready to prove the following theorem, which shows that using a perfect S-box is the optimal choice for classical division property (i.e. without using our extension technique).

**Theorem 2.4.6.** *For a given $n$-bit block-cipher where only the S-box remained to be determined (i.e. the linear layer $\mathcal{L}$ is fixed), using an S-box layer $\mathcal{S}^\star$ built with a perfect S-box $S^\star$ is optimal in terms of resistance against classical division property. We define optimal in the sense that if we denote by $r^\star$ (resp. $r$) the smallest number of round such that $\mathbb{K}_{r^\star} = \mathbb{E}_n$ (resp. $\mathbb{K}_r = \mathbb{E}_n$) when using a perfect S-box $S^\star$ (resp. any other S-box $S$), then we always have $r^\star \leq r$. This basically means that using a perfect S-box always gives the best resistance against classical division property.*

*Proof.* Let the following be a division trail when using a non-perfect S-box :

$$\boldsymbol{k}^0 \xrightarrow{\mathcal{S}} \widehat{\boldsymbol{k}}^0 \xrightarrow{\mathcal{L}} \boldsymbol{k}^1 \xrightarrow{\mathcal{S}} \ldots \xrightarrow{\mathcal{L}} \boldsymbol{k}^r$$

where $\boldsymbol{k}^r \in \mathbb{E}_n$. We can build the following division trail when using a perfect S-box :

$$\boldsymbol{k}^0 \xrightarrow{\mathcal{S}^\star} \widetilde{\boldsymbol{k}}^0 \xrightarrow{\mathcal{L}} \boldsymbol{k}^1 \xrightarrow{\mathcal{S}^\star} \ldots \xrightarrow{\mathcal{L}} \boldsymbol{k}^r.$$

For this division trail to be valid, we use the two previous lemma :

- Using Lemma 2.4.5, since $\boldsymbol{k}^i \xrightarrow{\mathcal{S}} \widehat{\boldsymbol{k}}^i$ we can build $\widetilde{\boldsymbol{k}}^i$ such that $\boldsymbol{k}^i \xrightarrow{\mathcal{S}^\star} \widetilde{\boldsymbol{k}}^i$ and $\widetilde{\boldsymbol{k}}^i \preceq \widehat{\boldsymbol{k}}^i$.
- Since $\widetilde{\boldsymbol{k}}^i \preceq \widehat{\boldsymbol{k}}^i$ and $\widehat{\boldsymbol{k}}^i \xrightarrow{\mathcal{L}} \boldsymbol{k}^{i+1}$, using Lemma 2.4.4 we know that $\widetilde{\boldsymbol{k}}^i \xrightarrow{\mathcal{L}} \boldsymbol{k}^{i+1}$ is a valid transition.

Hence, for any unit vector $\boldsymbol{e}$, there is a valid division trail over $r$ rounds which ends with $\boldsymbol{e}$ when using a perfect S-box, i.e. we have $\mathbb{K}_r = \mathbb{E}_n$. By definition, $r^\star$ is the smallest number of rounds which should verify this condition, thus $r^\star \leq r$. Moreover, by definition, the best distinguisher we can build when using a perfect S-box (resp. any other S-box) is over $r^\star - 1$ rounds (resp. $r - 1$ rounds). Hence why using a perfect S-box gives the best resistance against division property. □

Thus when considering classical division property, choosing the best S-box in regards to security is pretty clear. Note however that such an S-box has a very peculiar behavior with our technique. Indeed, since *every* monomial of degree $n-1$ appears in *every* component of the ANF, this property cannot still hold when we consider either $L \circ S$ or $S \circ L$ where $L$ is a linear mapping (different from a permutation). As such, if a perfect S-box is used, when considering our technique, the first and last round will be somewhat weaker, as the S-box will not be perfect for these rounds. However, every other round will still use this perfect S-box, thus in a way, the behavior "in the middle" of the cipher would still be good.

We decided to search for a better S-box to use for RECTANGLE, in the hope that it would lead to a better resistance against our technique. Since the rational behind S-box design highly depends on potential applications of the resulting block cipher, we restrict the search space to S-boxes linearly equivalent to the original RECTANGLE S-box. Indeed, linearly equivalent S-boxes have similar structures regarding differential and linear properties, see Section 4 of the Introduction.

For 4-bit S-boxes, as there are about $2^{14.3}$ invertible matrices of size 4, the main issue we are facing is the high complexity of trying all the $2^{28.6}$ candidates for $(A, B)$. Indeed, many hours are required to search for a division property distinguisher, making the whole search infeasible. Hence, we propose to use several heuristics to select which pairs $(A, B)$ to try.

**Selecting good S-boxes.** Our first idea was to compute the division property propagations tables of all candidates $(A, B)$. This required to perform $2^{28.6} \times 2^{2\times 4} = 2^{36.6}$ non trivial operations and took approximately 80h on a Xeon E5-2695 (72 cores). Among all those linearly equivalent S-boxes, none of them were perfect. However we found 56 *almost perfect* S-boxes, i.e. with 13 (instead of 14) transitions $k \to \{1000, 0100, 0010, 0001\}$. Note that many pairs lead to the same table but the division property only depends on the table. Hence it is enough to try only one representative per table. Since some implementations of block ciphers do not use a table to store the S-box, we believe it makes sense to select the representative which would add less extra XORs. Hence, for each of the 56 tables we selected the couple $(A, B)$ with the lower XOR count and ran our new automated search tool. As a result, we found that by using

$$A = \begin{pmatrix} 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 \end{pmatrix} \text{ and } B = \begin{pmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

which results in only 5 XOR, and replacing all S-boxes of RECTANGLE by $S' = B \circ S \circ A$ where $S$ is the original S-box of RECTANGLE, then even when using our technique, there is no distinguisher over 9 rounds of this variant of RECTANGLE. We were however able to find a distinguisher over 8 rounds of this variant, using our technique where $L_{in}$ is built with $L^0_{in}$ as defined below, others $L^i_{in}$ for $i \neq 0$ are the identity, and each block $L^i_{out}$ of $L_{out}$ are the same

$$L^0_{in} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix}, L^i_{out} = \begin{pmatrix} 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}.$$

This results in a distinguisher of data complexity $2^{63}$ resulting in 14 balanced bits. Note that the classic search algorithm for division property distinguishers lead to no distinguisher even over 8 rounds, which shows again that our extension technique can find new distinguishers. Moreover, even when using a perfect S-box (such as the one given in Proposition 2.4.3), the best distinguisher using the classic search algorithm is also over 7 rounds, which shows that our choice of S-box, even though it is not a perfect one, is optimal with respect to the classic division property.

We believe that this could lead to a new criteria when designing S-boxes, as for the case of RECTANGLE, it improves the resistance against division property based distinguishers by 2 rounds. We would thus first build the S-box according to classical criteria (differential and linear resistance, ...), then look at the linear equivalent S-boxes and take the one with *the best* division property propagation table. According to our experiment, while we do not have the same optimality proof as for classical division property, using an S-box with a division property propagation table as close as possible to the one of a perfect S-box seems to be the best choice.

**About golden S-boxes.** In 2007, Leander and Poschmann [97] analyzed all 4-bit S-boxes up to linear equivalence and exhibited a set of 16 equivalence classes leading to optimal S-boxes (called *golden S-boxes*) with respect to both differential and linear cryptanalysis. We went through all members of each of these equivalence classes to see if any of them is a perfect S-box. Indeed, recall that division property is not invariant through linear equivalence. As a result, it turns out that there is no S-boxes among all of these that is a perfect S-box, thus we cannot have an S-box which is optimal for both linear, differential and division property based cryptanalysis. However, four of these classes have some almost perfect S-boxes among them, i.e. with 55 transitions $\boldsymbol{k} \to \boldsymbol{k'}$ with $w(\boldsymbol{k'}) = 1$ (instead of the maximum of $4 \times 14 = 56$), namely classes $G_0, G_1, G_2$ and $G_8$. We give one example for each of these classes in Table 2.1, as well as an example with the maximum number of such transitions for each other class. We also give the number of S-boxes reaching that maximum number of transitions across each class. Note that according to Proposition 2.3.1, this number of transition is invariant by permutation equivalence, i.e. for a given S-box $S$ with $n_1$ such transitions, then for any permutations $P, P'$, $S' = P \circ S \circ P'$ also has $n_1$ such transitions. We can thus reduce the number of member of each class to examine by picking one member $S$ and examining all S-boxes built as $L_2 \circ S \circ L_1$ with $L_1 \in \mathbb{L}_{in}$ and $L_2 \in \mathbb{L}_{out}$, where $\mathbb{L}_{in}$ and $\mathbb{L}_{out}$ are the spaces defined in Section 2.3.2. The number of S-boxes given here is thus computed when considering that equivalence relation, but this shows that there are actually a decent amount of choice of S-boxes if one wants to consider other criterion than differential, linear and division property cryptanalysis for choosing an S-box. It is worth noting however that two S-boxes that are permutation equivalent do not necessarily lead to the same result for a given block cipher.

### 2.4.3 Division Property against PRESENT

PRESENT [33] is a 64-bit lightweight block cipher, using either 80 bits or 128 bits keys, with a round function very similar to RECTANGLE and using 4-bit S-boxes. The best known division property based integral distinguisher is from [143] over 9 rounds, requiring $2^{60}$ data and resulting in 1 balanced bit. We applied our previous algorithm to this block cipher, and were actually able to show that our technique cannot lead to a distinguisher over 10 rounds of PRESENT. Indeed, as mentioned at the end of the previous section, if we go through all vectors $\tilde{\boldsymbol{k}}_i^0$ of weight 2, then all of the resulting sets $\mathcal{T}_{out}^S[\mathbb{K}^j_{\tilde{\boldsymbol{k}}_i^0}]$ are empty, meaning that if there is at least one vector of weight 2 or lower in $\widetilde{\mathbb{K}}^0$, then this cannot result in some balanced bits after 10 rounds. Moreover, if we go through all linear mappings $L \in \mathbb{L}_{in}$ and compute all possibles propagations $\boldsymbol{k} \xrightarrow{S'} \mathbb{K}$ where $w(\boldsymbol{k}) = 3$ and $S' = S \circ L$ with $S$ the S-box of PRESENT, then $\mathbb{K}$ will *always* contains at least one vector of weight 2, or at

| Class | S-box | Number of transitions | Number of S-boxes |
|---|---|---|---|
| $G_0$ | $[0, 1, 11, 13, 2, 9, 15, 12, 7, 4, 5, 14, 8, 6, 3, 10]$ | **55** | 1536 |
| $G_1$ | $[0, 9, 3, 11, 13, 15, 10, 4, 1, 12, 6, 5, 14, 7, 2, 8]$ | **55** | 1536 |
| $G_2$ | $[0, 12, 3, 11, 13, 10, 15, 1, 4, 7, 6, 14, 5, 2, 9, 8]$ | **55** | 1536 |
| $G_3$ | $[0, 3, 7, 12, 13, 9, 11, 10, 2, 15, 14, 4, 5, 8, 6, 1]$ | 53 | 3360 |
| $G_4$ | $[0, 3, 7, 5, 11, 15, 2, 12, 13, 6, 14, 9, 10, 1, 8, 4]$ | 53 | 3360 |
| $G_5$ | $[0, 1, 14, 13, 7, 8, 3, 9, 11, 12, 4, 10, 5, 2, 15, 6]$ | 53 | 3360 |
| $G_6$ | $[0, 1, 7, 4, 14, 8, 3, 9, 11, 5, 13, 10, 12, 2, 15, 6]$ | 53 | 3360 |
| $G_7$ | $[0, 3, 1, 10, 14, 12, 11, 15, 7, 9, 8, 5, 6, 4, 2, 13]$ | 53 | 3360 |
| $G_8$ | $[0, 10, 1, 2, 15, 13, 12, 8, 4, 14, 9, 5, 6, 11, 7, 3]$ | **55** | 1536 |
| $G_9$ | $[0, 13, 2, 12, 14, 1, 10, 8, 7, 6, 4, 9, 15, 11, 5, 3]$ | 54 | 1344 |
| $G_{10}$ | $[0, 8, 3, 1, 15, 14, 13, 9, 4, 12, 10, 6, 5, 11, 7, 2]$ | 54 | 1344 |
| $G_{11}$ | $[0, 1, 6, 4, 11, 12, 2, 13, 14, 3, 9, 8, 10, 15, 5, 7]$ | 53 | 3360 |
| $G_{12}$ | $[0, 1, 2, 12, 7, 4, 14, 5, 10, 15, 9, 8, 11, 3, 13, 6]$ | 53 | 3360 |
| $G_{13}$ | $[0, 7, 11, 14, 13, 1, 5, 6, 4, 2, 9, 3, 10, 8, 15, 12]$ | 53 | 3360 |
| $G_{14}$ | $[0, 9, 3, 15, 5, 10, 8, 13, 4, 7, 14, 6, 11, 1, 2, 12]$ | 54 | 1344 |
| $G_{15}$ | $[0, 1, 6, 8, 11, 12, 14, 13, 2, 15, 9, 5, 3, 7, 4, 10]$ | 54 | 1344 |

Table 2.1: Examples of S-boxes with the maximum number of transitions $\boldsymbol{k} \to \boldsymbol{k'}$ with $w(\boldsymbol{k'}) = 1$

least one vector of weight 1. Hence, no matter which linear map we take from $\mathbb{L}_{in}$, after the first S-box layer, there will always be a vector of weight either 1 or 2, which lead to a set $\mathbb{K}^{10}$ containing all unit vectors, and thus no distinguisher over 10 rounds can be built using our technique.

### 2.4.4 Strengthening PRESENT

As for RECTANGLE, we search for another S-box to use which is linear equivalent to the S-box of PRESENT such that it would improve the resistance against division property based distinguishers. By using $S' = B \circ S \circ A$ with

$$A = \begin{pmatrix} 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \text{ and } B = \begin{pmatrix} 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

instead of $S$ for all S-boxes of PRESENT, we do not have any division property based distinguisher over 8 rounds of this variant of PRESENT even when using our extension technique. However, we found a distinguisher over 7 rounds with data complexity $2^{63}$ and with all 64 bits being balanced using our technique, with $L_{out}$ being the identity and $L_{in}$ being built with

$$L_{in}^0 = \begin{pmatrix} 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

and $L_{in}^i$ as the identity for $i \neq 0$. The classical search algorithm was only able to find a distinguisher on up to 6 rounds, and again, this choice is optimal for classic division property, as a perfect S-box also gives a classic division property based distinguisher over 6 rounds. This again highlights that our extension technique allows to find better distinguishers than the classical search. Note that, for

non table-based implementation, the new S-box we propose only requires two extra XORs compared to the original S-box of PRESENT.

## 2.5 Conclusion

We studied further the division property and the distinguishers that are built from it. We show that while the previous search methods were able to efficiently find some integral distinguishers based on the division property, the search space explored by these methods does not actually cover all possibilities. As such, we show that for $r$ rounds of a block cipher $E$, considering $E' = L_{out} \circ E \circ L_{in}$ instead of $E$, where $L_{out}$ and $L_{in}$ are block diagonal linear maps, can lead to some integral distinguisher over $E'$, while $E$ does not have any. We provide an algorithm to find such distinguisher, and successfully apply it to the block cipher RECTANGLE, on which we found an integral distinguisher over 10 rounds, requiring $2^{63}$ data and leading to 1 balanced bit. This is one more round than the previously known distinguishers. The design of our algorithm also allows us to prove that our technique cannot extend the best distinguisher on PRESENT over one more round. Finally, we give a criteria on S-boxes which allows to prove that if an S-box verifies this criteria, it will provide the best resistance against division property. To our knowledge, this is the first time that such an optimality result is given and formally proven for division property. According to our observations, we were able to exhibit some variants of RECTANGLE and PRESENT which have a better resistance against integral distinguisher based on the division property. Namely the maximum number of round on which we could find an integral distinguisher over our variant of RECTANGLE and PRESENT is 2 rounds lower than when using the original S-box. This might give a new design criteria for S-boxes and further research about this will be needed.

We believe that overall, this technique could open up a lot of questions and possibilities. Indeed, we basically decomposed a block cipher $E$ as

$$E = (E_2 \circ L_{out}^{-1}) \circ (L_{out} \circ E_1 \circ L_{in}) \circ (L_{in}^{-1} \circ E_0),$$

and merged $L_{in}$ and $L_{out}$ with the first S-box layer. But could we use the same technique at a lower level, i.e. decomposing the round function as $f = \mathcal{L} \circ L^{-1} \circ L \circ \mathcal{S}$, merging $L$ with $\mathcal{S}$ for example ? In a more general view, the question is : what is the best representation of a block cipher to propagate the division property ? Also, our algorithm focuses on finding any distinguisher over an SPN block cipher. Thus, how could we find an optimal distinguisher (in term of data) using this technique, as applying our algorithm when more than one S-box has an input division property which differs from $1 \dots 1$ seems quite hard in term of complexity. The same issue comes up when considering 8-bit S-boxes, as we need more calls to the solver, and the resulting MILP models are way more complicated, and thus takes a longer time to be solved. Finally, could this also apply to other constructions such as Feistel block ciphers or permutation based block ciphers ? Indeed, our algorithm is efficient because we can basically only study the propagation from after the first S-box layer to before the last S-box layer.

*"Failure is only the opportunity to begin again. Only this time, more wisely."*

— Uncle Iroh

# Chapter 3

# Variants of the AES Key Schedule for Better Truncated Differential Bounds

Differential attacks are one of the main ways to attack block ciphers. Hence, we need to evaluate the security of a given block cipher against these attacks. One way to do so is to determine the minimal number of active S-boxes, and use this number along with the maximal differential probability of the S-box to determine the maximal probability of any differential characteristic. Thus, if one wants to build a new block cipher, one should try to maximize the minimal number of active S-boxes. On the other hand, the related-key security model is now quite important, hence, we also need to study the security of block ciphers in this model. At FSE'17, Khoo et al. [84] give a byte-permutation to use instead of the original AES key-schedule which improves the security in this model when considering differential attacks.

In this chapter, we go further and study how we can design a *good* permutation to use as the key schedule in AES-128. More precisely, we first start by giving some bounds on the reachable minimal number of active S-boxes for up to 7 rounds of AES if we use a simple permutation as its key schedule. Especially, we show that there is no permutation that can reach a minimal number of active S-boxes of 18 or more over 5 rounds. These bounds allow us to know the results that a "perfect" permutation could reach. Then, we provide a method to search for such a permutation. To do so, we reused the meta-heuristic approach given by Nikolić in [111], combined with a Constraint Programming model inspired from the work of Gerault et al. in [67]. Especially, we give a way to model the underlying equations of a truncated differential characteristic, leading to a more precise model than the original one from [67]. Namely, the truncated differential characteristics found are always valid until we consider the DDT of the S-box.

We also went further and modified both the key schedule and one step of the AES round function (namely, `ShiftRows`) to see whether we can achieve better bounds. As a result, we exhibit a permutation $P_k$ which, when used as the AES key schedule, lead to a minimal number of active S-boxes of 20 over 6 rounds, while no characteristic has a probability larger than $2^{-128}$. When changing both the key schedule and the `ShiftRows` step, we give several pairs of permutations $(P_k^i, P_s^i)$ that have a minimal number of active S-boxes of 21 over 6 rounds, while again, no characteristic has a probability larger than $2^{-128}$. While we applied this method to AES, it is quite generic and could also be used on any block cipher, as long as one have an efficient enough way to compute the minimal number of active S-boxes. Our implementation is available at `https://github.com/TweakAESKS/TweakAESKS`.

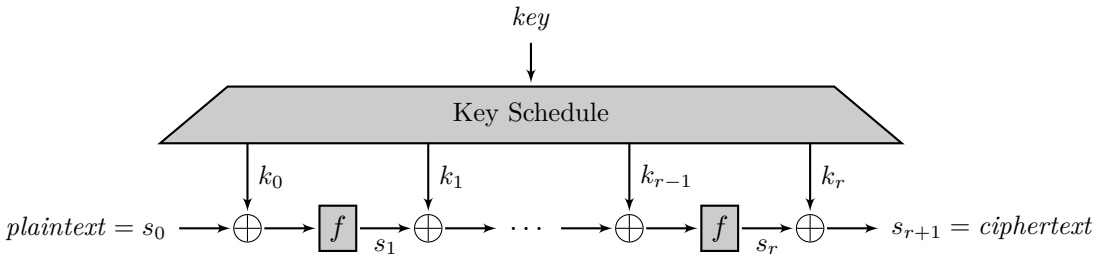


Figure 3.1: Generic iterated cipher construction [78]

## 3.1 Introduction

First introduced in 1990 by Biham and Shamir [17], differential cryptanalysis is one of the main tool to analyze and attack symmetric primitives. The main idea is to introduce some differences in the plaintext, and see how these differences propagate through the different steps of the algorithm, independently from the key. For example, given an encryption function $\mathcal{E}(p, k)$ encrypting the plaintext $p \in \mathbb{F}_2^{n_b}$ using a key $k \in \mathbb{F}_2^{n_k}$, if one is able to prove that there exists a pair of differences $\Delta_{in}, \Delta_{out} \in \mathbb{F}_2^{n_b}$ such that $\mathcal{E}(p \oplus \Delta_{in}, k) = \mathcal{E}(p, k) \oplus \Delta_{out}$ for all keys, then it gives a strong distinguisher for the encryption function $\mathcal{E}$. Moreover, due to the non-linearity of $\mathcal{E}$, such a differential relation could only hold with a certain probability. Consequently, a lot of work has been put into designing algorithms that search for the best possible differential characteristics of a given cipher. For instance, Matsui's algorithms [103] were the first designed. Most of modern ciphers are now built as *iterated ciphers*, i.e., a round function $f$ is built and repeated several times, XOR-ing a round key between each application of $f$, see Figure 3.1. Thus, to search for such a pair $(\Delta_{in}, \Delta_{out})$, one often studies the propagation of the input difference through each round of the cipher, thus leading to a *differential characteristic* consisting of all differences in each state $s_i$.

One can also choose to consider only *truncated differences*, that is, only look at whether or not the difference in one byte is zero. While this can also directly lead to various attacks, e.g., impossible differential attacks [15, 87], it can also be used to get some results in differential cryptanalysis. Indeed, in most cipher designs, the non-linear component consists of an S-box, a small non-linear function applied several times over all iterations. This S-box is the reason that some differential characteristic only holds with a certain probability. Given an S-box $S$ acting on a small number of $s$ bits, and for each pair $(\Delta_{in}, \Delta_{out}) \in \mathbb{F}_2^{2s}$, one can easily compute how many $x \in \mathbb{F}_2^s$ verifies the relation $S(x \oplus \Delta_{in}) = S(x) \oplus \Delta_{out}$. This allow to compute the Difference Distribution Table (DDT) of the S-box, which gives the probability that the above relation holds for each $(\Delta_{in}, \Delta_{out})$. Thus, given a differential characteristic, one can easily compute the probability that it holds, simply by multiplying all differential probabilities of each S-box together.[1] Hence, given a *truncated* differential characteristic, while we cannot determine the exact probability that this characteristic holds, we can deduce its minimal probability. Indeed, if the S-box has a maximal differential probability of $p$, and there are $n$ S-boxes with a non-zero difference (called *active S-boxes*), then the truncated differential characteristic holds with a probability at most $p^n$. Thus, given the maximal differential probability of the S-box used and the bit-length $n_k$ of the key, one can easily deduce the minimal number of active S-boxes $n_{min}$ that leads to $p^{n_{min}} < 2^{-n_k}$. So, if for a given number of round, we can prove that there is at least $n_{min}$ active S-boxes, we know that there would be no differential characteristic

[1] Using the fair assumption that each round is independent, which while obviously not true, is admitted as a reasonable assumption.

with a probability better than $2^{-n_k}$, which would mean that finding a pair of plaintexts satisfying this characteristic would *a priori* costs more than an exhaustive search for the key.

Such differentials and truncated differentials can also be considered in the *related-key model*. First introduced in 2009 to attack AES-192 and AES-256 [24, 26], this model allows the attacker to inject differences in the plaintext, but also in the key. Another worth-mentionning model is the more recent *related-tweak model* for tweakable block ciphers, where the attacker fully controls an additional input for the block cipher called a *tweak* [99, 148]. While this model is closer to chosen-plaintext attacks, the tweak is often (but not necessarily) used alongside the key and thus involved in the key schedule, such as in the TWEAKEY framework [80]. Since the attacker can now inject some differences in both the plaintext and the key, this causes a large increase in the complexity to search differential and truncated differential characteristics. Nonetheless, several tools have been designed to tackle this problem [28, 65, 67]. Hence, a few proposals were made to give another, more secure, key schedule for some primitives, such as [110, 43] for AES and [111] for SKINNY and AES-based constructions from FSE 2016 [79]. However, their main concern was mostly to design a more secure key schedule, without considering the possible loss in efficiency. To that regard, Khoo et al. [84] proposed a new key schedule for AES which consists in only a permutation at the byte level, based on their proof on the number of active S-boxes in the related-key model for AES. Using a permutation thus leads to a very efficient key schedule, both in software and hardware, and can also make the analysis easier.

## 3.2 Background

Differential cryptanalysis was first introduced by Biham and Shamir in 1990 [17] and mainly consists in studying the propagation of differences between two plaintexts through the cipher. Here, we only consider *truncated differences*, that is, we are only interested in whether a byte does have a non-zero difference (*active* byte) or not (*inactive* byte). Our work is centered around AES, for which we make a few remainders. AES is the NIST block cipher standard, derived from Rinjdael [46]. It uses an internal state of 128 bits, and several key sizes are available, namely 128, 192 and 256. Here, when mentioning AES, we refer to the 128-bit version.

It is an SPN block cipher, iterating a round function $R = \mathsf{MC} \circ \mathsf{SR} \circ \mathsf{SB} \circ \mathsf{ARK}$ 10 times, where each component of the round function is quickly described in the following. The state can be viewed as a $4 \times 4$ byte array, and thus we will often talk about *columns* of the state. The round function consists in four operations: `AddRoundKey` (ARK), `SubBytes` (SB), `ShiftRows` (SR) and `MixColumns` (MC). ARK XORs the round key into the internal state. This round key is derived from the master key using a key schedule KS, for which we do not give details, our ultimate goal being to change it. We refer the interested reader to [46] for the original descriptions. SB applies a non-linear operation (called S-box) on each byte of the state, then SR performs a cyclic shift of each row, where Row $j$ is shifted by $j-1$ bytes to the left, $j \in \{1, 2, 3, 4\}$. Finally, MC is a linear operation that multiplies each column of the internal state by an MDS matrix with coefficients in $\mathbb{F}_{2^8}$.

We first recall several well known properties of the MC operation, which will be used in the rest of the chapter. Here, $w(x)$ corresponds to the number of active bytes in $x$, which is either a state or a column of the state.

**Proposition 3.2.1** (`MixColumns` MDS property)**.** *Let $z$ and $y$ be two state columns such that $\mathsf{MC}(z) = y$. Then, either $w(z) + w(y) = 0$ or $w(z) + w(y) \geq 5$. Moreover, for any five bytes in $y$ and $z$, there exists one linear equation between those five bytes.*

*Proof.* This comes directly from the fact that the matrix used in the MC operation is MDS. □

**Proposition 3.2.2** (`MixColumns` linear property)**.** *Let $z, z', y, y'$ be four state columns such that $\mathsf{MC}(z) = y$ and $\mathsf{MC}(z') = y'$. Then, the `MixColumns` MDS property also holds for $(z \oplus z')$ and $(y \oplus y')$, that is: either $w(z \oplus z') + w(y \oplus y') = 0$ or $w(z \oplus z') + w(y \oplus y') >= 5$*

*Proof.* This comes directly from the previous proposition and the fact that $\mathsf{MC}$ is linear. □

**Lemma 3.2.3.** *Let $k, x, y, z$ be four state columns such that $\mathsf{MC}(z) = y$, $z$ contains at least one active byte and $x = y \oplus k$. Denote by $i_{y,z}$ the number of inactive bytes in $y$ and $z$ (i.e., $i_{y,z} = 8 - w(y) - w(z)$) and $c_{y,k,x}$ the number of bytes from $y$ that are cancelled by $k$ in $x$. If $i_{y,z} + c_{y,k,x} \geq 5$, then there is at least one linear equation on some bytes of $k$. Moreover, this can only happens if $c_{y,k,x} \geq 2$.*

*Proof.* If $i_{y,z} + c_{y,k,x} \geq 5$, then from the `MixColumns` MDS property, it follows that there is an equation between any five bytes chosen from the inactive ones in $y$ and $z$, and the bytes from $z$ which are cancelled by $k$. If we denote such a cancelled byte by $z_i$, that is, $z_i \oplus k_i = 0$, then we have $k_i = z_i$, hence the equation involves some bytes of $k$ and some inactive bytes from $y$ and $z$, which are zeros.

Since $z$ contains at least one active byte, we have $w(z) + w(y) \geq 5$, hence $i_{y,z} \leq 3$. Therefore, if $c_{y,k,x} = 1$ (i.e., only one byte is cancelled), we have $i_{y,z} + c_{y,k,x} \leq 4$, and thus no equation is implied. □

When considering truncated differentials, we are often interested in the number of active S-boxes, that is, the number of active bytes going through an S-box (i.e., active bytes at the beginning of the round). We will often refer to the (minimal) number of active S-boxes in a characteristic as the *length* of the characteristic, and to a *minimal characteristic* to refer to a characteristic which reaches the minimal number of active S-boxes. Given a truncated differential characteristic of length $n$, one can deduce the maximal probability that this characteristic can have once being instantiated. Indeed, if the S-box has a maximal non-zero differential probability of $p$, then the maximal probability of this characteristic is $p^n$. If one studies a block cipher with a key of length $n_k$ bits, then the goal is to prove that no characteristics can be instantiated with a probability larger than $2^{-n_k}$. Hence, for $\mathsf{AES}$, since the maximal differential probability of the S-box is $2^{-6}$, we know that if for a given number of rounds the minimal number of active S-boxes is greater or equal than 22, then no differential characteristic with a differential probability larger than $2^{-128}$ exists.

Searching whether a characteristic reaching a given length or maximal probability exists has been a major focus in academic research. One way to find the best probability is to proceed in two steps. First, one try to find a truncated differential characteristic with a minimal number of active S-boxes, and then try to instantiate this characteristic. When searching for such a truncated differential characteristic, one can choose to consider additional information about the cipher along with "basic" propagation rules coming from the round function, to avoid trying to instantiate characteristics that would not be instantiable anyway. Hence for $\mathsf{AES}$, we give the following definitions.

**Definition 3.2.4.** *A characteristic is said to be valid in the "truncated differential setting" if and only if the `MixColumns` linear property is always verified and there is at least one non-trivial solution to the system of equations (if any) induced by Lemma 3.2.3.*

*A characteristic that remains valid even when one does not consider the `MixColumns` linear property nor the equations is said to be valid in the* pure truncated differential setting*.*

The point of these definitions is twofold. On the one hand, since the pure truncated differential setting contains significantly less constraints, the minimal characteristic could be a lot easier to find. However, it may result in an invalid characteristic when one tries to instantiate it, which could have

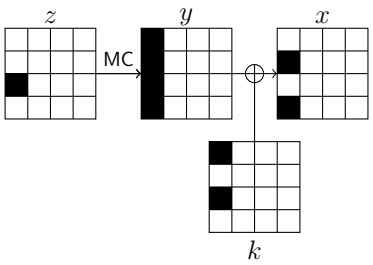


Figure 3.2: A partial round that implies one equation

been detected in the truncated differential setting. Conversely, finding the minimal characteristic in the truncated differential setting could be harder, but the only thing that could invalidate this characteristic is the S-box DDT.

We chose to use the same approach as Gerault et al. [67], who proposed to use two Constraint Programming models. The first one was used to find the minimal characteristics for AES, considering only the `MixColumns` linear property. The second one takes a list of truncated characteristic and tries to find the best instantiation (if any) of each characteristic with respect to its probability. As we aim at changing the key schedule, we changed these models, detailed in the following.

**Model 1.** *This model takes as input a permutation $P_k$ to use as the key schedule and a number of rounds, and output the minimal number of active S-boxes with these parameters in the truncated differential setting. Compared to the first model of [67], we directly model the equations coming from the `MixColumns` operation (see Lemma 3.2.3), resulting in a more reliable result, albeit being slower. We refer the reader to Section 3.2.1 for the method used to model these equations.*

**Model 2.** *This model also takes as input a permutation $P_k$ for the key schedule and a number of rounds, along with a list of truncated differential characteristics. It then goes through each of these truncated characteristics, and tries to find an instantiation with a probability larger than $2^{-128}$. If such an instantiation is found, it gives its probability and the differential characteristic, otherwise it just stops without trying to find an instantiation with a probability smaller than $2^{-128}$.*

### 3.2.1 Modelizing the MC equations in Constraint Programming

We will give here an example of how we generate constraints to modelize the equations coming from the MC operation. From the MDS property of MC, we know that there is an equation between any set of five bytes taken from the same column of $z$ and $y$. Specifically, we have the following equation, where coefficient are in $\mathbb{F}_{256}$:

$$5.z[0] + 7.z[1] + z[3] = 2.y[0] + y[2].$$

Now we take the situation given in Fig. 3.2. First, all bytes 0,1 and 3 of $z$ are inactive, hence we can replace $z[0], z[1]$ and $z[3]$ in the previous equation by zeros. Moreover, we can see that both $y[0]$ and $y[2]$ are cancelled by some bytes in $k$, i.e. $y[i] \oplus k[i] = 0, i \in \{0, 2\}$. Hence, our equation becomes $2.k[0] + k[2] = 0$.

So, if this situation occurs, we know that we have a specific equation involving bytes of $k$. However, this equation has coefficient in $\mathbb{F}_{256}$, which are not handled by Constraint Programming solvers. Hence, we modelize this equation at a bit-level, using the fact that the scalar multiplication in $\mathbb{F}_{256}$ corresponds to a linear operation in $\mathbb{F}_2^8$. By denoting $k^i_j, j \in [0, 7], i \in 0, 2$ the $j$-th bit of

$k[i]$, we have

$$\begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{pmatrix} \cdot \begin{pmatrix} k_0^0 \\ k_1^0 \\ k_2^0 \\ k_3^0 \\ k_4^0 \\ k_5^0 \\ k_6^0 \\ k_7^0 \end{pmatrix} + \begin{pmatrix} k_0^2 \\ k_1^2 \\ k_2^2 \\ k_3^2 \\ k_4^2 \\ k_5^2 \\ k_6^2 \\ k_7^2 \end{pmatrix} = 0.$$

We now have everything to modelize this case using an *if-constraint*. In our model, we have a binary variable for each byte of the state which is set to 0 if the corresponding byte is inactive, and 1 otherwise. Since all the equations only involves some key-bits, we also have binary variables for each bit of each subkey. Restricting this in the situation given in Fig. 3.2, we would have binary variables $z[i], y[i], x[i], k[i], i \in [0, 15]$ modelizing whether or not bytes are active, and binary variables $k_j^i, i \in [0, 15], j \in [0, 7]$ for each bit of the key. Obviously, we need to modelize the fact that if a key byte is inactive, then its bits are all zeros, which is easily modelized with

$$k[i] = 0 \iff k_j^i = 0 \; \forall j \in [0, 7].$$

Hence, the above equation only holds when $z[0] = z[1] = z[2] = 0, y[0] = y[2] = 1$ and $x[0] = x[2] = 0$. Note that we do not need to check that $k[0] = 1$ since the fact that $y[0] = 1$ and $x[0] = 0$ necessarily implies that $k[0] = 1$ (and the same argument goes for $k[2]$). So, to modelize this case, we use an *if-constraint*. Such a constraint is of the form $E \Rightarrow C$, and means that if the expression $E$ is true, then the constraint $C$ must hold. Thus, we modelize the above situation with the constraint

$$\begin{aligned} z[0] = 0 \wedge z[1] = 0 \wedge z[2] = 0 \wedge y[0] = 1 \\ \wedge\, y[1] = 1 \wedge x[0] = 0 \wedge x[2] = 0 \end{aligned} \Rightarrow \begin{aligned} k_7^0 + k_0^2 &= 0 \bmod 2 \wedge \\ k_0^0 + k_7^0 + k_1^2 &= 0 \bmod 2 \wedge \\ k_1^0 + k_2^2 &= 0 \bmod 2 \wedge \\ k_2^0 + k_7^0 + k_3^2 &= 0 \bmod 2 \wedge \\ k_3^0 + k_7^0 + k_4^2 &= 0 \bmod 2 \wedge \\ k_4^0 + k_5^2 &= 0 \bmod 2 \wedge \\ k_5^0 + k_6^2 &= 0 \bmod 2 \wedge \\ k_6^0 + k_7^2 &= 0 \bmod 2 \end{aligned}$$

Hence in our model, we need to do this for all rounds and for each column of the state. The number of constraints coming from this is easy to compute. For a fixed round and column, denote $i$ the number of inactive bytes taken in $z$, hence $\binom{4}{i}$ possibilities, with $1 \leq i \leq 3$. Denote $j$ the number of inactive bytes taken in $y$ hence $\binom{4}{j}$ possibilities. Hence, we have $5 - i - j$ active bytes (that are cancelled) in $y$, taken in the remaining $4 - j$ bytes, thus $\binom{4-j}{5-i-j}$ possibilites. Moreover, we know from Lemma 3.2.3 that we must have $5 - i - j \geq 2$. Thus, we have in total $656r$ constraints for $r$ rounds, as the number of constraints for a fixed round and a fixed column is

$$\sum_{i=1}^{3} \sum_{j=0}^{3-i} \binom{4}{i} \binom{4}{j} \binom{4-j}{5-i-j} = 164.$$

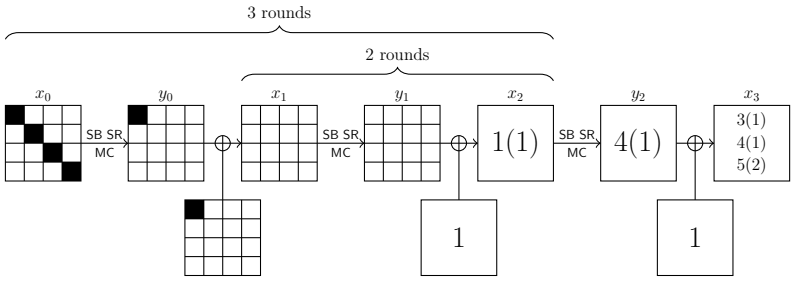


Figure 3.3: Characteristic always valid for 2,3 and 4 rounds. $x(y)$ means that there are $x$ active Sboxes somewhere in the state, with $y$ columns containing at least one active bytes. Multiple $x(y)$ in a state means that one of them must be true

## 3.3 Generic Bounds

Before trying to find a permutation that reaches a certain number of active S-boxes, we need to study which number of S-boxes we can reach. From the fact that using a permutation as the key schedule implies that the number of active bytes in the key is constant, we can deduce several bounds on the number of active S-boxes. To demonstrate these bounds, we show that there is always a differential characteristic of a certain length, independently from the permutation used in the key schedule.

**Proposition 3.3.1.** *Using a permutation as the key schedule, there is always a differential characteristic of length 1 (resp. 5). for 2 (resp. 3) rounds. For 4 rounds, there is always a characteristic of length either 8, 9 or 10. Moreover, these differential characteristics always remain valid in the truncated differential setting.*

*Proof.* Such a characteristic is depicted in Figure 3.3. For 2 rounds, there is only one active byte in the second state, which is cancelled by the active byte in the key. For 3 rounds, the previous characteristic is extended by adding one more round before it, and comes directly from the `MixColumns` MDS property.

For 4 rounds, we add one more round after the 3-round differential characteristic. Since $y_2$ has four active bytes on the same column, and since the key has one active byte anywhere in the key state, $x_3$ can have either 3, 4 or 5 active bytes, which results in a differential characteristic of length either 8, 9 or 10.

No equation is implied since there is always at most one active key byte that is cancelled with the $\mathsf{ARK}$ operation for each round (Lemma 3.2.3). Finally, there are only two `MixColumn` transitions with active bytes, one of the form $\mathsf{MC}(z) = y$ where $z$ and $y$ are one column of the state with $w(z) = 4$ and $w(y) = 1$ and another of the form $\mathsf{MC}(z') = y'$, where $w(z') = 1$ and $w(y') = 4$. Hence, $w(z \oplus z') \geq 3$ and $w(y \oplus y') \geq 3$, and thus the `MixColumns` linear property is always valid. □

**Corollary 3.3.2.** *Using a permutation as the key schedule, the optimal bounds on the number of active S-boxes that can be proven for 2, 3 and 4 rounds is respectively 1, 5 and 10 in the truncated differential setting.*

The proof of this corollary comes directly from the previous proposition. If we try to extend the previous characteristic with one more round, we obtain that there is always a characteristic of length either 19, 20, 21, 24 or 25 in the truncated differential setting. However, if we only consider the pure truncated differential setting, then we have the following proposition.

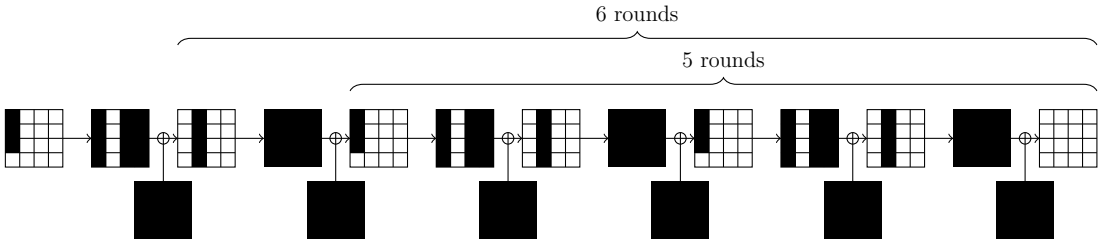


Figure 3.4: Characteristic always valid for 5, 6 and 7 rounds.

**Proposition 3.3.3.** *For 5, 6 and 7 rounds, there is always a characteristic of length respectively 14, 18 and 21 in the pure truncated differential setting.*

*Proof.* Such a characteristic is depicted in Figure 3.4. Note that considering how this kind of characteristic is built, there are a lot of underlying equations in the truncated differential setting, which is very likely to make this characteristic invalid. However, in the pure differential setting, these characteristics always remains valid as they come directly from the propagation rules of the AES round function. □

**Corollary 3.3.4.** *Using a permutation as the key schedule, the optimal bounds on the number of active S-boxes that can be proven for 5, 6 and 7 rounds is respectively 14, 18 and 21 in the pure truncated differential setting.*

Now the first question that we may ask is whether or not there exists a permutation which reaches all those bounds. Fortunately, such a permutation was already found by Khoo et al. in [84], which is

$$P_{KLPS} = (5, 2, 3, 8, 9, 6, 7, 12, 13, 10, 11, 0, 1, 14, 15, 4)$$

However, if we study this permutation in the truncated differential setting for 7 rounds using Model 1, then we have that the minimum number of active S-boxes becomes 22, proving that no differential characteristic with a probability larger than $2^{-128}$ can be found, hence the following theorem.

**Theorem 3.3.5.** *We can find a permutation for the key schedule which guarantees that no differential characteristic with a probability larger than $2^{-128}$ exists for 7 or more rounds of AES. Moreover, this does not depend on the S-box DDT.*

Obviously, now the main question is: How far can we go? Can we find a permutation that reach 22 S-boxes for 6 rounds or lower, or at least a permutation such that no differential characteristic with probability larger than $2^{-128}$ exists? This would allow us to show that even with an extremely simple and efficient key schedule, we can still have a rather good security against differential attacks in the related-key model. We study this in the next section.

## 3.4 Searching for a Permutation

### 3.4.1 Bound on 5 Rounds

In this section, we show that there is no permutation that can reach a minimal number of active S-boxes of 18 over 5 rounds. While this does not imply that we cannot find a permutation such

that there is no differential characteristic with a probability better than $2^{-128}$, this still gives us a good idea of what we can reach for 5 rounds.

To achieve this, we proceed in two steps. First, we search for a set of cycles such that using a given cycle of this set, one cannot build a truncated differential characteristic of length strictly lower than 18, which induces equations (according to Lemma 3.2.3) on at most 1 round. Since all permutations can be decomposed into a composition of cycles, this would not only speed up the search (since we do not need to check every permutation one at a time), but also gives a way to build all permutations that *could* reach 18 S-boxes on 5 rounds. To build such a set of cycles, we used a quite straightforward algorithm.

First, we suppose that the cycle starts with 0. Then, we guess the image of 0, and for each of those guesses, we have two cases: either the cycle is not complete, and thus we need to make another guess on the next element of the cycle, or the cycle is closed. Whenever we make a new guess or decide that the cycle is closed, we can build several truncated key characteristics $k_0 \rightarrow k_1 \rightarrow ... \rightarrow k_4$ according to the current (partial) cycle examined: each active byte in this truncated key characteristic must be a byte that belongs to the current (partial) cycle. Then, for each of those truncated key characteristics, we search the minimal number of active S-boxes that we can reach using this characteristic. To speed up the search, we only consider truncated characteristics that induce equations on at most 1 rounds, such that these characteristics are always valid in the truncated differential model. If, for a given (partial) cycle, one can find a corresponding truncated characteristic with strictly less than 18 S-boxes, then we know that this (partial) cycle cannot be part of the permutation we are looking for. If we were in the case where the cycle was not complete, then we know that we do not need any more guesses, and if the cycle was closed, we can dismiss it. Thus in the end, we will have a set of closed cycles which start with 0, and for which all truncated characteristics that induces equations on at most 1 rounds have at least 18 active S-boxes. We then need to apply the same algorithm, but this time with cycles beginning by 1 and not containing 0 (to avoid repetitions) and so on.

In the end, we have a set of permutations for which we know that, if a permutation reaches a minimal number of active S-boxes of 18 (or higher), then it must be built from this set of cycles. Thus, we just need to built all possible permutations from these cycles, and plug them into Model 1 to see if the actual minimal number of S-boxes is indeed 18 or higher. The number of cycles which can be used to build a permutation reaching 18 S-boxes is given in Table 3.1, and by testing all possible combinations, we found out that there is no such permutation, hence the following theorem.

**Theorem 3.4.1.** *There is no permutation that, when used as key schedule, can reach a minimal number of active S-boxes of 18 or higher over 5 rounds. Using the same method, we were also able to find at least one permutation which have a minimal number of active S-boxes of 16 over 5 rounds, namely:*

$$(15, 0, 2, 3, 4, 11, 5, 7, 6, 12, 8, 10, 9, 1, 13, 14).$$

However, the possibility of reaching 17 S-boxes over 5 rounds is still unknown, and the complexity of the algorithm for 6 rounds is too high. Hence, we focused our search for a permutation reaching 22 active S-boxes over 6 rounds, using another approach we detail in the next section.

### 3.4.2 Finding a Permutation over 6 Rounds

First of all, let us take a quick look at how we could naively search for such a permutation. This is rather straightforward: for each possible permutation, we check whether the minimal number of S-boxes is at least 22. Since we are looking for a permutation over 16 bytes, we have $16! > 2^{44}$

| Length of the cycle | Number of cycles |
|---|---|
| 1 | 16 |
| 2 | 120 |
| 3 | 796 |
| 4 | 6576 |
| 5 | 25656 |
| 6 | 78448 |
| 7 | 112608 |
| 8 | 74904 |
| 9 | 15576 |
| 10 | 1344 |
| 11 | 48 |

Table 3.1: Number of cycles which must be use to build a permutation reaching 18 A-boxes over 5 rounds

possible permutations. While $2^{44}$ basic operations could be achievable in a reasonable amount of time, the computation of the minimal number of S-boxes is actually quite costly. For example, if one would use the algorithm from [65] which has an approximate complexity of $2^{34}$ operations, this would raise the total cost to $2^{78}$ operations, which is clearly impractical. While we do not have a complexity estimation for our constraint programming tool, the average time to solve Model 1 is about 40 minutes for 6 rounds, which would lead to way too much time to try each permutation, so exhausting all permutations is clearly not a viable way to proceed.

On the other hand, one could try to pick a random permutation, evaluate its minimal number of S-boxes, and try again if this number is lower than 22. While the cost of computing the minimal number of S-boxes remains, this approach could be successful if the density of the set of permutation reaching 22 S-boxes overall permutations is high enough. Indeed, if we do this for 7 rounds, we are able to find a permutation reaching the same number of S-boxes for 7 rounds and lower as the permutation from [84] in about 200 tries. However, this approach was not able to find a permutation reaching 22 S-boxes over 6 rounds.

Hence, we need something more efficient for 6 rounds. Inspired by the work of Nikolić [111], we choose to use a meta-heuristic called *simulated annealing*. Meta-heuristics are a class of search algorithms which aim to find an (almost) optimal solution to an optimization problem, often inspired by some real-life phenomenon. To be more precise, unlike *Constraint Programming* or *Integer Linear Programming* which aims at recovering an *optimal* solution, meta-heuristics only look for a *good enough* solution: it may not be optimal, but it should be rather close to an optimal solution. In our case, we could define our optimization problem as: Which permutation maximize the minimal number of active S-boxes over 6 rounds? However, we are not really interested in maximizing the minimal number of S-boxes, we only need to find a permutation which reaches 22 S-boxes. Moreover, our problem is of the form *"Maximize the minimum value of a given function"*, which is not something easily handled by classical techniques like Constraint or Linear Programming. Finally, meta-heuristics are designed to be both relatively easy to implement and rather efficient, hence they seem quite appropriate to tackle this problem.

We give a generic algorithm for simulated annealing in Algorithm 3, also given in [111]. The main idea of this algorithm is to try to maximize a function $f(x)$ (called *objective function*) by progressively improving a solution, starting from a random one, while allowing degradation. To be more precise, starting from a random $x_0$, the algorithm builds another solution $x_i$ from $x_{i-1}$

using the function $\epsilon$. Then, if $f(x_i) > f(x_{i-1})$, then $x_i$ is accepted and the algorithm continues. However, if $f(x_i) \leq f(x_{i-1})$, which would mean that $x_i$ is worse than the previous solution $x_{i-1}$, $x_i$ is only accepted with some probability depending on a value $T$, and if it is rejected, another $x_i$ is generated from $x_{i-1}$. Then, the value $T$ is updated with a function $\alpha(T)$. For more details about this algorithm and the choice of its parameters, we refer the reader to [111, 40, 86].

**Algorithm 3** Simulated Annealing [111]

**Input:** initial temperature $T_0$, cooling schedule $\alpha(T)$, neighbor function $\epsilon(x)$
$x \leftarrow random, \quad T \leftarrow T_0$
**while** termination criteria not met **do**
  $x' \leftarrow \epsilon(x)$
  **if** $f(x') > f(x)$ **then**
    $x \leftarrow x'$
  **else**
    $r \leftarrow U[0,1]$ *Generate a uniformly random real number in [0,1]*
    **if** $r < e^{\frac{f(x')-f(x)}{T}}$ **then**
      $x \leftarrow x'$
    **end if**
  **end if**
  $T \leftarrow \alpha(T)$
**end while**
**Output:** $x$

Now, we need to see how we implement this algorithm in practice. As in [111], we did not observe major differences between different parameters for the initial temperature $T_0$ and the cooling schedule $\alpha(T)$. Hence, we only give one set of parameters, from which all our following results come from. For the initial temperature, we used $T_0 = 2$. For the cooling schedule, we used the same one as in [111], i.e., $\alpha(T) = \frac{T}{1+\beta T}$ with $\beta = 0.001$ Finally, the neighbor function $\epsilon$ generates a new permutation from the one that has been tested. This new permutation should be "close" to the previous one, hence we use a random transposition to generate a new permutation, namely, $\epsilon(x) = \tau \circ x$ where $\tau$ is a random transposition.

The only thing missing to implement the algorithm is a way to evaluate $f(x)$. Recall that in our case, $f(x)$ is the minimal number of active S-boxes for a given permutation $x$. A naive way to compute $f(x)$ would be to solve Model 1 with the permutation $x$. However, as mentioned before, solving this model is quite costly, which would results in a very slow meta-heuristic. Instead, we make the following observation. Let $n$ be the number of active S-boxes we want to prove, that is, we want to find a permutation for which the minimal number of active S-boxes is at least $n$. Then, given a certain permutation, we are only interested in one fact: does this permutation have a characteristic with a length strictly less than $n$? If so, then even if this characteristic is not a minimal one, we still know that this permutation will not reach our goal of a minimum of $n$ active S-boxes. This allows to slightly modify the original algorithm for a much quicker execution, which lead to more permutations being evaluated and thus better chances to find a good one. The complete algorithm is given as Algorithm 4, with a more detailed explanation below.

So instead of directly computing the minimal number of active S-boxes for a given permutation, we do the following. We first use the algorithm `quicksearch`, which is a classical dynamic programming algorithm which, given a permutation $x$ and a target number of S-boxes $n$, search for a *relatively* short characteristic of length $\leq n$. As mentioned before, the idea is to use the fact that

**Algorithm 4** Tweaked Simulated Annealing

**Input:** Target length $n$
$x \leftarrow$ *random permutation*, $T \leftarrow 2$, $l \leftarrow 0$
**while** $l < n$ **do**
  $\tau \leftarrow$ *random transposition*, $x' \leftarrow \tau \circ x$
  $l' \leftarrow \text{quicksearch}(x', n)$
  **if** $l' \geq n$ **then**
    $x \leftarrow x'$, $l \leftarrow \text{fullsearch}(x)$
  **else if** $l' > l$ **then**
    $x \leftarrow x'$, $l \leftarrow l'$
  **else**
    $r \leftarrow U[0,1]$ *Generate a uniformly random real number in [0,1]*
    **if** $r < e^{\frac{l'-l}{T}}$ **then**
      $x \leftarrow x'$, $l \leftarrow l'$
    **end if**
  **end if**
  $T \leftarrow \frac{T}{1+0.001T}$
**end while**
**Output:** $x$

we are mostly interested in whether or not a characteristic of length strictly less than $n$ exists. This algorithm performs this relatively quickly, without having to find *the* minimal number of S-boxes. Once we get such a characteristic of length $l'$, three cases can happen.

- If $l' \geq n$, then the permutation might be a good one. However, since the `quicksearch` algorithm does not return the length of the shortest characteristic, we need to call the `fullsearch` algorithm, which basically solves Model 1 using the provided permutation, and returns the real minimal number of S-boxes. If the output of `fullsearch` is greater or equal than $n$, then we found a permutation and the algorithm terminates. If not, we still choose to update $x$ to $x'$, because the fact that `quicksearch` returned a value greater or equal than $n$ means that the permutation looked quite good at first glance. We also update $l$ to the real minimal number of active S-boxes of $x$, since otherwise the algorithm would terminate while it did not find a permutation reaching $n$ S-boxes.
- Otherwise if $l' > l$, that is, the permutation $x'$ seems to have a minimal number of S-boxes greater than the previous one, then we update $x$ to $x'$ too. This corresponds to the case $f(x') > f(x)$ in the original Simulated Annealing algorithm.
- Finally, if $l' \leq l$, this is the same as the original algorithm. We accept the solution $x'$ and update $x$ to it only with a certain probability depending on the current temperature $T$ and the respective number of S-boxes found for $x$ and $x'$.

We first launched this algorithm using $n = 20$, and were able to find the permutation $P_k$ (given below) reaching this minimal number of S-boxes in about $2^{16}$ tries:

$$P_k = (8, 1, 7, 15, 10, 4, 2, 3, 6, 9, 11, 0, 5, 12, 14, 13).$$

Reaching 21 S-boxes is still an open question and for reference, we were able to test about $2^{24}$ permutations in several days. However, we were able to show that using $P_k$ as the key schedule,

| Number of rounds | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|
| Original key schedule | 1 | 3 | 9 | 11 | $13^{\dagger}$ | 15 |
| $P_{KLPS}$ | 1 | 5 | 10 | 14 | $18^{\dagger}$ | 22 |
| $P_k$ | 1 | 5 | 10 | 15 | $20^{\dagger}$ | 23 |

Table 3.2: Minimal number of S-boxes that our permutation $P_k$ reaches on a given number of rounds compared to the one from [84]. $^{\dagger}$No instantiation with a better probability than $2^{-128}$.

while only reaching a minimum amount of 20 S-boxes in the truncated setting, still guarantee that no characteristic with a probability better than $2^{-128}$ can be found when one use the DDT of the AES S-box. To do that, we used Model 2, which allows to check if there is a characteristic with a better probability than $2^{-128}$ and to exhibit one if that is the case. To make this model work, we need to give it a list of truncated differential characteristics, and it will check if such a characteristic can be instantiated with a probability better than $2^{-128}$. Hence, to prove that $P_k$ has no such characteristic, we need a list of all valid truncated characteristics of 20 and 21 S-boxes (since 22 S-boxes already guarantees that no characteristic will be instantiable with a probability better than $2^{-128}$). This can be computed rather quickly using Model 1 and asking the solver to find all characteristics of length 20 and 21. There are 253 characteristics of length 20 and 3284 of length 21. After about nine hours on a standard desktop to loop through all these characteristics, it turns out that none of them can be instantiated[2] with a probability better than $2^{-128}$. In conclusion, we were able to find a permutation $P_k$ such that using this permutation as the key schedule of AES-128 guarantees that no differential characteristic with a probability better than $2^{-128}$ exists over 6 or more rounds. For reference, we also ran Model 1 on this permutation to get the minimal number of active S-boxes for a lower amount of rounds, summarized in Table 3.2.

Now, even if we were able to find a permutation leading to no differential characteristic of probability better than $2^{-128}$ for 6 rounds or more, it still only reaches 20 S-boxes in the truncated setting. Hence, we would like to see if by modifying further the AES round function, we could reach more active S-boxes. This is treated in the next section.

## 3.5 Tweaking Both `ShiftRows` and the Key Schedule

Using the approach given in the previous section allowed to find a permutation for the key schedule, which induces a minimal number of S-boxes of 20 for 6 rounds. Here, we would like to see if by changing the `ShiftRows` operation in the AES-128, we could reach a better number of active S-boxes, namely 21 or 22. Obviously, we cannot try all possible permutations for `ShiftRows`, as again, there are $2^{44}$ permutations over 16 elements. Hence, we show here how we restricted ourselves to only a few thousand candidates for `ShiftRows`, which are the most likely to lead to a good minimal number of active S-boxes, and give a few examples of pairs $(P_s, P_k)$ that reach 21 S-boxes for 6 rounds, where $P_s$ is used instead of the `ShiftRows` operation, and $P_k$ instead of the original key schedule KS of AES.

First, we can see that we can drastically reduce the number of candidates for $P_s$ using the following two propositions. We denote $\mathcal{P}_i$ the set of all permutations $P_i$ acting insides the columns of the state, i.e., there exists four permutations $P_i^0, P_i^1, P_i^2, P_i^3$ over four elements such that $P_i^j$ acts on the $j$-th column and $P_i = P_i^0 \circ P_i^1 \circ P_i^2 \circ P_i^3$, and $\mathcal{P}_c$ the set of all permutations which permutes the columns of the state.

[2]For reference, the best probability we could reach among all the characteristics of length 20 was $2^{-134}$

**Proposition 3.5.1.** *Let $P_s$ and $P'_s$ be two permutations over 16 elements such that $P'_s = P'_i \circ P_s \circ P_i$, where $P_i, P'_i \in \mathcal{P}_i$, and let $P'_k = P_i^{-1} \circ P_k \circ P_i$. Then using $(P'_s, P'_k)$ instead of (SR, KS) will lead to the same minimal number of active S-boxes that using $(P_s, P_k)$ instead of (SR, KS). Hence, we can build equivalence classes $\mathcal{E}_i(P_s) = \{P'_s \mid \exists\, P_i, P'_i \text{ s.t. } P'_s = P_i \circ P_s \circ P'_i\}$, and there are 10147 such equivalence classes.*

*Proof.* We need to show that, for each characteristic we can build using $(P_s, P_k)$, one can find a characteristic with the same number of active S-boxes using $(P'_s, P'_k)$, where $P'_s = P'_i \circ P_s \circ P_i$ and $P'_k = P_i^{-1} \circ P_k \circ P_i$.

Given a characteristic $(X_0, \ldots, X_r)$ such that the length of the characteristic is given by $\sum_{i=0}^{r} X_i$, and denote $Y_i$ the state after the MC operation such that $X_{i+1} = Y_i \oplus K_i$. We have $Y_{i+1} = \mathsf{MC} \circ P_s \circ \mathsf{SB}(Y_i \oplus K_i)$ and $K_{i+1} = P_k(K_i)$, where $P_k$ is a bytewise permutation. For all $i$, let $K'_i = P_i^{-1}(K_i)$ and $Y'_i = P_i^{-1}(Y_i)$, hence we have

$$\begin{aligned} K'_{i+1} &= P_i^{-1}(K_{i+1}) = P_i^{-1} \circ P_k(K_i) \\ &= P_i^{-1} \circ P_k \circ P_i \circ P_i^{-1}(K_i) \\ &= P'_k \circ P_i'^{-1}(K_i) \\ &= P'_k(K'_i). \end{aligned}$$

So $P'_k$ is a valid key schedule. Furthermore, note that when considering the propagation of active bytes through MC, one only need to consider the number of active bytes before MC in one given columns to know the number of active byte after MC in that same column. Hence, since $P_i \in \mathcal{P}_i$ only permutes bytes inside each column, the number of active bytes does not change in each column and thus for any $P_i \in \mathcal{P}_i$, MC and $\mathsf{MC}' = \mathsf{MC} \circ P_i$ behave similarly when searching for truncated differential characteristics, i.e., replacing MC by $\mathsf{MC}'$ has no effect. In the same way, one can replace MC by $P_i \circ \mathsf{MC}$ with $P_i \in \mathcal{P}_i$. Moreover, SB acts on each byte separately, hence $P_i \circ \mathsf{SB} = \mathsf{SB} \circ P_i$. Thus, we have:

$$\begin{aligned} Y'_{i+1} &= P_i^{-1}(Y_i) = P_i^{-1} \circ \mathsf{MC} \circ P_s \circ \mathsf{SB}(Y_i \oplus K_i) \\ &= P_i^{-1} \circ \mathsf{MC} \circ P_s \circ \mathsf{SB}(P_i \circ P_i^{-1}(Y_i) \oplus P_i \circ P_i^{-1}(K_i)) \\ &= \mathsf{MC} \circ P_s \circ \mathsf{SB}(P_i(Y'_i) \oplus P_i(K'_i)) \quad \textit{replacing } P_i^{-1} \circ \mathsf{MC} \textit{ by MC has no effect} \\ &= \mathsf{MC} \circ P'_i \circ P_s \circ P_i \circ SB(Y'_i \oplus K'_i) \quad \textit{replacing MC by MC} \circ P'_i \textit{ has no effect} \\ &= \mathsf{MC} \circ P'_s \circ \mathsf{SB}(Y'_i \oplus K'_i). \end{aligned}$$

So $(P'_s, P'_k)$ correctly defines a round function and we have $X'_{i+1} = Y'_i \oplus K'_i = P_i^{-1}(Y_i \oplus K_i) = P_i^{-1}(X_{i+1})$ for all $i$. Hence, each $X'_i$ is a permutation of $X_i$, and thus the corresponding characteristic $(X'_0, \ldots, X'_r)$ has the same number of active S-boxes as $(X_0, \ldots, X_r)$. □

**Proposition 3.5.2.** *Let $P_s$ and $P'_s$ be two permutations over 16 elements such that $P'_s = P_c^{-1} \circ P_s \circ P_c$ where $P_c \in \mathcal{P}_c$, and let $P'_k = P_c^{-1} \circ P_k \circ P_c$. Then, using $(P'_s, P'_k)$ instead of (SR, KS) will lead to the same minimal number of active S-boxes that using $(P_s, P_k)$ instead of (SR,KS). Hence we can combine this with the previous proposition, and for each class representative $P_s$ of some class $\mathcal{E}_i(P_s)$ defined previously, we can build equivalence classes $\mathcal{E}(P_s) = \{P'_s \mid \exists\, P_c \in \mathcal{P}_c \text{ s.t. } P'_s = P_c^{-1} \circ P_s \circ P_c\}$, and there are 9186 such equivalence classes.*

*Proof.* As in the proof of Proposition 3.5.1, we need to show that, for each characteristic we can build using $(P_s, P_k)$, one can find a characteristic with the same number of active S-boxes using $(P'_s, P'_k)$, with $P'_s = P_c^{-1} \circ P_s \circ P_c$ and $P'_k = P_c^{-1} \circ P_k \circ P_c$, $P_c \in \mathcal{P}_c$.

Given a characteristic $(X_0, \ldots, X_r)$, and using the same notation as in the the proof of Proposition 3.5.1, for all $i$ let $K'_i = P_c^{-1}(K_i)$ and $Y'_i = P_c^{-1}(Y_i)$. Showing that $P'_k$ is a valid key-schedule is done in the same way as for Proposition 3.5.1. Furthermore, note that since $\mathsf{MC}$ acts on each column separately, we have $\mathsf{MC} \circ P_c^{-1} = P_c^{-1} \circ \mathsf{MC}$. In the same way, $\mathsf{SB}$ acts on each byte separately, hence $P_c \circ \mathsf{SB} = \mathsf{SB} \circ P_c$. Thus we have

$$\begin{aligned} Y'_{i+1} &= P_c^{-1}(X_{i+1}) = P_c^{-1} \circ \mathsf{MC} \circ P_s \circ \mathsf{SB}(Y_i \oplus K_i) \\ &= P_c^{-1} \circ \mathsf{MC} \circ P_s \circ \mathsf{SB}(P_c(Y'_i) \oplus P_c(K'_i)) \\ &= P_c^{-1} \circ \mathsf{MC} \circ P_s \circ \mathsf{SB} \circ P_c(Y'_i \oplus K'_i) \\ &= \mathsf{MC} \circ P_c^{-1} \circ P_s \circ P_c \circ \mathsf{SB}(Y'_i \oplus K'_i) \\ &= \mathsf{MC} \circ P'_s \circ \mathsf{SB}(Y'_i \oplus K'_i) \end{aligned}$$

So again, $(P'_s, P'_k)$ correctly defines a round function and $X'_{i+1} = P_c^{-1}(X_{i+1})$ for all $i$. Thus each $X'_i$ is a permutation of $X_i$, hence the corresponding characteristic $(X'_0, \ldots, X'_r)$ has the same number of active S-boxes as the characteristic $(X_0, \ldots, X_r)$. □

Hence, we only need to consider 9186 possible candidates $P_s$ to replace $\mathsf{SR}$, instead of $2^{44}$. Moreover, we would like to avoid weakening $\mathsf{AES}$ in the single-key model. In that model, the original `ShiftRows` allows to reach full diffusion after 3 rounds. So we only considered the permutations that also reached full diffusion in at most 3 rounds, and there are 4381 of them. Finally, recall that in the pure truncated differential setting, using the original `ShiftRows` implies that there is always a characteristic of length 18 which is built using a fully active key. While this characteristic has high chances of being invalidated once we consider the equations it implies on the key, we still would like to avoid it. To do that, we used the following proposition.

**Proposition 3.5.3.** *If one uses a permutation $P_s$ instead of `ShiftRows` such that $P_s$ send the bytes from any one column to at most three columns, then the characteristic from Proposition 3.3.3 cannot happen.*

*Proof.* The characteristic from Proposition 3.3.3 can be built because a state containing a single fully active column lead to a fully active state after $\mathsf{MC} \circ \mathsf{SR}$. However, if one uses a permutation $P_s$ which send the bytes from any one column to at most three columns, then the state after $\mathsf{MC} \circ P_s$ will contain at most 3 fully active column. Thus, when XOR-ing the key afterwards, the resulting state would have at least 4 active bytes, instead of 3 in the characteristic from Proposition 3.3.3, thus this characteristic cannot happen. □

Hence, we only want to try some permutations $P_s$ instead of `ShiftRows` which verify the previous propositions and achieve a full diffusion in at most 3 rounds in the single-key model, which lead to 3288 possible candidates for $P_s$. Now everything is quite straightforward. We reuse Algorithm 4 to search for a permutation leading to 21 S-boxes, except that we use a different permutation than `ShiftRows` in the `quicksearch` algorithm and modified Model 1 to use that permutation instead of `ShiftRows` for the `fullsearch` algorithm. We also added the additional condition that it should stop after 24 hours if no permutation reaching the objective was found. Surprisingly, the `quicksearch` algorithm ran faster with those permutations than with the original $\mathsf{SR}$, which allowed us to test about $2^{25}$ permutations $P_k$ on average in 24 hours for a specific candidate $P_s$. After a few more than 100 possible $P_s$ tried, we were able to find several pairs $(P_s, P_k)$ that reach 21 S-boxes which are given in Table 3.3.

For a given $P_s^i$, we also took a look at the permutations $P'_k$ that are rather "close" to the ones we found, that is, permutations $P'_k$ which are one or two transpositions away from each $P_k^i$. It turns

| $(P_s, P_k)$ | # iterations |
|---|---|
| $P_s^1 = (0,1,2,4,3,8,9,12,5,13,14,15,6,7,10,11)$<br>$P_k^1 = (10,4,12,11,6,2,5,1,8,0,9,7,13,14,15,3)$ | $3151253 \sim 2^{21.6}$ |
| $P_s^2 = (0,1,2,4,3,8,9,12,5,6,13,14,7,10,11,15)$<br>$P_k^2 = (15,14,11,10,6,12,4,0,3,8,1,9,2,5,13,7)$ | $42414349 \sim 2^{25.3}$ |
| $P_s^3 = (0,1,4,8,9,10,12,13,5,6,14,15,2,3,7,11)$<br>$P_k^3 = (14,12,8,6,7,4,0,1,3,11,10,2,9,5,13,15)$ | $8588115 \sim 2^{23}$ |
| $P_s^4 = (0,1,2,8,4,9,12,13,5,6,7,14,3,10,11,15)$<br>$P_k^4 = (12,14,11,4,8,0,3,7,10,15,2,9,6,13,5,1)$ | $15016901 \sim 2^{23.8}$ |
| $P_s^5 = (0,1,2,8,4,9,12,13,3,5,14,15,6,7,10,11)$<br>$P_k^5 = (5,9,15,13,3,4,6,2,11,7,10,0,8,14,1,12)$ | $51700477 \sim 2^{25.6}$ |

Table 3.3: Pairs $(P_s, P_k)$ which reach 21 S-boxes, along with the number of $P_k$ tried before founding it

out that, except for $(P_s^4, P_k^4)$, none of these permutations also reach 21 S-boxes. Oddly, there are 3 permutations that are 1 transposition away from $P_k^4$ which also reach 21 S-boxes when using $P_s^4$ instead of SR , and again, none of them has a differential characteristic with a probability better than $2^{-128}$ over 6 rounds. Those three permutations are

$$P_k^{4'} = (14,12,11,4,8,0,3,7,10,15,2,9,6,13,5,1),$$
$$P_k^{4''} = (12,14,11,4,10,0,3,7,8,15,2,9,6,13,5,1),$$
$$P_k^{4'''} = (12,14,11,4,8,0,3,7,2,15,10,9,6,13,5,1).$$

After testing about 1100 candidates for $P_s$, finding a pair $(P_s, P_k)$ that reaches 22 S-boxes is still an open problem. We also used Model 2, tweaked to use a different permutation instead of SR, to check if there is a differential characteristic with a probability better than $2^{-128}$ over 6 rounds with these pairs $(P_s, P_k)$, and again, none of these permutations allows such a characteristic.

## 3.6 Conclusion

In this chapter, we studied how AES would behave in the related-key model if we change its key schedule to a much simpler and efficient one, namely a permutation. We first gave a few generic bounds about the best number of active S-boxes reachable for a given number of round, and especially, we showed that no permutation can reach a minimal number of 18 or more active S-boxes over 5 rounds. However we were able to exhibit a permutation reaching 16 S-boxes over 5 rounds, hence closing the gap a bit further. We showed that we can find a permutation which allows to have at least 20 active S-boxes over 6 rounds, while guaranteeing that no characteristic with a probability larger than $2^{-128}$ exists. This allows us to reach the same amount round than with the original AES-128 key schedule (see [65]), but with a more efficient key schedule which is also easier to analyze and has a higher minimal number of active S-boxes. We also took a look at how modifying the SR operation could improve the minimal number of S-boxes over 6 rounds. It turns that we can find several pairs $(P_s, P_k)$ to use instead of SR and the key schedule (respectively) which allows to have at least 21 S-boxes over 6 rounds, and again, no characteristic with a probability better than $2^{-128}$.

We also provided a Constraint Programming model which can handle directly the equations coming from `MixColumns`, thus allowing to find the exact minimal number of active S-boxes considering everything but the S-box DDT in a reasonable amount of time and memory. Our implementation is available at `https://github.com/TweakAESKS/TweakAESKS`.

A few open questions remain. First, could we reach a minimal number of 22 active S-boxes changing only the key schedule (and possibly SR) for 6 rounds? In the same idea, could we close the gap for 5 rounds? We know that we cannot get 18 or more active S-boxes, but 16 S-boxes is reachable, thus the possibility of reaching 17 S-boxes is still unknown. Finally, we chose to change the SR operation, but how about changing either MC or the S-box? While changing everything would lead to a cipher that does not have much in common with AES, it could answer the following generic question: Can we build an AES-like SPN (with a round function structured as $\mathsf{MC} \circ P_s \circ \mathsf{SB}$ where $P_s$ is a permutation and MC uses an MDS matrix) using a permutation as the key schedule, which could reach either 22 S-boxes over 6 rounds, or guarantee that no characteristic with probability better than $2^{-128}$ exists over 5 rounds?

*"What's in the box ??"*

— David Mills

# Chapter 4

# On Recovering Affine Encodings in White-Box Implementations

This chapter focuses on two main parts. First, we propose a generic algorithm to recover affine encodings for any white-box implementation of a cipher following the CEJO framework, independent of the way the encodings are built. More generally, our algorithm solves the affine equivalence problem (given two maps $F$ and $S$ with the promise that they are affine equivalent, compute affine maps $\mathcal{A}$, $\mathcal{B}$, such that $F = \mathcal{B} \circ S \circ \mathcal{A}$) whenever one of the two maps is composed of the parallel application of distinct S-boxes. The general affine equivalence algorithm by Dinur [59] solves precisely this problem, without assuming any special structure on $S$ (this is also the case of the classic algorithm by Biryukov et al. [22]). However its complexity is $\mathcal{O}\left(n^3 2^n\right)$, which makes it unsuitable for recovering encodings on a typical block size of 128 bits. In contrast, we focus on the case where $S$ is made up of $k$ parallel $m$-bit S-boxes. In this setting, we propose an algorithm that solves the affine equivalence problem with a (typically much lower) time complexity of $\mathcal{O}\left(2^m n^3 + \frac{n^4}{m} + 2^m m^2 n\right)$. For the AES parameters $n = 128$, $m = 8$, $k = 16$, this yields a time complexity of $2^{32}$ basic operations[1] (to be compared with $2^{149}$ basic operations if the generic algorithm by Dinur was applied naively).

By design, our attack applies to a large class of white-box schemes following the CEJO framework, including [41, 42, 144, 83]. Beyond the previously cited schemes, which were already broken by ad-hoc attacks, we illustrate our attack on a new white-box design by Baek, Cheon and Hong [3]. One distinctive feature of this design that makes it particularly attractive to illustrate our attack (beside not being previously cryptanalyzed) is that it increases the state size by obfuscating two parallel rounds of AES, precisely to prevent generic attacks from being able to recover the affine encodings of the scheme. Indeed Baek et al. estimate the security level of their proposal to 110 bits based on their own specialized version of an affine equivalence algorithm. However our generic attack on this scheme requires only about $2^{35}$ basic operations.

As a second contribution, we analyze the scheme by Baek et al. more closely, and introduce another technique able to break this scheme. This new technique extracts and solves a standalone problem from the scheme by Baek et al. as follow. Let $F$, $h_1$, $h_2$ be three non-linear mappings from $m$ bits to $m$ bits, and let $A_1$, $A_2$ be two linear mappings on $m$ bits. Given oracle access to $G(x, y) = F(A_1(x) \oplus A_2(y)) \oplus h_1(x) \oplus h_2(y)$, recover $A_1$ and $A_2$ (up to equivalence). Ultimately, it is able to recover the secret key of the scheme in time complexity $2^{31}$. This is verified with an implementation, available at `http://wbcheon.gforge.inria.fr/`. This dedicated attack on Baek et al.'s scheme is also more powerful as it allows us to fully recover the key, while the generic attack only creates a decryption function without recovering the key.

[1] In practice the constants hidden in the $\mathcal{O}()$ notation for our algorithm are quite small, and we disregard them when giving complexity estimates.

## 4.1 Introduction

Historically, cryptanalysis is performed within the black-box model: the cryptographic algorithm under attack is executed in a trusted environment, and the view of the attacker is limited to the input-output behavior of the algorithm. Depending on the type of attack under consideration, the attacker may be able to observe the inputs and outputs of encryption or decryption queries, and perhaps choose the corresponding inputs, but nothing more. Such attack models are particularly relevant in scenarios where the attacker does not have direct access to an implementation of the scheme, whether because it is executed remotely, or within a protected hardware environment such as a secure enclave.

Since the advent of side-channel attacks however, new attack models have come into the light, wherein the attacker has access to some auxiliary information leaked by the implementation. These models are sometimes called *gray-box* models, in contrast with the *black-box* model outlined in the previous paragraph. Attacks in the gray-box model may exploit physical leakage such as computation time, power consumption, or electromagnetic leakage, among many others. Such attacks can result in practical breaks against schemes that would otherwise appear secure in the standard black-box model.

**White-box cryptography.** Going one step further, in 2002, Chow et al. introduced the white-box model [41, 42]. In this model, the attacker has full access to an implementation of the target cryptographic algorithm, including the ability to control its execution environment. Therefore he can observe memory content, set breakpoints in the execution flow, change arbitrary values in the code or the memory, *etc.* In this setting, the security assumptions of the black-box model clearly no longer hold. However, it may still be desirable that the adversary should be unable to extract the secret key of the cryptographic algorithm under attack.

This model is relevant in the context of software distribution, whenever a piece of software containing sensitive cryptographic information (such as an encryption algorithm) is to be widely distributed, and hence can be downloaded and analyzed by adverse parties. The most prominent application occurs in Digital Rights Management, where attackers may wish to recover a decryption key used to protect copyrighted content (digital music, TV broadcasts, video games, *etc*). A successful attacker is then able to distribute the secret key to unauthorized users, providing them with illegitimate access to the protected content. In effect, the goal is to protect sensitive functions within the deployed software, such as cryptographic algorithms, in much the same way that a trusted environment would protect security-critical functions in a hardware context. Ideally, white-box cryptography would thus achieve the software equivalent of trusted enclaves, specialized to particular cryptographic algorithms.

In order to achieve this goal, white-box cryptography techniques attempt to obfuscate the implementation of the target cryptographic algorithm. Ideally, an attacker in possession of the obfuscated cipher should be unable to interact with it in any meaningful way, beside simply executing it on chosen inputs. While Barak et al. have shown that general program obfuscation is impossible [7], the context of white-box cryptography presents two key differences. The first is that white-box cryptography merely attempts to obfuscate particular function families (such as block ciphers), which Barak et al.'s result has no bearing on. Another key difference is that white-box models do not generally require guarantees as strong as those offered by black-box obfuscation: in the case of a white-box implementation of AES for instance, it may be enough that the adversary is unable to recover the secret key (for a detailed discussion of white-box models, see e.g. [50, 66]).

**The CEJO framework.** In their original 2002 articles, Chow et al. proposed such a white-box scheme for DES and AES [41, 42]. While their proposals were quickly broken [77, 20], their work opened the path to white-box encryption. Follow-up works often reused the same general framework, which we will call the "CEJO framework".

In the CEJO framework, each round function is obfuscated by being composed with carefully crafted input and output encodings. That is, the round function $E^{(r)}$ at round $r$ is replaced in the white-box implementation by $f^{(r+1)^{-1}} \circ E^{(r)} \circ f^{(r)}$, where $f^{(r)}$, $f^{(r+1)^{-1}}$ are bijections called respectively the *input* and *output encoding*. By design, the output encoding of each round is canceled out by the input encoding of the next round.

$$\cdots \circ \underbrace{f^{(r+1)^{-1}} \circ E^{(r)} \circ f^{(r)}}_{F^r} \circ \underbrace{f^{(r)^{-1}} \circ E^{(r-1)} \circ f^{(r-1)}}_{F^{r-1}} \circ \cdots$$

Figure 4.1: The CEJO framework.

For each round, the white-box implementation gives access to the encoded version of the round function $F^r = f^{(r+1)^{-1}} \circ E^{(r)} \circ f^{(r)}$, but not directly to the underlying round function $E^{(r)}$.

Chow et al. proposed to define the encodings $f^{(r)}$ as the composition of a non-linear mapping and an affine mapping. The idea is to follow a classic concept in symmetric cryptography : the non-linear mapping will add some *confusion* on the intermediate values of the state, while the affine mapping will add some *diffusion* (see Sec. 3.3 and 3.4 in [42]). In addition, in a typical SPN block cipher, round keys are XORed into the inner state of the cipher. In that case, whenever the constant of the affine encoding is uniformly random, a single obfuscated round completely hides the value of the round key, which implies that a successful key-recovery attack must target multiple rounds simultaneously. Thus the CEJO framework is a natural approach to attempt to obfuscate a block cipher, especially in the case of SPN ciphers such as AES.

In addition to the above, some external input/output encodings $M_{out}/M_{in}$ can be added before and after the cipher. In that case, the implementation provides a map from encoded plaintexts to encoded ciphertexts. These encodings are merged into the tables used for the initial and final encoded round function. The implementation is then equivalent to an encoded version of the cipher, which can be expressed as $M_{out} \circ E^{(R)} \circ \cdots \circ E^{(1)} \circ M_{in}$.

External encodings can be used to increase security, as the attacker is denied direct access to raw plaintexts/ciphertexts. On the other hand, external encodings assume that the implementation surrounding the white-box cipher takes these encodings into account. As such, a white-box implementation with external encodings is not properly speaking an implementation of the cipher it contains. For this reason, in this work, we shall explicitly signal the presence of external encodings, and use the term white-box implementation *with external encodings* when appropriate.

It is crucial that, given the encoded round function $F^r$, the adversary should be unable to compute and peel off the encodings $f^{(r+1)^{-1}}$ and $f^{(r)}$. Indeed, for typical ciphers such as AES, granting direct access to a single round $E$ would allow the adversary to easily recover the corresponding round key, and from there the secret key of the cipher. However attacks on white-box implementations typically achieve precisely this, by taking advantage of the specific structure of the encodings $A$ and $B$. In white-box implementations following the CEJO framework, encodings are composed of a very simple non-linear layer, together with a more complex affine layer. Attacks generally peel off the non-linear component, then proceed to recover the affine layer. This is typically achieved in an ad-hoc way, by exploiting specific properties of the scheme under attack.

**Related Work.**

Literature on white-box cryptography, especially designs and attacks following the framework of Chow et al., is quite extensive. The first white-box candidate constructions by Chow et al. [41, 42] were quickly broken in practical time [77, 20].

In 2009, Xiao and Lai proposed to rely on *larger* affine encodings covering two S-boxes at once [144]. However, their proposal was broken in about $2^{32}$ operations by De Mulder et al. [48]. To thwart this attack, Karroumi proposed to use a dual representation of the AES round function in order to change the structure of each AES round [83]. But this was also broken in about $2^{22}$ operations by Lepoint et al. [98].

The previous attack also applies to the original scheme by Chow et al.; and another work by De Mulder et al. also provides improvement on the original BGE attack [49]. Note that all aforementioned attacks exploit the specific structure of the encodings used in the scheme under attack. As a result, they are more efficient than our generic algorithm, which works regardless of the structure of the encodings. Our algorithm also applies to these schemes and succeeds in practical time; but the point is that it is much more general: it does not require any structure in the affine encodings, and applies to all previous schemes at once, and more generally to all schemes in the CEJO framework. This includes Karroumi's scheme as it has been shown to be equivalent to the CEJO framework [49, 98].

A useful tool in the context of white-box cryptanalysis is the linear and affine equivalence algorithm by Biryukov et al. [22]. Their algorithm solves the following problem: given two bijections $S_1, S_2$ on $n$ bits, find affine (or linear, depending on the variant of the problem) mappings $\mathcal{A}$, $\mathcal{B}$ such that $S_2 = \mathcal{B} \circ S_1 \circ \mathcal{A}$, if they exist. Biryukov et al.'s algorithm is both able to ascertain whether such mappings exist, and enumerate all solutions. The time complexity of their solution is $\mathcal{O}\left(n^3 2^n\right)$ when $\mathcal{A}, \mathcal{B}$ are linear, and $\mathcal{O}\left(n^3 2^{2n}\right)$ when they are affine. In both cases, these complexities are practical when considering standard S-box sizes, such as $n = 8$.

This algorithm has been further improved in the affine case by Dinur [59], bringing the complexity down to $\mathcal{O}\left(n^3 2^n\right)$. Note however that this improved algorithm was designed for random permutations. Indeed, the AES S-box being self-affine equivalent, which is fairly rare in the random case, will lead to a failure of the algorithm. This was mentioned by the author, who also proposed a workaround. However our own implementation of the algorithm shows that it still fails on the AES S-box even when using the workaround. Hence, in that case of the AES S-box, we use the algorithm from [22] which has a higher complexity, but works on the AES S-box.

The main algorithm we propose in this chapter is essentially the same as the algorithm appearing in Section 2.3 of the structural cryptanalaysis of SASAS by Biryukov and Shamir [29]. However it is worth noting that this algorithm, from 2001, *predates* the first white-box constructions, due to Chow et al. in 2002; and a fortiori later constructions in the CEJO framework. Yet, to the best of our knowledge, it has not yet been clearly pointed out in the literature that this older algorithm actually solves the critical step in attacks on white-box schemes in the CEJO framework, as we show in this chapter. And indeed this algorithm is not referred to in any of the attacks mentioned above. Thus, we regard as a worthwhile contribution for practitioners in the field to point out that all known constructions in the CEJO framework can be uniformly broken (as far as recovering affine layers, which is the critical step in most cases) by combining this algorithm with a generic affine equivalence algorithm.

Our attack is also related to the attack by Minaud et al. [107] on the ASASA construction [21], as well as the followup work by Biryukov and Khovratovich [25]. However, the ASASA attack would only recover the output spaces of S-boxes, not their input spaces, which we also need. In the setting where the ASASA (and SASAS) attack was developed, this was inconsequential, because

the attacker had access to both the ASASA function and its inverse, so the problem was symmetric between input and output. However for us this is not the case: a key feature of our setting is that we only have access to an ASA mapping, but not its inverse. This difference is significant, as recovering the input spaces of the S-boxes from their output spaces seems as hard as breaking the scheme in the first place. And indeed, in the designs by Chow et al. to realize white-box AES and DES [41, 42], we are not aware of any way to invert the encoded round function without also breaking the scheme. In addition to qualitative differences in the setting considered, the algorithm by Minaud et al. is also more expensive for typical parameters (e.g. $n = 128$ or 256), as it costs about $2^m n^2 + n^6$ operations, where the last term is due to having to solve a quadratic system in $n$ variables. Running the ASASA algorithm on the scheme by Baek et al., recovering only the output spaces of S-boxes, would require $2^{48}$ operations instead of $2^{35}$ with our attack. Thus the SASAS algorithm [29], which we use, is the better approach in our setting.

At SAC 2008, Michiels, Gorissen and Hollmann also proposed a generic algorithm to break white-box implementations following the framework by Chow et al. [106]. Their work considers non-linear encodings, but requires two extra hypotheses: (1) the input space of each individual S-box through the input encoding should be known; and (2) the diffusion matrix of the scheme should satisfy a property called *disjoint spanning block sets*. In particular, that work does not solve the general problem of recovering arbitrary affine encodings surrounding a known S-box layer. Moreover, no overall complexity bound is provided[2], as some steps of the algorithm are not accompanied by a time complexity bound. There is also no implementation, which further prevents assessing performance.

The idea of considering a specialized variant of Biryukov et al.'s generic affine equivalence algorithm in the context we have described thus far (i.e. where the inner non-linear layer is composed of distinct S-boxes) was also proposed by Baek, Cheon and Hong in [3], who proposed the *specialized affine equivalence algorithm* (SAEA) for solving this problem. However, SAEA is very inefficient for larger $n$ in our setting, with a time complexity of $\mathcal{O}\left(\min(n^{m+4}2^{2m}/m, n\log(n)2^{n/2})\right)$. Baek et al. used SAEA to assess the security of their own white-box implementation with external encodings of AES, predicting a security level of $2^{110}$ operations. Our own generic algorithm, however, merely requires an estimated $2^{35}$ basic operations, breaking the scheme with practical complexity.

Incidentally, both the previously cited works by Michiels et al. and by Baek et al., while introducing interesting new techniques, also illustrate the lack of awareness around the fact that the SASAS technique by Biryukov and Shamir [29], combined with a generic affine equivalence algorithm, solves the ASA problem generically. In this respect our work may be regarded as filling a gap in the literature.

Finally, an interesting and recent line of work has exhibited side-channel attacks on white-box implementations [35, 4]. These approaches are quite powerful in that they require only "gray-box" access to the implementation, but are not generic attacks in the sense of our work. For example they are not applicable to the scheme by Baek et al. (not only because the scheme obfuscates two parallel executions of AES simultaneously, but also because it uses external encodings on both ends of the cipher). By nature this approach also relies on experimentation, rather than providing analytical bounds as we do.

Recent work in this direction has shed more light on the success of the gray-box approach outlined above, and studied more closely the effect of affine and non-linear encodings on the resistance of a white-box implementation against side-channel attacks [116, 31]. These works show that 4-bit non-linear encodings, which were recommended in the original scheme by Chow et al. for size reasons, are

[2] In Section 7, there is a claim that in the particular case of AES and Serpent, the time complexity of their algorithm would be dominated by the generic affine equivalence algorithm for each S-box. However that claim is not backed by any analytical bound, nor is it backed by an implementation.

insecure in that context. Both works focus their analysis mainly on non-linear encodings, and on the (practically highly relevant) case of a white-box implementation of AES following [42]. By contrast our work considers only affine encodings and requires full white-box access, but does so within a more general CEJO framework with an arbitrary SPN cipher and arbitrary (affine) encodings.

**Structure of the Chapter.**

In Section 4.2, we describe our generic algorithm to recover affine encodings in SPN ciphers in detail, together with its complexity analysis. In Section 4.3, we describe the white-box scheme by Baek et al.. In Section 4.4, we first point out that our algorithm from Section 4.2 breaks this scheme in a generic manner, then develop a second dedicated attack underpinned by a different technique, and discuss its implementation.

## 4.2 A Generic Algorithm to Recover Affine Encodings in SPN Ciphers

In this section, we present our algorithm for solving the affine equivalence problem in the case where the inner non-linear layer is composed of parallel S-boxes. As discussed in the introduction, solving this problem amounts to recovering affine encodings from a white-box implementation of any SPN cipher based on Chow et al.'s approach, regardless of the way the encodings are built. More precisely, our algorithm solves the following problem.

**Problem 1.** *Let $F$ be an $n$-bit to $n$-bit permutation such that $F = \mathcal{B} \circ S \circ \mathcal{A}$, where:*

1. *$\mathcal{A}$ and $\mathcal{B}$ are $n$-bit affine layers;*
2. *$S = (S_1, \ldots, S_k)$ consists of the parallel application of $k$ permutations $S_i$ on $m$ bits each (called S-boxes). Note that $n = km$.*

*Knowing $S$, and given oracle access to $F$ (but not $F^{-1}$), find affine $\mathcal{A}'$, $\mathcal{B}'$ such that $F = \mathcal{B}' \circ S \circ \mathcal{A}'$.*

Before we move on to the algorithm itself, a few remarks are in order.

**Remark 1.** First, our statement of the problem allows the algorithm to query $F$, but not $F^{-1}$. This is tailored to match the real situation of recovering an affine white-box encoding. Indeed, white box schemes following the CEJO framework allow access to $F$, but not to $F^{-1}$, as the output of $F$ is computed as a sum of some hard-coded table outputs, and inverting $F$ would require knowing how to split a given output of $F$ into the appropriate sum. To the best of our knowledge, the most straightforward way to achieve this is actually to break the scheme.

Of course, in other contexts, a variant of Problem 1 where the algorithm is granted access to both $F$ and $F^{-1}$ may also be worth considering. If $n$ is small, it should be noted that $F^{-1}$ can be computed exhaustively in $2^n$ operations, so if we are willing to pay $2^n$ calls to $F$, both variants of the problem become equivalent. In fact, our own algorithm will first isolate the input and output space of each S-box, then exhaust that space in $2^m$ operations for each S-box, which will allow us to access the inverse mapping of each S-box. Thus, essentially, our own algorithm will allow us to revert back to the case where the direct and inverse mappings are both available. In particular, it is not obvious how our algorithm could be improved even if $F^{-1}$ were accessible. In this regard, we note that Baek et al. explicitly provide an algorithm to solve Problem 1 when $F$ and $F^{-1}$ are both available, in $\mathcal{O}\left(n^4 2^{3m}/m\right)$ operations [3]. However this is slower than our algorithm for all reasonable parameter ranges, even though our algorithm does not require access to $F^{-1}$ (as noted

in the introduction, Baek et al. also propose an algorithm when only $F$ is accessible, but it is much slower).

**Remark 2.** As stated, Problem 1 asks to recover *some* affine encodings $\mathcal{A}'$, $\mathcal{B}'$ such that $F = \mathcal{B}' \circ S \circ \mathcal{A}'$, but not necessarily $\mathcal{A}$ and $\mathcal{B}$. This is because $\mathcal{A}$ and $\mathcal{B}$ may not be uniquely defined. In fact, if all S-boxes are identical (as is common in SPN ciphers), and as soon as there is more than one S-box, $\mathcal{A}$ and $\mathcal{B}$ *cannot* be uniquely defined: indeed, any solution $(\mathcal{A}, \mathcal{B})$ can be replaced by $(P \circ \mathcal{A}, \mathcal{B} \circ P^{-1})$, where $P$ is any permutation swapping S-box inputs. Problem 1 merely asks to recover *a* solution. However, because our algorithm eventually reduces the problem to the affine equivalence problem for each S-box, which is solved using the algorithm by Dinur, and that algorithm is able to enumerate all solutions if desired, it is straightforward to adapt our algorithm so that it outputs *every* solution.

**Remark 3.** The special case of Problem 1 where encodings are linear instead of affine may also be worth considering. As mentioned in the previous remark however, our algorithm eventually reduces Problem 1 to the affine equivalence problem for each S-box separately. As such, our algorithm can be trivially adapted to the linear variant of the problem by using a linear equivalence algorithm on each S-box, instead of an affine one.

**Remark 4.** In the special case where $k = 1$, i.e. $S$ is composed of a single S-box, Problem 1 is precisely the affine equivalence problem tackled by Biryukov et al. [22] and Dinur [59], with the caveat that $F^{-1}$ is not accessible. However, as mentioned in the introduction, the $\mathcal{O}\left(n^3 2^n\right)$ time complexity of the faster algorithm by Dinur precludes its use on full 128-bit blocks. From this perspective, the point of our algorithm is to achieve better time complexity, and in particular, practical complexity for $n$ upwards of 128 bits, by using the fact that $S$ is split into relatively small $m$-bit S-boxes.

### 4.2.1 Overview of the Algorithm

In a nutshell, the idea of the algorithm is to first isolate the input and output subspaces of each S-box, then apply the generic affine equivalence algorithm by Dinur to each S-box separately.

Thus, the first step of the algorithm is to find the input subspace of each S-box. More precisely, we want to build a subspace of dimension $m$ of the input space, such that this subspace spans all $2^m$ possible values at the input of a single fixed S-box, and yields a constant value at the input of all other S-boxes. To achieve this, we use a differential cryptanalysis approach. Namely, we pick uniformly at random an input difference $\Delta$. With probability $2^{-m}$, $\Delta$ yields a zero difference at the input of a particular S-box. We can easily ascertain whether this is the case by checking that the set of output differences generated by input difference $\Delta$ spans a subspace of dimension $n - m$. If that is the case, then $\Delta$ yields a zero difference at the input of one S-box, and non-zero differences at the output of all other $k - 1$ S-boxes[3].

By repeating this process a few times, we can eventually find $n - m$ linearly independent input differences that yield a zero difference at the input of the same S-box. By going through this process for each S-box, we recover $k$ spaces of dimension $n - m$, each yielding a zero difference at the input of a distinct S-box. Now if we pick any $k - 1$ of these spaces and compute their intersection, we obtain a space of dimension $m$ that yields a zero difference at the input of $k - 1$ S-boxes, and spans all values at the input of the remaining S-box. This is precisely the space we wanted to build.

[3]It should be noted that our algorithm makes a (very mild) assumption about the non-linearity of S-boxes: namely, we assume that, for most differences at the input of one S-box, the corresponding set of reachable output differences spans the whole output space of that S-box. In particular, this requires that the S-box does not have a linear approximation of probability one (in the sense of linear cryptanalysis). By construction, cryptographic S-boxes are expected to fulfill this requirement.

Indeed, if we query the overall permutation $F$ on all $2^m$ values forming such a subspace, we obtain a mapping that is affine equivalent to the corresponding S-box. It remains to apply the affine equivalence algorithm by Dinur to recover affine mappings witnessing the affine equivalence for that S-box. We repeat this process for all S-boxes. Finally we merge together the affine mappings thus recovered for each S-box to obtain the overall solution.

### 4.2.2 Description of the Algorithm

We will first detail our algorithm in the case that all S-boxes are the same, and then explain how to adapt it to the case of different S-boxes. The main idea to solve this problem is to find all input difference spaces $I_i$ which activate only one of the S-boxes. That is, for a difference $\Delta \in I_i$ and any message $x \in \mathbb{F}_2^n$, the difference after the application of $\mathcal{A}$, i.e. $\Delta' = \mathcal{A}(x) \oplus \mathcal{A}(x \oplus \Delta)$, is zero except on $m$ consecutive bits corresponding to the input of the $i$-th S-box. Indeed for such an input difference space $I_i \subset \mathbb{F}_2^n$, since the S-boxes are bijective, the output difference space $O_i = F(x) \oplus F(x \oplus I_i) \subset \mathbb{F}_2^n$ is of dimension $m$, for any $x \in \mathbb{F}_2^n$. Note that this output space $O_i$ does not depend on the choice of $x$. Therefore we can compute affine mappings $\mathcal{P}_i$ (from $\mathbb{F}_2^m$ to $I_i$) and $\mathcal{Q}_i$ (from $O_i$ to $\mathbb{F}_2^m$) such that $S' = \mathcal{Q}_j \circ F \circ \mathcal{P}_j$ is a bijection over $\mathbb{F}_2^m$ which is affine equivalent to the S-box $S$. We can then use the affine equivalence algorithm by Dinur to recover two affine mappings $\mathcal{A}_i, \mathcal{B}_i$ such that $S' = \mathcal{B}_i \circ S \circ \mathcal{A}_i$. By doing this for each S-box, we will be able to build two affine layers $\mathcal{A}'$ and $\mathcal{B}'$ such that $F = \mathcal{B}' \circ (S, \ldots, S) \circ \mathcal{A}'$.

**Computing the $I_i$'s.** To compute the input spaces that we are looking for, we will begin by computing all input spaces $V_i$ which activate at most $k-1$ S-boxes. More precisely, for $i$ from 1 to $k$ the space $V_i$ is such that, for any $\Delta \in V_i$ and $x \in \mathbb{F}_2^n$, we have that $\mathcal{A}(x) \oplus \mathcal{A}(x \oplus \Delta)$ is zero on $m$ bits corresponding to the input of the $i$-th S-box. There is $k$ such spaces and once we have them, we can recover all the input spaces $I_j$ by computing the intersection of $k-1$ spaces $V_i$.

**Computing the $V_i$'s.** We first remark that if we have a difference $\Delta \in V_i$, then the output vector space of differences $O_i$ will be of dimension $n-m$ instead of $n$ since one S-box will be inactive. This is the test we will use to construct the $V_i$'s. The idea is to pick a difference $\Delta$ at random as well as $n-m+l$ messages and then check whether the dimension of the output is lower or equal to $n-m$. For a large enough value $l$, a difference $\Delta$ will satisfy the condition if and only if it belongs to one of the $V_i$'s. Repeating this procedure enough time would allow us to fully recover the spaces $V_i$. However this would lead to a lot of rank computations. Instead we observe that, once we found an element of $V_i$, we can build the full output difference space $O_i$. Hence we compute a parity-check matrix of $O_i$, *i.e.* a matrix $H_i$ such that for any $x \in \mathbb{F}_2^n$, $H_i \cdot x = 0$ if and only if $x \in O_i$. This parity-check matrix can be used to quickly verify whether a vector belongs to $O_i$, and, as a result, whether a difference $\Delta$ belongs to $V_i$.

**Recovering affine layers.** The two previous steps allow us to build the spaces $I_i$ and $O_i$ that we were looking for. As described above, we thus get some affine mappings $\mathcal{A}_i, \mathcal{B}_i, \mathcal{P}_i, \mathcal{Q}_i$ for $i = 1 \ldots k$. Note that we do not know which S-box is activated by the space $I_i$, and thus one could think that we need to try all possible arrangement of those affine mappings. However this is not necessary, since we could always write $F$ as $F = \mathcal{B} \circ P^{-1} \circ (S, \ldots, S) \circ P \circ \mathcal{A}$ where $P$ is a permutation over the consecutive blocks of $m$ bits. Therefore, we build a block diagonal affine mapping $\mathcal{D}_\mathcal{A}$ (resp. $\mathcal{D}_\mathcal{B}$) where the blocks are the mappings $\mathcal{A}_1, \ldots, \mathcal{A}_k$ (resp. $\mathcal{B}_1, \ldots, \mathcal{B}_k$), as well as the two affine mappings $\mathcal{P}$ and $\mathcal{Q}$ built as

$$\mathcal{P} = (\mathcal{P}_1 | \ldots | \mathcal{P}_k), \ \mathcal{Q} = \left( \begin{array}{c} \mathcal{Q}_1 \\ \hline \vdots \\ \hline \mathcal{Q}_k \end{array} \right)$$

That way, we have that $\mathcal{D}_{\mathcal{A}} = \mathcal{A} \circ \mathcal{P}$ and $\mathcal{D}_{\mathcal{B}} = \mathcal{Q} \circ \mathcal{B}$ and thus by taking $\mathcal{A}' = \mathcal{D}_{\mathcal{A}} \circ P^{-1}$ and $\mathcal{B}' = Q^{-1} \circ \mathcal{D}_{\mathcal{B}}$, we have our equivalent function $F = \mathcal{B}' \circ (S, \ldots, S) \circ \mathcal{A}'$.
The whole algorithm is summarized as pseudo code in Algorithm 5

**Algorithm 5** Computing $\widetilde{A}$ and $\widetilde{B}$.

1: **for** $i = 1 \ldots k$ **do**
2: $\Delta \leftarrow$ random element in $\mathbb{F}_2^n$
3: $X \leftarrow \{n - m + l$ random elements in $\mathbb{F}_2^n\}$
4: $O_i \leftarrow F(X) \oplus F(X \oplus \Delta)$
5: **if** $(rank(O_i) > n - m)$ OR $(O_i = O_j$ for any $j < i)$ **then**
6: Go back to line 2
7: **else**[With probability $2^{-m}$]
8: $V_i = \{\Delta\}$ *$V_i$ will contain a basis of $n - m$ elements*
9: **while** $\#V_i < n - m$ **do**
10: $\Delta \leftarrow$ random element in $\mathbb{F}_2^n$ s.t. $\Delta \notin \text{span}(V_i)$ *$\sim 2^m$ values for $\Delta$*
11: $x \leftarrow$ random element in $\mathbb{F}_2^n$ *$l$ values for $x$*
12: **if** [ **then**Using a parity-check matrix of $O_i$]$F(x) \oplus F(x \oplus \Delta) \in Span(O_i)$
13: $V_i = V_i \cup \{\Delta\}$
14: **end if**
15: **end while**
16: **end if**
17: **end for**

18: **for** [ **do**$j = 1 \ldots k$]**each** intersection $I_j$ of $k - 1$ spaces $V_i$
19: Compute a $m$-bit to $n$-bit projection $\mathcal{P}_j$ from $\mathbb{F}_2^m$ to $I_j$
20: Compute a $n$-bit to $m$-bit projection $\mathcal{Q}_j$ from $O_j$ to $\mathbb{F}_2^m$
21: $S' \leftarrow \mathcal{Q}_j \circ F \circ \mathcal{P}_j$
22: *$S'$ is a bijection over $\mathbb{F}_2^m$ which is affine equivalent to $S$*
23: Use the affine equivalence algorithm from Dinur to recover two affine mappings $\mathcal{A}_j, \mathcal{B}_j$ of size $m$ such that $S' = \mathcal{B}_j \circ S \circ \mathcal{A}_j$
24: **end for**

25: $\mathcal{D}_{\mathcal{A}} \leftarrow diag(\mathcal{A}_1, \ldots, \mathcal{A}_k)$ *Block diagonal affine mapping with block size $m$*
26: $\mathcal{D}_{\mathcal{B}} \leftarrow diag(\mathcal{B}_1, \ldots, \mathcal{B}_k)$ *Block diagonal affine mapping with block size $m$*
27: $\mathcal{P} \leftarrow (\mathcal{P}_1 | \ldots | \mathcal{P}_k)$ $\mathcal{B}' = \mathcal{Q} \circ \mathcal{B}$
28: $\mathcal{Q} \leftarrow \begin{pmatrix} \mathcal{Q}_1 \\ \hline \vdots \\ \hline \mathcal{Q}_k \end{pmatrix}$ $\mathcal{A}' = \mathcal{A} \circ \mathcal{P}$
29: $\mathcal{A}' \leftarrow \mathcal{D}_{\mathcal{A}} \circ P^{-1}$ and $\mathcal{B}' \leftarrow Q^{-1} \circ \mathcal{D}_{\mathcal{B}}$ *That way, we have $F = \mathcal{B}' \circ (S, \ldots, S) \circ \mathcal{A}'$*

**Complexity of the algorithm.** The first step is to compute all vector spaces $V_i$. We can split this step into two parts. First, the computation of the output space $O_i$. Note that our test only checks whether $\Delta \in \cup_{j=1}^{k} V_j$, and this happens with probability $k2^{-m}$. Hence we need to try $2^m$ values for $\Delta$ on average to determine all the $k$ output spaces. Taking $n - m + l$ elements in $X$ leads to a probability of a false positive, i.e. $rank(O_i) = n - m$ while $\Delta$ activates all S-boxes, of $2^{-ml}$ for one value of $\Delta$. The effective value of $l$ will depend on the overall probability of failure that we wish to achieve for the whole algorithm and will be detailed below. Then computing the rank of $O_i$ can be done in $(n - m + l)^2 n = \mathcal{O}(n^3)$ operations. All in all, the computation of the output spaces

$O_1, O_2, \ldots, O_k$ has complexity

$$\mathcal{O}\left(2^m n^3\right).$$

The second part is to compute a basis of the input space $V_i$ which is of dimension $n-m$. To get each of those $n-m$ vectors (minus $\Delta_0$ which we already know), we first remark that as above, the probability that a difference $\Delta$ is valid is $2^{-m}$, hence $2^m$ tries for $\Delta$. Each value of $\Delta$ will be tested using $l$ values of $x$, leading to a probability of false positive of $2^{-ml}$ for one specific $\Delta$. The parity-check matrix of $O_i$ can be computed at the same time as the rank computation, and thus adds no cost here. This matrix is of size $m \times n$, therefore checking if one output difference belongs to $O_i$ costs about $\mathcal{O}(mn)$ operations. Therefore, using that $n = km$, the complexity of computing the basis of size $n-m$ for each of the $k$ spaces $V_i$ is

$$\mathcal{O}(k(n-m)2^m lmn) = \mathcal{O}(2^m klmn^2) = \mathcal{O}(2^m ln^3).$$

Computing all intersections of $(k-1)$ vector spaces $V_i$ can be done in $\mathcal{O}(kn^3)$ operations as shown latter in this section. Then, we need to make $k$ calls to the affine equivalence algorithm, which leads to a complexity of $\mathcal{O}(km^3 2^m)$. All in all, the total complexity of our algorithm is

$$\mathcal{O}\left(2^m n^3 + 2^m ln^3 + \frac{n^4}{m} + 2^m m^2 n\right).$$

As mentioned previously, the algorithm from [59] was designed for random permutations. This algorithm has a certain probability to fail, which is higher when the size of the S-box is low, or when the affine equivalence problem has multiple solutions, which is the case for the AES S-box since it is self-affine equivalent. This was mentioned by the author, along with a trick which could make the algorithm work on the AES S-box. However, we did implement this trick, along with further tweaking, and the algorithm would still fail for this specific choice of S-box. Hence, if the algorithm from Dinur fails, one would need to use the algorithm from Biryukov et al. [22], which raises the complexity to

$$\mathcal{O}\left(2^m n^3 + 2^m ln^3 + \frac{n^4}{m} + 2^{2m} m^2 n\right).$$

**Computing all Intersections of $k-1$ Subspaces among $k$ Subspaces**

In the previous algorithm, we have $k$ vector spaces $V_i \subset \mathbb{F}_2^n$ of dimension $n-m$, and we need to compute each intersection of $(k-1)$ spaces $V_i$. Let $B_i$ be the $n \times (n-m)$ matrix such that its columns are the vector of a basis of $V_i$. In order to save computations, we begin by echelonizing each matrix $(B_i \mid I_n)$ where $I_n$ denote the identity matrix of size $n$. This leads to matrices with the following structure:

$$\left(\begin{array}{c|c} I_{n-m} & C_i \\ \hline 0 & D_i \end{array}\right),$$

where $C_i$ is a matrix of size $(n-m) \times n$ and $D_i$ is of size $m \times n$. We note that a vector $x$ belongs to $V_i$ if and only if it belongs to Ker $D_i$. Therefore, with $D$ the matrix built as

$$D = \begin{pmatrix} D_1 \\ \vdots \\ D_k \end{pmatrix},$$

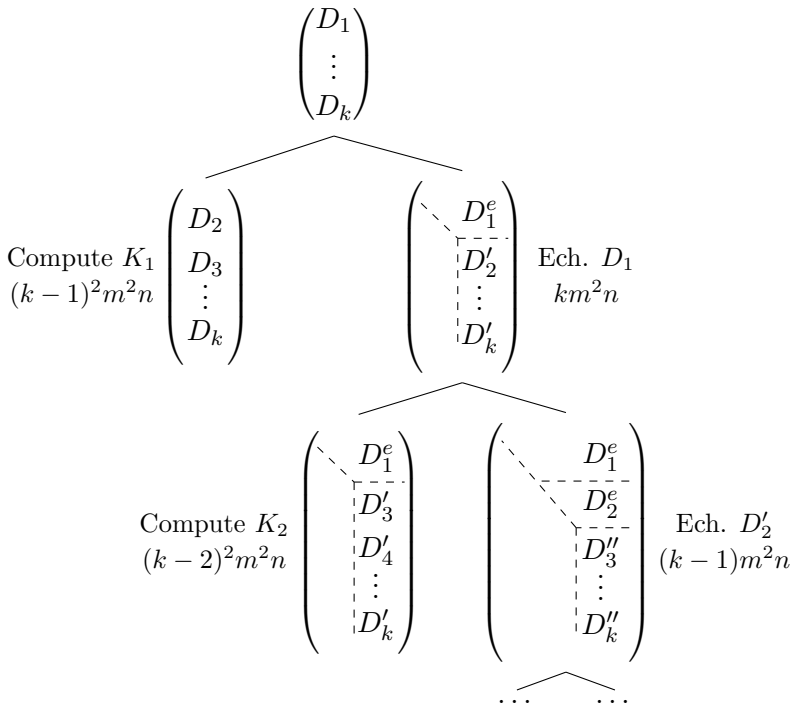


Figure 4.2: Efficient computation of the kernels

if $x \in \text{Ker } D$, then for all $i$, $x \in \text{Ker } D_i$, which leads to $x \in V_i$ and thus $x \in V_1 \cap \cdots \cap V_k$.

In our case, we do not need the intersection of all $V_1, \ldots, V_k$, but all the intersection of $k-1$ spaces $V_i$. To do so, instead of building $D$ from *all* the matrices $D_i$, one can build $D$ from only $k-1$ matrices $D_i$, leading to the intersection $\bigcap_{i \neq j} V_i$ for each $j = 1 \ldots k$.

The complexity of this whole computation is as follows. We first need to echelonize each $(B_i|I_n)$ on their first $n-m$ columns. Note that this computation can be done at the same time as the line 10 in the previous algorithm: since we need to draw $\Delta$ linearly independent from the previous computed vectors of $V_i$, we can echelonize the basis of $V_i$ as we build it. Since $B_i$ is of size $n \times (n-m)$, the cost of doing this for each $i$ is thus $kn^2(2n-m) = \mathcal{O}(kn^3)$. Then we need to compute the kernel of the matrices $D$ built from $k-1$ matrices $D_i$. Note that, those matrices being of size $(k-1)m \times n$, computing the $k$ kernels needs about $((k-1)m)^2 n = \mathcal{O}(n^3)$ operations. However, by doing this in a clever way, one can avoid repeating the same computations and thus improve the constant hidden in the $\mathcal{O}()$ notation.

First, denote by $K_i$ the kernel computed from the matrices $D_j$ with $j \neq i$, and

$$D = \begin{pmatrix} D_1 \\ \vdots \\ D_k \end{pmatrix}.$$

Remark that computing $K_i$ with $i \neq 1$ (i.e. all kernels containing $D_1$) is the same as removing one block $D_j, j \neq 1$ from $D$ and echelonizing the resulting matrix. Thus by doing this naively, one would echelonize several times from the $m$ rows of $D_1$. So we want to avoid those redundant computation. Therefore, we first echelonize $D$ on the $m$ rows of $D_1$, leading to the matrix $D'$ with

the following structure

$$D' = \begin{pmatrix} \ddots & D_1^e & \\ & D_2' & \\ & \vdots & \\ & D_k' & \end{pmatrix}$$

This matrix $D'$ can be used to compute all kernels containing $D_1$ by removing one of the block $D_j'$. Again, doing this naively would result in a lot of redundant echelonization on the rows of $D_2'$, therefore we repeat the previous procedure by echelonizing $D'$ on the rows of $D_2'$ once for all, leading to a matrix $D''$ which we will use to compute all the kernels containing $D_1$ and $D_2$. A summary of this procedure is depicted in the Figure 4.2, along with the complexity of each step in the tree. To be more precise about this complexity, let give a look at the operations we need to do on the $i$-th level of the tree. We need to compute the kernel $K_i$ from a $(k-1)m \times n$ matrix which has already be echelonized on $im$ rows, thus leading to a complexity of $(k-1-i)^2m^2n$ operations. We also need to echelonize on $m$ rows of a matrix of total size $km \times n$, which is also already echelonized on $im$ rows, which needs $(k-i)m^2n$ operations. Therefore, the total complexity of this way to compute the kernels is

$$\sum_{i=0}^{k-2}(k-i-1)^2m^2n + (k-i)m^2n = \frac{m^2n\left(k^3+2k-1\right)}{3}$$

To compare, the naive way to do, i.e. computing each kernel independently, would lead to a complexity of

$$k(k-1)^2m^2n = m^2n(k^3-2k^2+k)$$

leading to a speed up of about 3 for this step.

**Distinct S-boxes.**

In the analysis so far, we have assumed that all $k$ S-boxes are identical. We now discuss how the previous algorithm can be adapted to handle the case of different S-boxes. The only step where the concrete definition of the S-boxes is used is when we want to use the affine equivalence algorithm by Dinur. Recall that given two permutations $S$ and $S' = \mathcal{B}_i \circ S \circ \mathcal{A}_i$ where $\mathcal{A}_i$ and $\mathcal{B}_i$ are unknown, this algorithm finds $\mathcal{A}_i', \mathcal{B}_i'$ such that $S' = \mathcal{B}_i' \circ S \circ \mathcal{A}_i'$. Note that when we compute the input spaces $I_j$, we do not know which S-box remains active. Thus for each $I_j$, we need to call the affine equivalence algorithm for each possible S-box $S_i$ which may seems costly. However, the affine equivalence algorithm scales very well in the case we want to search for an equivalence from a set of S-boxes. That is, given $S' = \mathcal{B}_i \circ S_i \circ \mathcal{A}_i$ where $i$ is also unknown, find which S-box is affine equivalent to $S'$.

To compute the affine equivalence between two S-boxes $S$ and $S'$, we use the algorithm from [22]. However, our problem is actually to find affine equivalences between two sets of $k$ S-boxes: namely the $k$ S-boxes $S_1, \ldots, S_k$ known in advance, and the $k$ S-boxes $S_1', \ldots, S_k'$ which we recover from $F$ using our algorithm. Each one of the $S_i$'s is affine equivalent to one of the $S_j'$'s, but we do not know a priori which one. We could simply try all $\binom{k}{2}$ possible matches, however there is a better algorithm. Indeed, as observed in [22, Section 3.1], in this setting their algorithm can be made to only grow linearly in $k$, rather than quadratically. A brief summary of how this is achieved is provided in Algorithm 6; we refer to [22] for more details about how a canonical representative is computed. For simplicity, Algorithm 6 only outputs the set of affine equivalent pairs, but it can be easily modified to also output the corresponding affine mappings. In the end, we have an overall

complexity $\mathcal{O}(km^3 2^{2m})$ to match all pairs of S-boxes. Note that the same idea can be applied to the improved affine equivalence algorithm from Dinur [59], which thus lead to a complexity of $\mathcal{O}(km^3 2^m)$.

**Algorithm 6** Given $S_1, \ldots, S_k$, and $S'_1, \ldots, S'_k$ find all affine equivalent pairs $(S_i, S'_j)$.

```
T ← empty map
for i = 1 ... k do
    for all a ∈ F_2^m do
        R ← canonical representative of the linear equivalence class of S_i ⊕ a
        Append i to T[R] (viewed as a set)
    end for
end for
for j = 1 ... k do
    for all b ∈ F_2^m do
        R ← canonical representative of the linear equivalence class of S'_j ⊕ b
        for i in T[R] do
            Output (i, j)
        end for
    end for
end for
```

All in all, this adds a factor $k$ in the overall complexity, which becomes

$$\mathcal{O}\left(2^m n^3 + 2^m l n^3 + \frac{n^4}{m} + k 2^{2m} m^2 n\right) = \mathcal{O}\left(2^m l n^3 + \frac{n^4}{m} + 2^{2m} m n^2\right)$$

when using the algorithm from Biryukov et al., and $\mathcal{O}\left(2^m l n^3 + \frac{n^4}{m} + 2^m m n^2\right)$ when using the improved affine equivalence algorithm from Dinur.

### Probability of failure

We now study the probability of failure of our main algorithm, Algorithm 5. In this algorithm, the number of messages we use is parametrized by the value $l$, and the probability of failure decreases with $l$. Failures in our algorithm stem e.g. from generating $n - m + l$ output differences activating all S-boxes, and these output differences spanning a subspace of dimension $n - m$ despite all S-boxes being active. Intuitively, it seems clear that the probability of such an event decreases exponentially with $l$. However the exact probability of a failure depends on the S-boxes under consideration, and more specifically, it depends on their differential distribution table. As a result, an exact analysis of the failure probability is quite complex.

In what follows, to keep the analysis in check, when a random input difference activates all S-boxes, we approximate output differences by uniformly random vectors. We submit that for cryptographic S-boxes, this is a reasonable approximation of reality as far as the dimension of the output space is concerned, which is what matters for our algorithm. Moreover, we have successfully run experiments (using the AES S-box, as well as random ones) to validate that failure probability behaves as expected.

**During the computation of the $O_i$'s.** When we search the output space $O_i$, we draw $n - m + l$ random elements to test whether the output space is of dimension lower or equal to $n - m$. Here, a false positive would be a difference $\Delta$ such that $\text{rank}(O_i) = n - m$ while $\Delta$ activates all S-boxes.

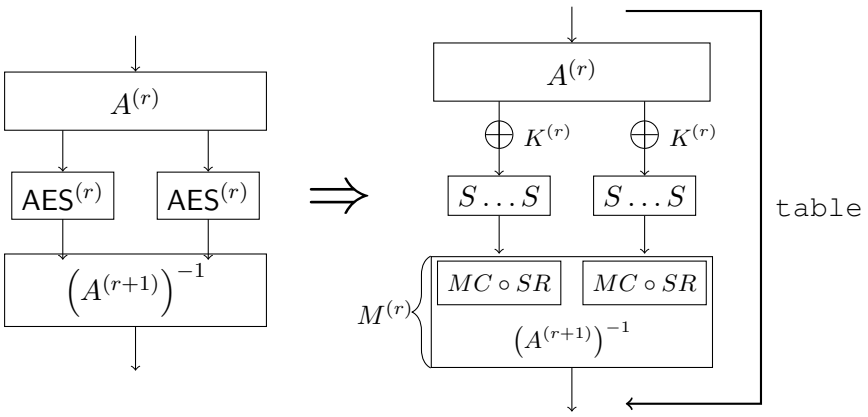


Figure 4.3: The Baek *et al.* proposal

Therefore the probability of a false positive at this step is upper-bounded by $2^{-ml}$ for *one* value of $\Delta$. Since we do this step for about $2^m$ values of $\Delta$, the probability that a false positive occurs in this step over all the algorithm is upper bounded by $2^m 2^{-ml} = 2^{m(1-l)}$.

**During the computation of the $V_i$'s.** For each value of $\Delta$, we want to test whether $F(x) \oplus F(x \oplus \Delta) \in \text{span}(O_i)$ for $l$ values of $x$. In that case, a false positive is a value of $\Delta$ such that this test is verified while $\Delta$ activates all S-boxes. Again, since $\dim(O_i) = n - m$, the probability of a false positive of a specific value of $\Delta$ is $2^{-ml}$. We try about $2^m$ values of $\Delta$ on average, and need to do this to find all the $n - m$ basis vectors for each of the $k$ spaces $V_i$. So the probability of a false positive at this step is upper bounded by $k(n-m)2^{m(1-l)}$.

**Overall failure probability.** The probability of failure of our algorithm is upper-bounded by the sum of the two previous probabilities, which is to say:

$$(k(n-m)+1)2^{m(1-l)}.$$

As noted in Section 4.2.2, for the Baek et al. proposal, the parameters are $n = 256, m = 8$ and $k = 32$. Thus, using only $l = 5$ messages, the failure probability is $2^{-16}$. In practice, failures are not a concern: in our experiments we set $l = 5$, and never encountered a failure.

## 4.3 Description of the White-Box Scheme by Baek et al.

Baek et al. provide a toolbox to break any white-box scheme in the CEJO framework [3]. Their results suggest that the main weakness in the previous proposals for white-box AES is the size of the internal state. Thus, they proposed to concatenate two AES instances, and encode them together in order to increase the size of the internal state (Fig. 4.3). We note that their proposal is a white-box scheme with external encodings.

Baek et al. also showed that the cost of removing the non-linear encodings is lower than recovering the affine encodings, so they focused only on designing affine encodings. Let us recall the round function of AES, denoted as $\mathsf{AES}^{(r)}$, built from the four sub-steps AddRoundKey(ARK), SubBytes(SB), ShiftRows(SR) and MixColumns(MC):

$$\mathsf{AES}^{(r)} = \begin{cases} \mathsf{MC} \circ \mathsf{SR} \circ \mathsf{SB} \circ \mathsf{ARK}, & \text{if } r = 1, \ldots, 9, \\ \mathsf{ARK} \circ \mathsf{SR} \circ \mathsf{SB} \circ \mathsf{ARK}, & \text{if } r = 10. \end{cases}$$

Thus, the encoded round function is the 256-bit to 256-bit mapping

$$F^{(r)} = \left(\mathcal{A}^{(r+1)}\right)^{-1} \circ \left(\mathsf{AES}^{(r)}, \mathsf{AES}^{(r)}\right) \circ \mathcal{A}^{(r)},$$

where $A^{(r)}$ are affine mappings on 256 bits. However, using a random affine mapping would result in some impractical tables since these mappings are of input size 256.

Therefore, they proposed to build 32 tables from 16 bits to 256 bits for each round, using some structured affine mappings as follows: Let $A^r$ be an invertible linear map of dimension 256 over $\mathbb{F}_2$, and denote the $(i,j)$-th $8 \times 8$ block of $A^r$ by $A^r_{i,j}$, $i,j = 0,\ldots,31$. Then $A^r$ is built such that $A^r_{i,j}$ is the zero matrix for all $(i,j) \neq (i,i), (i,i+1)$ and $(31,0)$. Finally, let $a^r = (a^r_0,\ldots,a^r_{31})$ be a random 256-bit vector, where each $a^r_i$ is an 8-bit block. Then we define the input encoding of the $r$-th round $\mathcal{A}^{(r)}$ with:

$$\mathcal{A}^{(r)}(x) = A^r \cdot x \oplus a^r = \begin{pmatrix} A^r_{0,0} & A^r_{0,1} & 0 & 0 & \ldots & 0 \\ 0 & A^r_{1,1} & A^r_{1,2} & 0 & \ldots & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ A^r_{31,0} & 0 & 0 & 0 & \ldots & A^r_{31,31} \end{pmatrix} \begin{pmatrix} x_0 \\ x_1 \\ \vdots \\ x_{31} \end{pmatrix} \oplus \begin{pmatrix} a^r_0 \\ a^r_1 \\ \vdots \\ a^r_{31} \end{pmatrix}. \tag{4.1}$$

To generate the tables, we will merge $\left(\mathcal{A}^{(r+1)}\right)^{-1}$ with the linear part of AES, that is, we define $\mathcal{M}^{(r)} = \left(\mathcal{A}^{(r+1)}\right)^{-1} \circ (\mathsf{MC} \circ \mathsf{SR}, \mathsf{MC} \circ \mathsf{SR})$ which is an affine mapping of size 256. Then, as depicted on Fig. 4.3 our encoded round function becomes $F^{(r)} = \mathcal{M}^{(r)} \circ (S,\ldots,S) \circ \mathsf{ARK} \circ \mathcal{A}^{(r)}$ for $r = 1,\ldots,9$, where $K^r$ is the $r$-th round key. The last round ($r = 10$) is slightly different and will be treated in a later part.

**Table construction.** We split the linear part of $M^{(r)}$ into 32 linear blocks of size $256 \times 8$ $M^r_i$ such that $\mathcal{M}^{(r)}(x) = (M^r_0,\ldots,M^r_{31}) \cdot x \oplus m^r$ where $m^r$ is a 256-bit vector representing the affine part of $\mathcal{M}^{(r)}$. Also take 31 random 256-bit vectors $m^r_i, i = 0,\ldots,30$ and $m^r_{31} = m^r \oplus m^r_0 \oplus \cdots \oplus m^r_{30}$. Then for $i = 0,\ldots,31$, we have the 16-bit to 256-bit tables $F^{(r)}_i$ defined as:

$$F^{(r)}_i = \mathsf{AC}_{m^r_i} \circ M^r_i \circ S \circ \mathsf{AC}_{K^r_i \oplus a^r_i} \circ \begin{pmatrix} A^r_{i,i} & A^r_{i,i+1} \end{pmatrix}$$

where $\mathsf{AC}_a$ is defined as $\mathsf{AC}_a(x) = x \oplus a$ and the index are taken modulo 32 when necessary. Thus, one can evaluate the encoded round function $F^{(r)}$ as the sum of $F^{(r)}_i$:

$$F^{(r)}(x_0, x_1, \ldots, x_{31}) = \bigoplus_{i=0}^{31} F^{(r)}_i(x_i, x_{i+1}).$$

Therefore to implement our encoded round function $F^{(r)}$, instead of having an unreasonable 256-bit to 256-bit table, we juste need to store 32 tables from 16 bits to 256 bits.

However, the partial application $F^{(r)}_i(x,0) = \mathsf{AC}_{m^r_i} \circ M^r_i \circ S \circ \mathsf{AC}_{K^r_i \oplus a^r_i} \circ A^r_{i,i}(x)$ is an 8-bit to 256-bit mapping which can be reduced to an 8-bit bijection by applying a projection. Then it is affine equivalent to $S$, and one can efficiently recover the affine mappings with the affine equivalence algorithm described in [22] in about $2^{25}$ operations. To prevent this weakness, Baek *et al.* proposed to replace $F^{(r)}_i$ by $T^{(r)}_i$ such that

$$T^{(r)}_i(x,y) = F^{(r)}_i(x,y) \oplus h^{(r)}_i(x) \oplus h^{(r)}_{i+1}(y),$$

where $h_i^{(r)}$ is a random 8-bit to 256-bit function, and we get

$$\bigoplus_{i=0}^{31} T_i^{(r)}(x_i, x_{i+1}) = \bigoplus_{i=0}^{31} F_i^{(r)}(x_i, x_{i+1}) = F^{(r)}(x_0, x_1, \ldots, x_{31})$$

using the fact that the index are taken modulo 32. We will later see that this choice was not enough to hide the structure of $F_i^{(r)}$.

**External encodings.** Consider two random 256-bit affine functions $\mathcal{M}_{in}$ and $\mathcal{M}_{out}$. The external input encoding function is then defined by $F^{(0)} = \left(\mathcal{A}^{(1)}\right)^{-1} \circ \mathcal{M}_{in}$, which is implemented with a $256 \times 256$ matrix and a 256-bit vector. The external output encoding $\mathcal{M}_{out}$ allows us to define the last encoded round function as

$$F^{(10)} = \mathcal{M}_{out} \circ \left(\mathsf{AES}^{(10)}, \mathsf{AES}^{(10)}\right) \circ \mathcal{A}^{(10)},$$

where $\mathsf{AES}^{(10)} = \mathsf{AC}_{K^{11}} \circ \mathsf{SR} \circ (S, \ldots, S) \circ \mathsf{AC}_{K^{10}}$.

This function is then split into 32 tables $T_i^{(10)}$ using the same technique as above. That way, we have

$$F^{(10)} \circ \cdots \circ F^{(1)} \circ F^{(0)} = \mathcal{M}_{out} \circ (\mathsf{AES}, \mathsf{AES}) \circ \mathcal{M}_{in}.$$

Since one encoded round function is implemented with 32 tables from 16 bits to 256 bits, the memory required for each encoded round function is

$$32 \times 2^{16} \times 256 \text{ bits} = 64 \text{ MB},$$

leading to 640MB for the full scheme with external encodings. In their paper, Baek *et al.* evaluate the security of this construction to $2^{110}$ using their toolbox. However, as we will show in the next section, we are able to decrypt any message in $\sim 10 \times 2^{30}$ operations, and fully break this construction by recovering the key in $\sim 2^{31}$ operations.

## 4.4 Cryptanalysis of the Scheme by Baek et al.

Baek *et al.* assessed the security level of their proposition to 110 bits. Recall that each encoded round function is of the form $F = \mathcal{M} \circ (S, \ldots, S) \circ \mathcal{A}$ where $\mathcal{M}$ and $\mathcal{A}$ are affine mappings. Therefore, our generic algorithm from Section 4.2 can be used to compute an equivalent round function $F = \mathcal{M}' \circ (S, \ldots, S) \circ \mathcal{A}'$ where $\mathcal{A}'$ and $\mathcal{M}'$ are known affine mappings, in about $\sim 2^{34.6}$ operations. However, one can exploit the specific structure of the encodings to mount a more efficient dedicated attack on their scheme. We will first begin by giving a method of complexity $\sim 2^{30}$ to recover a computationally easy to invert equivalent representation of one encoded round function. Next, we will show that instead of using this method 10 times (for each round function), we are able to fully break this scheme in $\sim 2^{31}$ operations, that is, recovering the secret key used in the underlying AES as well as the external encodings $\mathcal{M}_{in}$ and $\mathcal{M}_{out}$.

### 4.4.1 Building an Equivalent Representation of the Scheme

Let us consider one encoded round function and drop the exponent notation for the round as it is not relevant here, and also merge the key addition with the input affine encoding. Given an encoded round function $F$ of the form $\mathcal{M} \circ (S, \ldots, S) \circ \mathcal{A}$ where $\mathcal{M}$ and $\mathcal{A}$ are secret affine mappings, and $\mathcal{A}$ has the structure depicted in (4.1), our goal is to provide a computationally easy to invert representation

of $F$, that is, finding two equivalent affine mappings $\mathcal{M}'$ and $\mathcal{A}'$ such that $F = \mathcal{M}' \circ (S, \ldots, S) \circ \mathcal{A}'$. In that case, inverting one round would only cost two inversions of 256-bit affine mappings. Remember that the encoded round function is hidden in the tables $T_i(x, y) = F_i(x, y) \oplus h_i(x) \oplus h_{i+1}(y)$ where $h_i$ are random functions.

**Reducing the Problem to Block Diagonal Input Encodings**

Finding the input encoding can easily be done if this encoding is a block diagonal affine mapping where each block is of size 8. By applying an appropriate projection, one can obtain some 8-bit bijections that are affine equivalent to the AES S-box. In that case, recovering the affine mappings used can be done in about $2^{25}$ operations with the affine equivalence algorithm from [22]. Because of the random mappings $h_i$, one cannot use this algorithm directly on the tables in the Baek et al. proposal. However, we will show that we can decompose the secret input encoding $\mathcal{A}$ in $\mathcal{A} = \mathcal{B} \circ \widetilde{A}$ where:

- $B$ is a secret block diagonal affine mapping, built from blocks $B_i$ of size $8 \times 8$,
- $\widetilde{A}$ is a known linear mapping which has the same structure as $\mathcal{A}$ (4.1).

Let us denote the 16-bit to 8-bit linear mapping $L_i = (A_{i,i}\ A_{i,i+1})$, which is unknown by the attacker. By construction, since we want the affine encodings to be invertible, we know that $L_i$ is of rank 8. If one is able to recover Ker $L_i$, which is then a linear space over $\mathbb{F}_2^{16}$ of dimension $16-8 = 8$, then there exists an $8 \times 8$ invertible matrix $B_i$ such that $L_i = B_i \circ (0_8\ \mathsf{Id}_8) \circ V_i^{-1}$, where the linear mapping $V_i$ is built as $(v_1 \ldots v_{16})$ with $\{v_1, \ldots, v_8\}$ a basis of Ker $L_i$ and $\{v_9, \ldots, v_{16}\}$ a completion of this basis. In that case, while the matrices $B_i$ are still unknown for the attacker and will form the block diagonal matrix $\mathcal{B}$, one can build the matrix $\widetilde{A}$ from the $8 \times 16$ blocks $(0_8\ \mathsf{Id}_8) \circ V_i^{-1}$.

So now, we only need a way to compute Ker $L_i$ from the tables $T_i$, which can be done using the following lemma.

**Lemma 4.4.1.** *For any $(a, b) \in \mathbb{F}_2^8 \times \mathbb{F}_2^8$:*

1. $x \in$ *Ker* $A_{i,i} \quad \Rightarrow y \mapsto T_i(a \oplus x, b \oplus y) \oplus T_i(a, b \oplus y)$ *is constant,*
2. $y \in$ *Ker* $A_{i,i+1} \Rightarrow x \mapsto T_i(a \oplus x, b \oplus y) \oplus T_i(a \oplus x, y)$ *is constant,*
3. $(x, y) \in$ *Ker* $L_i \Rightarrow T_i(a, b) \oplus T_i(a \oplus x, b) \oplus T_i(a, b \oplus y) \oplus T_i(a \oplus x, b \oplus y) = 0$.

*Proof.* We will only prove the first and the last points, since the second one is very similar to the first. From the construction of $T_i$, we can write it as

$$T_i(x, y) = \widetilde{S_i}\left[A_{i,i}(x) \oplus A_{i,i+1}(y) \oplus c_i\right] \oplus h_i(x) \oplus h_{i+1}(y)$$

where $\widetilde{S_i} = M_i \circ S$.

1. Let us take $(a, b)$ a fixed element in $\mathbb{F}_2^8 \times \mathbb{F}_2^8$ and $x \in$ Ker $A_{i,i}$. Then for any $y \in \mathbb{F}_2^8$ we have

$$\begin{aligned}
&T_i(a \oplus x, b \oplus y) \oplus T_i(a, b \oplus y) \\
&= \widetilde{S_i}\left[A_{i,i}(a \oplus x) \oplus A_{i,i+1}(b \oplus y) \oplus c_i\right] \oplus h_i(a \oplus x) \oplus h_{i+1}(b \oplus y) \\
&\quad \oplus \widetilde{S_i}\left[A_{i,i}(a) \oplus A_{i,i+1}(b \oplus y) \oplus c_i\right] \oplus h_i(a) \oplus h_{i+1}(b \oplus y) \\
&= \widetilde{S_i}\left[A_{i,i}(a) \oplus A_{i,i+1}(b \oplus y) \oplus c_i\right] \oplus h_i(a \oplus x) \\
&\quad \oplus \widetilde{S_i}\left[A_{i,i}(a) \oplus A_{i,i+1}(b \oplus y) \oplus c_i\right] \oplus h_i(a) \\
&= h_i(a \oplus x) \oplus h_i(a),
\end{aligned}$$

which does not depend on $y$, therefore $y \mapsto T_i(a \oplus x, b \oplus y) \oplus T_i(a, b \oplus y)$ is constant.

3. First note that $(x, y) \in \text{Ker } L_i \Leftrightarrow L_i(x, y) = 0 \Leftrightarrow A_{i,i}(x) = A_{i,i,+1}(y)$.

So let take $(a, b)$ a fixed element in $\mathbb{F}_2^8 \times \mathbb{F}_2^8$ and $(x, y) \in \text{Ker } L_i$, then

$$
\begin{aligned}
& T_i(a,b) \oplus T_i(a \oplus x, b) \oplus T_i(a, b \oplus y) \oplus T_i(a \oplus x, b \oplus y) \\
&= \widetilde{S_i}\left[A_{i,i}(a) \oplus A_{i,i+1}(b) \oplus c_i\right] \oplus h_i(a) \oplus h_{i+1}(b) \\
&\quad \oplus \widetilde{S_i}\left[A_{i,i}(a \oplus x) \oplus A_{i,i+1}(b) \oplus c_i\right] \oplus h_i(a \oplus x) \oplus h_{i+1}(b) \\
&\quad \oplus \widetilde{S_i}\left[A_{i,i}(a) \oplus A_{i,i+1}(b \oplus y) \oplus c_i\right] \oplus h_i(a) \oplus h_{i+1}(b \oplus y) \\
&\quad \oplus \widetilde{S_i}\left[A_{i,i}(a \oplus x) \oplus A_{i,i+1}(b \oplus y) \oplus c_i\right] \oplus h_i(a \oplus x) \oplus h_{i+1}(b \oplus y) \\
&= \widetilde{S_i}\left[A_{i,i}(a) \oplus A_{i,i+1}(b) \oplus c_i\right] \\
&\quad \oplus \widetilde{S_i}\left[A_{i,i}(a) \oplus A_{i,i}(x) \oplus A_{i,i+1}(b) \oplus c_i\right] \\
&\quad \oplus \widetilde{S_i}\left[A_{i,i}(a) \oplus A_{i,i+1}(b) \oplus A_{i,i+1}(y) \oplus c_i\right] \\
&\quad \oplus \widetilde{S_i}\left[A_{i,i}(a) \oplus A_{i,i}(x) \oplus A_{i,i+1}(b) \oplus A_{i,i+1}(y) \oplus c_i\right] \\
&= \widetilde{S_i}\left[A_{i,i}(a) \oplus A_{i,i+1}(b) \oplus c_i\right] \\
&\quad \oplus \widetilde{S_i}\left[A_{i,i}(a) \oplus A_{i,i}(x) \oplus A_{i,i+1}(b) \oplus c_i\right] \\
&\quad \oplus \widetilde{S_i}\left[A_{i,i}(a) \oplus A_{i,i+1}(b) \oplus A_{i,i}(x) \oplus c_i\right] \\
&\quad \oplus \widetilde{S_i}\left[A_{i,i}(a) \oplus A_{i,i}(x) \oplus A_{i,i+1}(b) \oplus A_{i,i}(x) \oplus c_i\right] = 0. \qquad \square
\end{aligned}
$$

Note that the third point is a strict implication. Indeed, if one takes $x \in \text{Ker } A_{i,i}$, one can easily see that for any $y \in \mathbb{F}_2^8$, the third equation holds while $(x, y)$ is not necessarily in Ker $L_i$. So to compute Ker $L_i$, we first need to recover Ker $A_{i,i}$ and Ker $A_{i,i+1}$.

We can safely assume that if $x \notin \text{Ker } A_{i,i}$, the function $f_x : y \mapsto T_i(a \oplus x, b \oplus y) \oplus T_i(a, b \oplus y)$ behaves like a random function and then is constant with overwhelmingly low probability. Therefore, by choosing any $(a, b) \in \mathbb{F}_2^8 \times \mathbb{F}_2^8$, one can check if $x \in \text{Ker } A_{i,i}$ by computing $f_x$ and checking whether or not $f_x$ is constant. Obviously, the same method can be applied to recover Ker $A_{i,i+1}$.

Once Ker $A_{i,i}$ and Ker $A_{i,i+1}$ are recovered, one can recover the remaining elements $(x, y) \in$ Ker $L_i$ with $x \notin$ Ker $A_{i,i}$ and $y \notin$ Ker $A_{i,i+1}$ by using the third implication: if $(x, y) \notin$ Ker $L_i$, we can assume that the resulting value of the equation behaves like a random variable over $\mathbb{F}_2^8$ and is then equal to 0 with probability $2^{-8}$. Therefore, one can check if $(x, y) \in$ Ker $L_i$ by choosing *a few*[4] values for $(a, b)$ and checking if the equation stands for all these $(a, b)$.

In that way, we can recover Ker $L_i$ in roughly $\sim 2^{18}$ table lookups using the method described above and which is summarized in Algorithm 7. Since we need to repeat this operation 32 times, we end up with a complexity of $\sim 2^{23}$ table lookups to decompose $\mathcal{A}$ into $\mathcal{A} = \mathcal{B} \circ \widetilde{A}$.

### Building an Equivalent Representation of the Round Function

At this point, our encoded round function is $F = \mathcal{M} \circ (S, \ldots, S) \circ \mathcal{B} \circ \widetilde{A}$ where $\widetilde{A}$ is known and $\mathcal{B}$ is block diagonal, built with $8 \times 8$ affine mappings $\mathcal{B}_0, \ldots, \mathcal{B}_{31}$, but is still secret. Our goal is to find an equivalent representation of the round function, that is, finding affine mappings $\mathcal{M}'$ and $\mathcal{B}'$ which behave like $\mathcal{M}$ and $\mathcal{B}$ in the sense that $F = \mathcal{M}' \circ (S, \ldots, S) \circ \mathcal{B}' \circ \widetilde{A}$.

The idea is to find 32 affine mappings $\mathcal{B}'_i$ of size 8 to build $\mathcal{B}'$. Note that here, these $\mathcal{B}'_i$ will not necessarily be equal to $\mathcal{B}_i$, but we will see that we can then build $\mathcal{M}'$ in a way that solves this problem.

[4] In practice, 4 values are sufficient.

**Algorithm 7** Computing Ker $L_i$

*Compute Ker $A_{i,i}$ using implication 1*
$(a, b) \leftarrow$ random element in $\mathbb{F}_2^8 \times \mathbb{F}_2^8$
**for** $x \in \mathbb{F}_2^8$ **do**
  **if** $f_x$ is constant **then**
    Ker $A_{i,i} \leftarrow$ Ker $A_{i,i} \cup \{x\}$
  **end if**
**end for**

*Compute Ker $A_{i,i+1}$ using implication 2*
$(a, b) \leftarrow$ random element in $\mathbb{F}_2^8 \times \mathbb{F}_2^8$
**for** $y \in \mathbb{F}_2^8$ **do**
  **if** $f_y$ is constant **then**
    Ker $A_{i,i+1} \leftarrow$ Ker $A_{i,i+1} \cup \{y\}$
  **end if**
**end for**

*Compute the remaining elements of Ker $L_i$ using implication 3*
Ker $L_i \leftarrow (\mathbb{F}_2^8 \times \mathbb{F}_2^8) \backslash$(Ker $A_{i,i} \times$ Ker $A_{i,i+1}$)
**for** i = 1...4 **do**
  $(a, b) \leftarrow$ random element in $\mathbb{F}_2^8 \times \mathbb{F}_2^8$
  **for** $(x, y) \in$ Ker $L_i$ **do**
    **if** the equation does not holds **then**
      Ker $L_i \leftarrow$ Ker $L_i \backslash \{(x, y)\}$
    **end if**
  **end for**
**end for**
**return** Ker $L_i \cup$ (Ker $A_{i,i} \times$ Ker $A_{i,i+1}$)

Recall that we can evaluate the encoded round function $F$ by summing over the tables $T_j$. For $x_i \in \mathbb{F}_2^8$, let consider the function

$$\bigoplus_{j=0}^{31} T_j \circ \widetilde{A}^{-1}(0, \ldots, x_i, \ldots, 0).$$

Since $\mathcal{B}$ is block diagonal with blocks of size 8, only one S-box will be active, and so this function is a 8-bit to 256-bit mapping of the form $\mathcal{H} \circ S \circ \mathcal{B}_i$ where $\mathcal{H}$ is some affine function of size $8 \times 256$. Note that $\mathcal{H}$, $S$ and $\mathcal{B}_i$ are all injective (at least) by construction. So we can compute this function and deduce an affine projection $\mathcal{P}$ such that $\mathcal{P} \circ \mathcal{H} \circ S \circ \mathcal{B}_i$ is a bijection over $\mathbb{F}_2^8$. This bijection is then affine equivalent to the AES S-box, and we can use the affine equivalence algorithm from [22] to recover $\mathcal{B}_i$ in $\sim 2^{25}$.

However, there are some self-equivalence relations on the AES S-box, which means there exist some[5] affine mappings $\mathcal{A}_1, \mathcal{A}_2$ of size $8 \times 8$ such that $\mathcal{A}_2 \circ S \circ \mathcal{A}_1 = S$. Therefore, the affine equivalence algorithm will not exactly recover $\mathcal{B}_i$, but one $\mathcal{B}'_i = \mathcal{A}_1 \circ \mathcal{B}_i$ without knowing which $\mathcal{A}_1$ is used. In our present case where we only want to provide an equivalent representation of the round function, this does not really matter. We can choose any candidate for each $\mathcal{B}'_i$, and we will show how to build an affine mapping $\mathcal{M}'$ to compensate the action of $\mathcal{A}_1$.

[5]There are 2040 such pairs $(\mathcal{A}_1, \mathcal{A}_2)$, see [22].

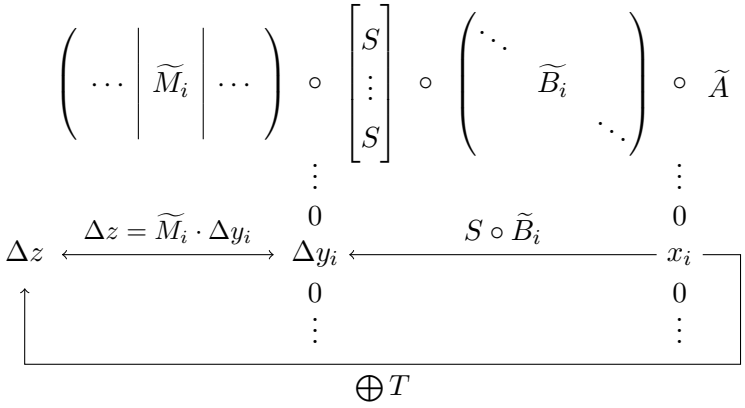


Figure 4.4: Building $\widetilde{M}_i$

So we are looking at our equivalent round-function $\mathcal{M}' \circ (S, \ldots, S) \circ \mathcal{B}' \circ \widetilde{A}$, where $\mathcal{B}'$ and $\widetilde{A}$ are known, but we still need to find $\mathcal{M}'$. The overall strategy for that is depicted in Fig. 4.4 and detailed below. As in the description of the scheme, let us split the linear part of $\mathcal{M}'$ into $(M'_0 \ldots M'_{31})$ where $M'_i$ is of size $256 \times 8$. Algorithm 2 gives the procedure to compute $M'_i$. The idea is just to compute the image each vector of the canonical basis through $M'$, which can be done using the fact that we fixed one candidate for each $\mathcal{B}'_i$.

**Algorithm 8** Computing $M'_i$

$x_i^0 \leftarrow$ random element in $\mathbb{F}_2^8$
$x^0 \leftarrow (0 \ldots x_i^0 \ldots 0) \in \mathbb{F}_2^{256}$
$z^0 \leftarrow \bigoplus T \circ \widetilde{A}^{-1}(x^0)$
$y_i^0 \leftarrow S(\mathcal{B}'_i(x_i^0))$ *since we know $\mathcal{B}'_i$*
**for each** $e_j = (0 \ldots 1 \ldots 0) \in \mathbb{F}_2^8$ **do** *with a 1 at the $j$-th position*
    $y_i^j \leftarrow y_i^0 \oplus e_j$
    $x_i^j \leftarrow \mathcal{B}'^{-1}_i(S^{-1}(y_i^j))$
    $x^j \leftarrow (0 \ldots x_i^j \ldots 0)$
    $z^j \leftarrow \bigoplus T \circ \widetilde{A}^{-1}(x^j)$
    $\Delta z^j \leftarrow z^0 \oplus z^j$
    $j$-th column of $M'_i \leftarrow \Delta z^j$ *since $\Delta y^j = (0 \ldots e_j \ldots 0)$*
**end for**

We can apply this method for all 32 blocks $\mathcal{B}'_i$ to recover the linear part of $\mathcal{M}'$. After that, to recover the affine translation $m'$ of $\mathcal{M}'$, we only need to compute

$$z' = M'.(S, \ldots, S) \circ \mathcal{B} \circ \widetilde{A}(x)$$

and $z = \bigoplus T_i(x)$ for one $x \in \mathbb{F}_2^{256}$, then we can easily recover $m'$ since in that case $z = z' \oplus m'$.

So we are able to provide a computationally easy to invert equivalent representation of the encoded round function as $\mathcal{M}' \circ (S, \ldots, S) \circ \mathcal{B}' \circ \widetilde{A}$. The complexity of building $\mathcal{M}'$ and $\mathcal{B}'$ is dominated by the 32 calls to the affine equivalence algorithm to get each $\mathcal{B}'_i$, which lead to a complexity of about $32 \times 2^{25} = 2^{30}$, which is therefore the complexity of this whole 1-round attack.

**Building an Equivalent Representation of the Scheme**

Therefore, we can already provide an attack on the full 10-round scheme: indeed, we just need to apply the above method on each encoded round function $F^{(r)}$. Note that the external encodings do not pose any problem here. For the external input encoding $\mathcal{M}_{in}$, recall that we know the affine mapping $F^{(0)} = \left(\mathcal{A}^{(0)}\right)^{-1} \circ \mathcal{M}_{in}$. Using the previous technique, we are able to recover an equivalent representation $\widetilde{F}^{(1)}$ of $F^{(1)}$, such that $\widetilde{F}^{(1)} = F^{(1)}$ while $\widetilde{F}^{(1)}$ is easy to invert. So since we then have $\widetilde{F}^{(1)} \circ F^{(0)} = F^{(1)} \circ F^{(0)}$, we do not need to do anything about $\mathcal{M}_{in}$ to provide an equivalent representation of the scheme.

For the external output encoding $\mathcal{M}_{out}$, recall that the last encoded round function $F^{(10)}$ is defined by

$$F^{(10)} = \mathcal{M}_{out} \circ \left(\mathsf{AES}^{(10)}, \mathsf{AES}^{(10)}\right) \circ \mathcal{A}^{(10)} = \mathcal{M}^{(10)} \circ (S, \ldots, S) \circ \mathcal{A}^{(10)}$$

where $\mathcal{M}^{(10)} = \mathcal{M}_{out} \circ \oplus_{K^{11}} \circ \mathsf{SR}$. Then our technique applied on $F^{(10)}$ gives us 3 affine mappings $\mathcal{M}'^{(10)}, \mathcal{B}'^{(10)}$ and $\mathcal{A}'^{(10)}$ such that

$$F^{(10)} = \widetilde{F}^{(10)} = \mathcal{M}'^{(10)} \circ (S, \ldots, S) \circ \mathcal{B}'^{(10)} \circ \mathcal{A}'^{(10)}$$

while $\widetilde{F}^{(10)}$ is easy to invert, so again, $\mathcal{M}_{out}$ does not pose any problem here.

All in all, we have built 10 easy to invert equivalent round-functions $\widetilde{F}^{(r)}$ such that

$$\widetilde{F}^{(10)} \circ \cdots \circ \widetilde{F}^{(1)} \circ F^{(0)} = F^{(10)} \circ \cdots \circ F^{(1)} \circ F^{(0)} = \mathcal{M}_{out} \circ (\mathsf{AES}, \mathsf{AES}) \circ \mathcal{M}_{in},$$

which is the original scheme. The cost for doing this is to repeat 10 times the 1-round attack, which gives us a complexity of $10 \times 2^{30}$. While this is already practical, we only have an equivalent representation of the scheme, but we did not recover the key nor the encodings.

### 4.4.2 Recovering the Key

While we could just use the previous method 10 times on each encoded round function to provide an easy to invert representation of the full scheme, we can do better and fully break the scheme by recovering the key in a more efficient way by exploiting two consecutive rounds.

So let us start at the point where we decomposed one round into $F = \mathcal{M} \circ (S, \ldots, S) \circ \mathcal{B} \circ \widetilde{A}$, with $\widetilde{A}$ known and $\mathcal{B}$ an affine diagonal mapping. Recall that using the affine equivalence algorithm from [22] for each block does not give us exactly $\mathcal{B}_i$, but roughly $2^{11}$ candidates $\mathcal{B}'_i$. If we want to recover exactly the key and the encodings, we need to identify which candidate is exactly $B_i$. Note that since we have $2^{11}$ candidates for each of the 32 $\mathcal{B}_i$, we cannot exhaust them all.

To be able to quickly identify the correct candidate, one can first apply the previous method on two consecutive rounds. By doing so, we decompose these two rounds into

$$F^{(r+1)} = \mathcal{M}^{(r+1)} \circ (S, \ldots, S) \circ \mathcal{B} \circ \widetilde{A}$$
$$F^{(r)} = \mathcal{M}^{(r)} \circ (S, \ldots, S) \circ \mathcal{C} \circ \widehat{A}$$

where $\widetilde{A}, \widehat{A}$ are known and $\mathcal{B}, \mathcal{C}$ are affine block diagonal mappings, which are still secret, but for which we know $2^{11}$ candidates for each block $\mathcal{B}_i$ and $\mathcal{C}_i$.

In that case, we can write $F^{(r)}$ as

$$\widetilde{A}^{-1} \circ \mathcal{B}^{-1} \circ (\mathsf{MC} \circ \mathsf{SR}, \mathsf{MC} \circ \mathsf{SR}) \circ (S, \ldots, S) \circ \mathcal{C} \circ \widehat{A}.$$

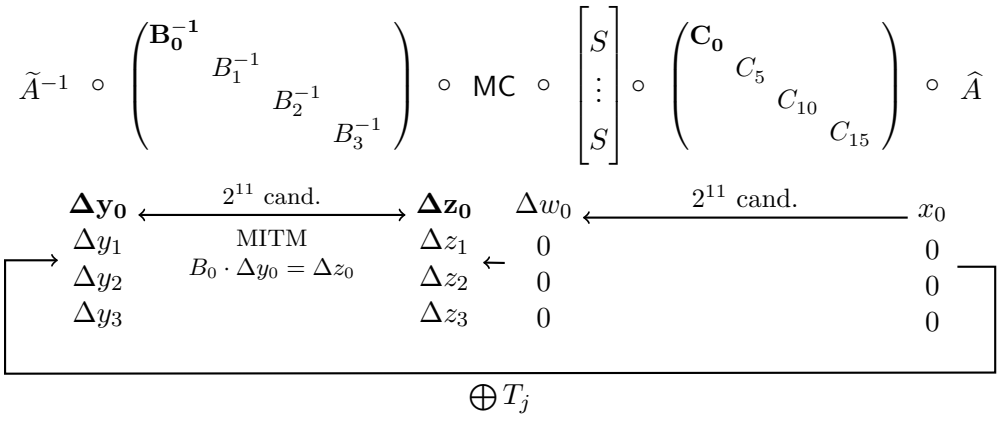


Figure 4.5: Identifying correct blocks.

Since $\mathcal{B}$ is block diagonal, so is its inverse, and we know $\widetilde{A}$ and $\widehat{A}$. Our problem in then reduced to block diagonal input and output encodings for one encoded round function, with $2^{11}$ candidates for each block $\mathcal{B}_i$ and $\mathcal{C}_i$. We now need to recover which are the correct $\mathcal{B}_i$ and $\mathcal{C}_i$. To do so, we will use a Meet-in-the-Middle approach depicted in Fig.4.5 and detailed below.

The MixColumns operation of AES works on words of four bytes, so we restrict our work on four $B_i$ and the corresponding four $C_i$ that will be used as input of the same MixColumns operation. For example, as depicted in Fig.4.5, we can first consider $B_0, B_1, B_2, B_3$ and $C_0, C_5, C_{10}, C_{15}$ which will be on the same MixColumns operation after the application of ShiftRows. For an easier understanding, we will describe our MITM method using these blocks, as it will be exactly the same for the other $B_i$ and $C_i$. The detailed procedure is given in Algorithm 9.

We want to use the MITM to identify the correct $(\mathcal{B}_0, \mathcal{C}_0)$, for which we have $2^{22}$ candidates in total. As we will search a match with $\Delta z_0^1, \ldots, \Delta z_0^m$ where each $\Delta z_i^j$ is an 8-bit value, taking $m = 4$ leads to a 32-bit filter, which is enough to leave only the right candidates. Building the hash table costs $\sim 2^{11}$, and so does the matching step. Once $\mathcal{B}_0$ and $\mathcal{C}_0$ are recovered, we only need to go through all the candidates for the remaining $\mathcal{B}_i$ and $\mathcal{C}_i$, which is done separately. Since we have $2^{11}$ candidates for each of them, the total cost of this step is roughly $8 \times 2^{11} = 2^{14}$. Finally, we need to do this method on each of the 8 groups of 4 $\mathcal{B}_i$ and 4 $\mathcal{C}_i$, leading to a complexity of $\sim 2^{17}$ to recover $\mathcal{B}$ and $\mathcal{C}$.

### Extracting the Key

Note that the reason why we used the differences $\Delta z_i$ instead of the values are because when we decomposed $F^{(r+1)}$ into $\mathcal{M}^{(r+1)} \circ (S, \ldots, S) \circ \mathcal{B} \circ \widetilde{A}$, $\mathcal{B}$ contains the key in its affine translation: that is, we have $\mathcal{B}(x) = B.x \oplus (b \oplus K^{(r+1)})$ such that $B.\widetilde{A}.x \oplus b = \mathcal{A}^{(r+1)}$. The same phenomenon happens with $\mathcal{C}$, which we recover as $\mathcal{C}(x) = C.x \oplus (c \oplus K^{(r)})$. So when we need to use $\mathcal{B}_i$ for the MITM, the affine translation will not be the *good* one, while the linear part is. However, once we recovered the correct $\mathcal{B}$ and $\mathcal{C}$, we can use this fact to recover the key $K^{(r+1)}$, and thus also recovering exactly the affine translation of $\mathcal{B}$ and $\mathcal{C}$. Indeed, denote $z = (\mathsf{MC} \circ \mathsf{SR}, \mathsf{MC} \circ \mathsf{SR}) \circ (S, \ldots, S) \circ \mathcal{C} \circ \widehat{A}(x)$ and $y = \widetilde{A} \circ F^{(r)}(x)$ where again, $F^{(r)}$ is computed using the tables. Then we know that $B.z \oplus b = y$, and since we know $B, y$ and $z$, we can easily compute $b$. Finally, since we previously recovered $\mathcal{B}(x) = B.x \oplus (b \oplus K^{(r+1)})$, we get $K^{(r+1)}$.

Since the key schedule of AES is invertible, one can do this procedure on the first two rounds, given through the tables $T^{(1)}$ and $T^{(2)}$. That way we can compute $K^{(1)}$, which is the master key

**Algorithm 9** Identifying correct blocks

$x^0, \ldots, x^m \leftarrow$ messages with byte $x_0^j$ taking different values and $x_i^j = 0$ if $i \neq 0$
**for each** candidate for $\mathcal{C}_0$ **do**
    $\Delta w_0^j \leftarrow S(\mathcal{C}_0(x_0^0)) \oplus S(\mathcal{C}_0(x_0^j))$  *$x_i^j$ is constant if $i \neq 0$, so $\Delta w_i^j = 0$ if $i \neq 0$*
    $(\Delta z_0^j, \Delta z_1^j, \Delta z_2^j, \Delta z_3^j) \leftarrow \mathsf{MC}.(\Delta w_0^j, 0, 0, 0)$
    Store $\mathcal{C}_0$ in a hash table $\mathcal{T}_z$ indexed by $\Delta z_0^1, \ldots, \Delta z_0^m$
**end for**
$y^j \leftarrow \widetilde{A} \circ F^{(r)} \circ \widehat{A}^{-1}(x^j)$  *$F^{(r)}$ can be evaluated using the tables $T_j$*
$\Delta y_0^j \leftarrow y_0^0 \oplus y_0^j$
**for each** candidate for $\mathcal{B}_0$ **do**
    $\Delta \tilde{z}_0^j \leftarrow B_0.\Delta y_0^j$
    **if** $\Delta \tilde{z}_0^1, \ldots, \Delta \tilde{z}_0^m \in \mathcal{T}_z$ **then**
        We have the correct $\mathcal{B}_0$ and $\mathcal{C}_0 = \mathcal{T}_z[\Delta \tilde{z}_0^1, \ldots, \Delta \tilde{z}_0^m]$
        **break**
    **end if**
**end for**
*Once we have the correct $\mathcal{C}_0$, we know the correct values of $\Delta z_i^j$, so we do not need any hash table*
**for each** candidate for $\mathcal{B}_i, i = 1, 2, 3$ **do**
    **if** $B_i.\Delta y_i^j = \Delta z_i^j$ **then**
        We have the correct $\mathcal{B}_i$
    **end if**
**end for**
*Once we have all the correct $\mathcal{B}_0, \mathcal{B}_1, \mathcal{B}_2, \mathcal{B}_3$, we can use the same kind of computation to identify the correct remaining $\mathcal{C}_i$ using messages with $x_i^j$ taking different values and $x_l^j$ constant for $l \neq i$*

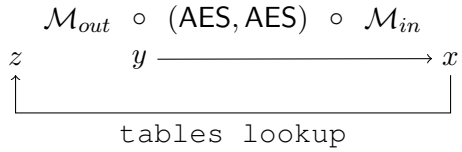


Figure 4.6: Recovering $M_{out}$

used in AES, from $K^{(2)}$. This only leaves the external encodings to be recovered, which is an easy task now. We can recover exactly which affine translation was used for $\mathcal{C}$ for which we knew $\mathcal{C}(x) = C.x \oplus (c \oplus K^{(1)})$. Then we can recover the first input encoding $\mathcal{A}^{(1)}$ as $\mathcal{A}^{(1)}(x) = C.\widehat{A}.x \oplus c$. Now recall that the external input encoding we knew was $F^{(0)} = \left(\mathcal{A}^{(1)}\right)^{-1} \circ \mathcal{M}_{in}$. We recovered $\mathcal{A}^{(1)}$ so it is easy to compute $\mathcal{M}_{in}$.

Recovering $\mathcal{M}_{out}$ is not hard either, see Fig. 4.6. We know the key, meaning we can easily compute the two parallel AES, and we also know $\mathcal{M}_{in}$. So for any $y \in \mathbb{F}_2^{256}$, one can compute $x$ such that $y = (\mathsf{AES}, \mathsf{AES}) \circ \mathcal{M}_{in}(x)$, then $z = \mathcal{M}_{out}(y)$ from $x$ by using the tables. Therefore, we only need to do this for 257 values of $y$: the zero vector to get the affine translation of $\mathcal{M}_{out}$ and then each of the 256 canonical basis vectors.

All in all, we recovered the key of the AES as well as both external encodings. The cost of

| | |
|---|---|
| Diagonal decomposition | $\sim n \cdot 2^{22}$ |
| Inverting one encoding | $\sim n^3 \cdot 2^{21}$ |
| Affine equivalence | $\sim n \cdot 2^{30}$ |
| MITM | $\sim n \cdot 2^{16}$ |
| Implementation size | $n^2 \cdot 160$ MB |

Figure 4.7: Complexity of our attack and implementation size for $n$ parallel AES instances.

doing this is dominated by the cost of the 64 calls to the affine equivalence algorithm to get some candidates for $\mathcal{B}_i$ and $\mathcal{C}_i$, which leads to complexity of $\sim 2^{31}$.

**Implementation.** We implemented the attack in `C++`, relying on NTL [123] for linear algebra. The total time to recover both the key and the external encodings is about 12 seconds, with roughly 10 seconds spent on the 64 affine equivalences, and using a negligible amount of memory. This was run on a `Intel Core i7-6600U CPU @ 2.60GHz` on a single core. Our implementation is available at `http://wbcheon.gforge.inria.fr/`.

### 4.4.3 Using More AES Instances in Parallel

A natural question is whether the white-box scheme by Baek et al. could be made secure by increasing the number $n$ of AES instances encoded in parallel. However in this section, we show that this is not the case, as the storage requirement of storing the actual white-box implementation quickly becomes limiting.

More precisely, Table 4.7 shows the complexity of each step of our dedicated attack as $n$ increases, together with the size of the corresponding white-box implementation. Recall that in this section, $n$ denotes the number of parallel AES instances (rather than the total block size).

So the dominating cost comes from either the computation of the inverse of one encoding or the calls to the affine equivalence algorithm. For $n \leq 22$, the affine equivalence is dominating and lead to a complexity of $\sim 2^{35}$ for an implementation of size $\sim 64$ GB when $n = 22$. Otherwise the inversion is dominating, and obtaining even a 60-bit security would need $n = 2^{13}$ parallel AES, which lead to an implementation of size $\sim 2^{13}$ TB, which is definitely not realistic.

## 4.5 Conclusion

In this chapter, we propose a generic algorithm to recover affine encodings for SPN ciphers, in the context of white-box schemes following the framework of Chow et al. More generally, our algorithm solves the affine equivalence problem in the special case where one of the two maps is composed of the parallel application of distinct S-boxes. We illustrate the efficiency of our attack on a white-box implementation of AES with external encodings proposed by Baek, Cheon and Hong, which was precisely designed to make a generic ASA approach out of computational reach. Nevertheless our generic attack breaks the scheme in $2^{35}$ basic operations, compared to the assessment by its authors that $2^{110}$ would be required. We then took a closer look at the Baek et al. scheme, and identified another attack vector, which reduces the attack to a simple standalone problem. This

second approach results recovers the secret key in time complexity $2^{31}$. A full implementation of the attack confirms the complexity estimate.

It may be fair to suggest that a secure white-box implementation in anything resembling the CEJO framework is implausible, considering every attempt to date has been closely followed by a devastating attack. In a nutshell, in this work we showed that obfuscating the round function of an SPN cipher (such as AES) using affine encodings is essentially impossible. In this light, our result suggests that non-linear encodings should play a central role in any future endeavor in this direction.

*"We're in the endgame now."*

— Dr. Strange

# Conclusion

In this thesis, we saw several ways of studying and optimizing different components of block ciphers algorithms, as well as some new cryptanalysis results. We often focused on a specific criterion to optimize, and used different tools such as Constraint Programming, meta-heuristics, Mixed Integer Linear Programming, theoretical arguments or just plain coding. Our results gave a better idea of how to design some block ciphers components as well as different strategies to be considered. Still, some open problems remains.

In the first chapter, we study the choice of the permutation used to build a Generalized Feistel Network, focusing on finding the best permutations w.r.t the diffusion round. We gave a new characterization for a permutation to reach full diffusion, leading us to an efficient algorithm to search for optimal permutations. As a result, we were able to give new results for up to 42 blocks, either by giving optimal permutations or better lower bounds on the diffusion round. Specifically, we solved a 10-year-old problem, first given in [131], by finding all optimal permutations for a 32-block GFN. While our algorithm should be scalable to a few more blocks, it seems to be difficult go up to 64 blocks and beyond. We thus leave open the problem of finding optimal permutations for even more blocks than what we exhibited, even though the application of such a large GFN might be thin. Another interesting way of research is that our optimal permutations (w.r.t diffusion round) have slightly worse security results when considering differentials attacks, compared to the non-optimal ones given in [39]. As such, it would be interesting to try to find the best possible trade-off between a given set of criteria, which is a work that will be pursued in the future.

In Chapter 2, we gave some new insight about division property cryptanalysis. We showed that the representation of a block cipher is crucial when trying to find a division property based distinguisher, as some representation can lead to better distinguishers. We described an efficient algorithm to take care of a specific case of such different representation, namely when considering a linearly equivalent block cipher $L_{out} \circ E \circ L_{in}$ when $L_{out}$ and $L_{in}$ are block diagonal linear mappings. As a result, we were able to find a new distinguisher for RECTANGLE over one more rounds than previously known. We also gave a new criterion to build S-boxes so that they have better resistance against division property. Specifically, we proved that a certain class of S-boxes are optimal w.r.t to division property, and proposed new S-boxes for RECTANGLE and PRESENT to improve their resistance by 2 rounds, even when considering our previous extension technique. A natural limitation to our current algorithm is that it is costly to try to find distinguisher with a reduced data complexity. The same can be said if we want to study a block cipher using 8-bit S-boxes, not only because our algorithm needs more calls to the MILP solver, but also because the resulting MILP models are way more costly to solve. Another improvement that could be made would be to try considering a larger class of block cipher. Indeed, we focused on SPN block-ciphers, but some idea should be reusable for Feistel for example.

Then in Chapter 3, we tried to find a better key-schedule algorithm for AES. Specifically, we looked to replace the AES key-schedule with a permutation, so that we have a higher minimal number of active S-boxes. Along with giving some theoretical bounds, we used a combination of Constraint Programming and meta-heuristics to exhibit permutations which are close to the optimal goal for AES, namely, reaching 22 active S-boxes over 6 rounds. Even if our permutations do not reach 22 active S-boxes, all of them still have the property that no differential characteristic with probability higher than $2^{-128}$ exists. An obvious open question is whether we can actually reach 22 active S-boxes over 6 rounds. In the same idea, we proved that 5 rounds cannot lead to 18 or more

S-boxes, and we found a permutation reaching 16 active S-boxes, thus the 17 S-boxes case is still open. Essentially, a more general question is whether we could build an AES-like block cipher such that, using a permutation a key-schedule, we have at least 22 active S-boxes over 6 rounds.

Finally in the last chapter, we gave new improvements for the cryptanalysis of some white-box schemes. We described a generic algorithm to efficiently solve a special case of the affine equivalent problem, namely, given two affine equivalent functions $F$ and $S$ such that $S$ is the concatenation of several S-boxes, find affine mappings $\mathcal{A}$ and $\mathcal{B}$ such that $F = \mathcal{B} \circ S \circ \mathcal{A}$. This is the core problem of white-box schemes derived from the CEJO framework, and solving efficiently this problem essentially allows us to create the decryption function when given the white-box implementation of the encryption function, thus completely breaking the security of the scheme. Our algorithm scales very well, and when considering AES parameters for example, has a complexity of about $2^{32}$ operations. We then focused on a dedicated cryptanalysis of a white-box scheme from Baek et al.. While our generic algorithm applies here, we were able to go further and do a key-recovery attack of complexity $2^{31}$, while our previous generic algorithm would need about $2^{35}$ operations to find the decryption function. Thus, for any white-box scheme based on the CEJO framework, the so-called "affine encodings" only bring a negligible security factor, and thus cannot provide a secure white-box implementation. Using non-linear encodings would be the only way, but considering the known attacks, it would quickly lead to an implementation of unreasonable size even for a decent level of security. As such, we need to find a totally new framework to build white-box schemes, as the CEJO framework seems to have reached its limits.

# Bibliography


[1] Abdelkhalek, A., Sasaki, Y., Todo, Y., Tolba, M., Youssef, A.M.: MILP Modeling for (Large) S-boxes to Optimize Probability of Differential Characteristics. IACR Trans. Symmetric Cryptol. **2017**(4), 99–129 (2017)

[2] Aoki, K., Ichikawa, T., Kanda, M., Matsui, M., Moriai, S., Nakajima, J., Tokita, T.: Camellia: A 128-Bit Block Cipher Suitable for Multiple Platforms - Design and Analysis. In: Selected Areas in Cryptography, 7th Annual International Workshop, SAC 2000, Waterloo, Ontario, Canada, August 14-15, 2000, Proceedings. pp. 39–56 (2000). doi: 10.1007/3-540-44983-3_4, https://doi.org/10.1007/3-540-44983-3_4

[3] Baek, C.H., Cheon, J.H., Hong, H.: White-box AES implementation revisited. Journal of Communications and Networks **18**(3), 273–287 (2016). doi: 10.1109/JCN.2016.000043, http://dx.doi.org/10.1109/JCN.2016.000043

[4] Banik, S., Bogdanov, A., Isobe, T., Jepsen, M.: Analysis of software countermeasures for whitebox encryption. IACR Transactions on Symmetric Cryptology (ToSC) **2017**(1), 307–328 (2017)

[5] Banik, S., Bogdanov, A., Isobe, T., Shibutani, K., Hiwatari, H., Akishita, T., Regazzoni, F.: Midori: A block cipher for low energy. In: Advances in Cryptology - ASIACRYPT 2015 - 21st International Conference on the Theory and Application of Cryptology and Information Security, Auckland, New Zealand, November 29 - December 3, 2015, Proceedings, Part II. pp. 411–436 (2015). doi: 10.1007/978-3-662-48800-3_17, https://doi.org/10.1007/978-3-662-48800-3_17

[6] Bar-On, A.: Improved higher-order differential attacks on MISTY1. In: Fast Software Encryption - 22nd International Workshop, FSE 2015, Istanbul, Turkey, March 8-11, 2015, Revised Selected Papers. pp. 28–47 (2015). doi: 10.1007/978-3-662-48116-5_2, https://doi.org/10.1007/978-3-662-48116-5_2

[7] Barak, B., Goldreich, O., Impagliazzo, R., Rudich, S., Sahai, A., Vadhan, S.P., Yang, K.: On the (im)possibility of obfuscating programs. In: Advances in Cryptology - CRYPTO 2001, 21st Annual International Cryptology Conference, Santa Barbara, California, USA, August 19-23, 2001, Proceedings. pp. 1–18 (2001). doi: 10.1007/3-540-44647-8_1, http://dx.doi.org/10.1007/3-540-44647-8_1

[8] Beaulieu, R., Shors, D., Smith, J., Treatman-Clark, S., Weeks, B., Wingers, L.: The SIMON and SPECK Families of Lightweight Block Ciphers. IACR Cryptology ePrint Archive **2013**, 404 (2013), http://eprint.iacr.org/2013/404

[9] Beaulieu, R., Shors, D., Smith, J., Treatman-Clark, S., Weeks, B., Wingers, L.: The SIMON and SPECK lightweight block ciphers. In: Proceedings of the 52nd Annual Design Automation Conference, San Francisco, CA, USA, June 7-11, 2015. pp. 175:1–175:6 (2015). doi: 10.1145/2744769.2747946, https://doi.org/10.1145/2744769.2747946

[10] Beierle, C., Jean, J., Kölbl, S., Leander, G., Moradi, A., Peyrin, T., Sasaki, Y., Sasdrich, P., Sim, S.M.: The SKINNY Family of Block Ciphers and Its Low-Latency Variant MANTIS. In: Advances in Cryptology - CRYPTO 2016 - 36th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 14-18, 2016, Proceedings, Part II. pp. 123–153 (2016)

[11] Bellare, M., Kohno, T.: A theoretical treatment of related-key attacks: Rka-prps, rka-prfs, and applications. In: Advances in Cryptology - EUROCRYPT 2003, International Conference on the Theory and Applications of Cryptographic Techniques, Warsaw, Poland, May 4-8, 2003, Proceedings. pp. 491–506 (2003). doi: 10.1007/3-540-39200-9_31, https://doi.org/10.1007/3-540-39200-9_31

[12] Biham, E.: New types of cryptanalytic attacks using related keys (extended abstract). In: Advances in Cryptology - EUROCRYPT '93, Workshop on the Theory and Application of of Cryptographic Techniques, Lofthus, Norway, May 23-27, 1993, Proceedings. pp. 398–409 (1993). doi: 10.1007/3-540-48285-7_34, https://doi.org/10.1007/3-540-48285-7_34

[13] Biham, E., Biryukov, A., Dunkelman, O., Richardson, E., Shamir, A.: Initial observations on skipjack: Cryptanalysis of skipjack-3xor. In: Selected Areas in Cryptography '98, SAC'98, Kingston, Ontario, Canada, August 17-18, 1998, Proceedings. pp. 362–376 (1998). doi: 10.1007/3-540-48892-8_27, https://doi.org/10.1007/3-540-48892-8_27

[14] Biham, E., Biryukov, A., Shamir, A.: Cryptanalysis of skipjack reduced to 31 rounds using impossible differentials. In: Advances in Cryptology - EUROCRYPT '99, International Conference on the Theory and Application of Cryptographic Techniques, Prague, Czech Republic, May 2-6, 1999, Proceeding. pp. 12–23 (1999). doi: 10.1007/3-540-48910-X_2, https://doi.org/10.1007/3-540-48910-X_2

[15] Biham, E., Biryukov, A., Shamir, A.: Cryptanalysis of Skipjack Reduced to 31 Rounds Using Impossible Differentials. In: Advances in Cryptology - EUROCRYPT '99, International Conference on the Theory and Application of Cryptographic Techniques, Prague, Czech Republic, May 2-6, 1999, Proceeding. pp. 12–23 (1999). doi: 10.1007/3-540-48910-X_2, https://doi.org/10.1007/3-540-48910-X_2

[16] Biham, E., Biryukov, A., Shamir, A.: Miss in the middle attacks on IDEA and khufu. In: Fast Software Encryption, 6th International Workshop, FSE '99, Rome, Italy, March 24-26, 1999, Proceedings. pp. 124–138 (1999). doi: 10.1007/3-540-48519-8_10, https://doi.org/10.1007/3-540-48519-8_10

[17] Biham, E., Shamir, A.: Differential cryptanalysis of DES-like cryptosystems. In: Advances in Cryptology - CRYPTO '90, 10th Annual International Cryptology Conference, Santa Barbara, California, USA, August 11-15, 1990, Proceedings. pp. 2–21 (1990). doi: 10.1007/3-540-38424-3_1, https://doi.org/10.1007/3-540-38424-3_1

[18] Biham, E., Shamir, A.: Differential cryptanalysis of feal and n-hash. In: Advances in Cryptology - EUROCRYPT '91, Workshop on the Theory and Application of of Cryptographic Techniques, Brighton, UK, April 8-11, 1991, Proceedings. pp. 1–16 (1991). doi: 10.1007/3-540-46416-6_1, https://doi.org/10.1007/3-540-46416-6_1

[19] Biham, E., Shamir, A.: Differential cryptanalysis of the full 16-round DES. In: Advances in Cryptology - CRYPTO '92, 12th Annual International Cryptology Conference,

Santa Barbara, California, USA, August 16-20, 1992, Proceedings. pp. 487–496 (1992). doi: 10.1007/3-540-48071-4_34, https://doi.org/10.1007/3-540-48071-4_34

[20] Billet, O., Gilbert, H., Ech-Chatbi, C.: Cryptanalysis of a white box AES implementation. In: International Workshop on Selected Areas in Cryptography. pp. 227–240. Springer (2004)

[21] Biryukov, A., Bouillaguet, C., Khovratovich, D.: Cryptographic schemes based on the ASASA structure: Black-box, white-box, and public-key. In: International Conference on the Theory and Application of Cryptology and Information Security (ASIACRYPT). pp. 63–84. Springer (2014)

[22] Biryukov, A., Cannière, C.D., Braeken, A., Preneel, B.: A toolbox for cryptanalysis: Linear and affine equivalence algorithms. In: Advances in Cryptology - EUROCRYPT 2003, International Conference on the Theory and Applications of Cryptographic Techniques, Warsaw, Poland, May 4-8, 2003, Proceedings. pp. 33–50 (2003). doi: 10.1007/3-540-39200-9_3, http://dx.doi.org/10.1007/3-540-39200-9_3

[23] Biryukov, A., Gong, G., Stinson, D.R. (eds.): Selected Areas in Cryptography - 17th International Workshop, SAC 2010, Waterloo, Ontario, Canada, August 12-13, 2010, Revised Selected Papers, Lecture Notes in Computer Science, vol. 6544. Springer (2011). doi: 10.1007/978-3-642-19574-7, https://doi.org/10.1007/978-3-642-19574-7

[24] Biryukov, A., Khovratovich, D.: Related-key cryptanalysis of the full AES-192 and AES-256. In: Matsui, M. (ed.) Advances in Cryptology - ASIACRYPT 2009, 15th International Conference on the Theory and Application of Cryptology and Information Security, Tokyo, Japan, December 6-10, 2009. Proceedings. Lecture Notes in Computer Science, vol. 5912, pp. 1–18. Springer (2009). doi: 10.1007/978-3-642-10366-7_1, https://doi.org/10.1007/978-3-642-10366-7_1

[25] Biryukov, A., Khovratovich, D.: Decomposition attack on SASASASAS. Cryptology ePrint Archive, Report 2015/646 (2015), https://eprint.iacr.org/2015/646

[26] Biryukov, A., Khovratovich, D., Nikolic, I.: Distinguisher and related-key attack on the full AES-256. In: Halevi, S. (ed.) Advances in Cryptology - CRYPTO 2009, 29th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 16-20, 2009. Proceedings. Lecture Notes in Computer Science, vol. 5677, pp. 231–249. Springer (2009). doi: 10.1007/978-3-642-03356-8_14, https://doi.org/10.1007/978-3-642-03356-8_14

[27] Biryukov, A., Leurent, G., Roy, A.: Cryptanalysis of the "kindle" cipher. In: Selected Areas in Cryptography, 19th International Conference, SAC 2012, Windsor, ON, Canada, August 15-16, 2012, Revised Selected Papers. pp. 86–103 (2012). doi: 10.1007/978-3-642-35999-6_7, https://doi.org/10.1007/978-3-642-35999-6_7

[28] Biryukov, A., Nikolic, I.: Automatic search for related-key differential characteristics in byte-oriented block ciphers: Application to aes, camellia, khazad and others. In: Gilbert, H. (ed.) Advances in Cryptology - EUROCRYPT 2010, 29th Annual International Conference on the Theory and Applications of Cryptographic Techniques, French Riviera, May 30 - June 3, 2010. Proceedings. Lecture Notes in Computer Science, vol. 6110, pp. 322–344. Springer (2010). doi: 10.1007/978-3-642-13190-5_17, https://doi.org/10.1007/978-3-642-13190-5_17

[29] Biryukov, A., Shamir, A.: Structural cryptanalysis of SASAS. Advances in Cryptology – EUROCRYPT 2001 pp. 395–405 (2001)

[30] Blondeau, C., Gérard, B.: Differential cryptanalysis of puffin and puffin2. In: ECRYPT Workshop on Lightweight Cryptography. p. 1. Citeseer (2011)

[31] Bock, E.A., Brzuska, C., Michiels, W., Treff, A.: On the ineffectiveness of internal encodings-revisiting the DCA attack on white-box cryptography. In: International Conference on Applied Cryptography and Network Security (ACNS). pp. 103–120. Springer (2018)

[32] Bogdanov, A.: On unbalanced feistel networks with contracting MDS diffusion. Des. Codes Cryptogr. **59**(1-3), 35–58 (2011). doi: 10.1007/s10623-010-9462-0, https://doi.org/10.1007/s10623-010-9462-0

[33] Bogdanov, A., Knudsen, L.R., Leander, G., Paar, C., Poschmann, A., Robshaw, M.J.B., Seurin, Y., Vikkelsoe, C.: PRESENT: An Ultra-Lightweight Block Cipher. In: Cryptographic Hardware and Embedded Systems - CHES 2007, 9th International Workshop, Vienna, Austria, September 10-13, 2007, Proceedings. pp. 450–466 (2007)

[34] Bogdanov, A., Rijmen, V.: Linear hulls with correlation zero and linear cryptanalysis of block ciphers. Des. Codes Cryptogr. **70**(3), 369–383 (2014). doi: 10.1007/s10623-012-9697-z, https://doi.org/10.1007/s10623-012-9697-z

[35] Bos, J.W., Hubain, C., Michiels, W., Teuwen, P.: Differential computation analysis: Hiding your white-box designs is not enough. In: International Conference on Cryptographic Hardware and Embedded Systems (CHES). pp. 215–236. Springer (2016)

[36] Boura, C., Canteaut, A.: On the influence of the algebraic degree of $f^{-1}$ on the algebraic degree of G ∘ F. IEEE Trans. Information Theory **59**(1), 691–702 (2013). doi: 10.1109/TIT.2012.2214203, https://doi.org/10.1109/TIT.2012.2214203

[37] Boura, C., Canteaut, A.: Another View of the Division Property. In: Advances in Cryptology - CRYPTO 2016 - 36th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 14-18, 2016, Proceedings, Part I. pp. 654–682 (2016)

[38] Carlet, C., Charpin, P., Zinoviev, V.A.: Codes, bent functions and permutations suitable for des-like cryptosystems. Des. Codes Cryptogr. **15**(2), 125–156 (1998). doi: 10.1023/A:1008344232130, https://doi.org/10.1023/A:1008344232130

[39] Cauchois, V., Gomez, C., Thomas, G.: General Diffusion Analysis: How to Find Optimal Permutations for Generalized Type-II Feistel Schemes. IACR Trans. Symmetric Cryptol. **2019**(1) (2019)

[40] Černỳ, V.: Thermodynamical approach to the traveling salesman problem: An efficient simulation algorithm. Journal of optimization theory and applications **45**(1), 41–51 (1985)

[41] Chow, S., Eisen, P., Johnson, H., Van Oorschot, P.C.: A white-box DES implementation for DRM applications. In: ACM Workshop on Digital Rights Management. pp. 1–15. Springer (2002)

[42] Chow, S., Eisen, P., Johnson, H., Van Oorschot, P.C.: White-box cryptography and an AES implementation. In: International Workshop on Selected Areas in Cryptography. pp. 250–270. Springer (2002)

[43] Choy, J., Zhang, A., Khoo, K., Henricksen, M., Poschmann, A.: AES variants secure against related-key differential and boomerang attacks. In: Ardagna, C.A., Zhou, J. (eds.) Information Security Theory and Practice. Security and Privacy of Mobile Devices in Wireless Communication - 5th IFIP WG 11.2 International Workshop, WISTP 2011, Heraklion, Crete, Greece, June 1-3, 2011. Proceedings. Lecture Notes in Computer Science, vol. 6633, pp. 191–207. Springer (2011). doi: 10.1007/978-3-642-21040-2_13, https://doi.org/10.1007/978-3-642-21040-2_13

[44] Coppersmith, D.: The data encryption standard (DES) and its strength against attacks. IBM Journal of Research and Development **38**(3), 243–250 (1994). doi: 10.1147/rd.383.0243, https://doi.org/10.1147/rd.383.0243

[45] Daemen, J., Knudsen, L.R., Rijmen, V.: The block cipher square. In: Fast Software Encryption, 4th International Workshop, FSE '97, Haifa, Israel, January 20-22, 1997, Proceedings. pp. 149–165 (1997). doi: 10.1007/BFb0052343, https://doi.org/10.1007/BFb0052343

[46] Daemen, J., Rijmen, V.: AES proposal: Rijndael (1999)

[47] Daemen, J., Rijmen, V.: The wide trail design strategy. In: Cryptography and Coding, 8th IMA International Conference, Cirencester, UK, December 17-19, 2001, Proceedings. pp. 222–238 (2001). doi: 10.1007/3-540-45325-3_20, https://doi.org/10.1007/3-540-45325-3_20

[48] De Mulder, Y., Roelse, P., Preneel, B.: Cryptanalysis of the Xiao–Lai white-box AES implementation. In: International Conference on Selected Areas in Cryptography. pp. 34–49. Springer (2012)

[49] De Mulder, Y., Roelse, P., Preneel, B.: Revisiting the BGE attack on a white-box AES implementation. IACR Cryptology ePrint Archive **2013**, 450 (2013), http://eprint.iacr.org/2013/450

[50] Delerablée, C., Lepoint, T., Paillier, P., Rivain, M.: White-box security notions for symmetric encryption schemes. In: International Conference on Selected Areas in Cryptography. pp. 247–264. Springer (2013)

[51] Derbez, P., Fouque, P.: Exhausting Demirci-Selçuk Meet-in-the-Middle Attacks Against Reduced-Round AES. In: Fast Software Encryption - 20th International Workshop, FSE 2013, Singapore, March 11-13, 2013. Revised Selected Papers. pp. 541–560 (2013)

[52] Derbez, P., Fouque, P., Jean, J., Lambin, B.: Variants of the AES key schedule for better truncated differential bounds. In: Selected Areas in Cryptography - SAC 2018 - 25th International Conference, Calgary, AB, Canada, August 15-17, 2018, Revised Selected Papers. pp. 27–49 (2018). doi: 10.1007/978-3-030-10970-7_2, https://doi.org/10.1007/978-3-030-10970-7_2

[53] Derbez, P., Fouque, P., Lambin, B.: Linearly equivalent s-boxes and the division property. IACR Cryptology ePrint Archive **2019**, 97 (2019), https://eprint.iacr.org/2019/097

[54] Derbez, P., Fouque, P., Lambin, B., Minaud, B.: On recovering affine encodings in white-box implementations. IACR Trans. Cryptogr. Hardw. Embed. Syst. **2018**(3), 121–149 (2018). doi: 10.13154/tches.v2018.i3.121-149, https://doi.org/10.13154/tches.v2018.i3.121-149

[55] Derbez, P., Fouque, P., Lambin, B., Mollimard, V.: Efficient search for optimal diffusion layers of generalized feistel networks. IACR Trans. Symmetric Cryptol. **2019**(2), 218–240 (2019). doi: 10.13154/tosc.v2019.i2.218-240, https://doi.org/10.13154/tosc.v2019.i2.218-240

[56] DES: Data Encryption Standard. FIPS PUB 46, Federal information processing standards publication 46 (1977)

[57] D'Halluin, C., Bijnens, G., Rijmen, V., Preneel, B.: Attack on six rounds of crypton. In: Fast Software Encryption, 6th International Workshop, FSE '99, Rome, Italy, March 24-26, 1999, Proceedings. pp. 46–59 (1999). doi: 10.1007/3-540-48519-8_4, https://doi.org/10.1007/3-540-48519-8_4

[58] Dinu, D., Perrin, L., Udovenko, A., Velichkov, V., Großschädl, J., Biryukov, A.: Design strategies for ARX with provable bounds: Sparx and LAX. In: Advances in Cryptology - ASIACRYPT 2016 - 22nd International Conference on the Theory and Application of Cryptology and Information Security, Hanoi, Vietnam, December 4-8, 2016, Proceedings, Part I. pp. 484–513 (2016). doi: 10.1007/978-3-662-53887-6_18, https://doi.org/10.1007/978-3-662-53887-6_18

[59] Dinur, I.: An improved affine equivalence algorithm for random permutations. In: Advances in Cryptology – EUROCRYPT 2018. Springer (2018)

[60] Dunkelman, O., Keller, N., Shamir, A.: A practical-time related-key attack on the KASUMI cryptosystem used in GSM and 3g telephony. In: Advances in Cryptology - CRYPTO 2010, 30th Annual Cryptology Conference, Santa Barbara, CA, USA, August 15-19, 2010. Proceedings. pp. 393–410 (2010). doi: 10.1007/978-3-642-14623-7_21, https://doi.org/10.1007/978-3-642-14623-7_21

[61] Eskandari, Z., Kidmose, A.B., Kölbl, S., Tiessen, T.: Finding Integral Distinguishers with Ease. IACR Cryptology ePrint Archive (accepted at SAC2018) **2018**, 688 (2018)

[62] Even, S., Mansour, Y.: A construction of a cioher from a single pseudorandom permutation. In: Advances in Cryptology - ASIACRYPT '91, International Conference on the Theory and Applications of Cryptology, Fujiyoshida, Japan, November 11-14, 1991, Proceedings. pp. 210–224 (1991). doi: 10.1007/3-540-57332-1_17, https://doi.org/10.1007/3-540-57332-1_17

[63] Ewing, L.: Tux the penguin, created with The GIMP (1996), lewing@isc.tamu.edu

[64] Feistel, H.: Block cipher cryptographic system (Mar 19 1974), uS Patent 3,798,359

[65] Fouque, P., Jean, J., Peyrin, T.: Structural evaluation of AES and chosen-key distinguisher of 9-round AES-128. In: Canetti, R., Garay, J.A. (eds.) Advances in Cryptology - CRYPTO 2013 - 33rd Annual Cryptology Conference, Santa Barbara, CA, USA, August 18-22, 2013. Proceedings, Part I. Lecture Notes in Computer Science, vol. 8042, pp. 183–203. Springer (2013). doi: 10.1007/978-3-642-40041-4_11, https://doi.org/10.1007/978-3-642-40041-4_11

[66] Fouque, P.A., Karpman, P., Kirchner, P., Minaud, B.: Efficient and provable white-box primitives. In: Advances in Cryptology–ASIACRYPT 2016: 22nd International Conference on the Theory and Application of Cryptology and Information Security, Hanoi, Vietnam, December 4-8, 2016, Proceedings, Part I 22. pp. 159–188. Springer (2016)

[67] Gérault, D., Lafourcade, P., Minier, M., Solnon, C.: Revisiting aes related-key differential attacks with constraint programming. IACR Cryptology ePrint Archive **2017**, 139 (2017), https://eprint.iacr.org/2017/139

[68] Gérault, D., Lafourcade, P., Minier, M., Solnon, C.: Revisiting AES related-key differential attacks with constraint programming. Inf. Process. Lett. **139**, 24–29 (2018). doi: 10.1016/j.ipl.2018.07.001, https://doi.org/10.1016/j.ipl.2018.07.001

[69] Gerault, D., Minier, M., Solnon, C.: Constraint programming models for chosen key differential cryptanalysis. In: Principles and Practice of Constraint Programming - 22nd International Conference, CP 2016, Toulouse, France, September 5-9, 2016, Proceedings. pp. 584–601 (2016). doi: 10.1007/978-3-319-44953-1_37, https://doi.org/10.1007/978-3-319-44953-1_37

[70] Giraud, C.: DFA on AES. In: Advanced Encryption Standard - AES, 4th International Conference, AES 2004, Bonn, Germany, May 10-12, 2004, Revised Selected and Invited Papers. pp. 27–41 (2004). doi: 10.1007/11506447_4, https://doi.org/10.1007/11506447_4

[71] Gueron, S., Mouha, N.: Simpira v2: A family of efficient permutations using the AES round function. In: Advances in Cryptology - ASIACRYPT 2016 - 22nd International Conference on the Theory and Application of Cryptology and Information Security, Hanoi, Vietnam, December 4-8, 2016, Proceedings, Part I. pp. 95–125 (2016). doi: 10.1007/978-3-662-53887-6_4, https://doi.org/10.1007/978-3-662-53887-6_4

[72] Guo, J., Nikolic, I., Peyrin, T., Wang, L.: Cryptanalysis of zorro. IACR Cryptology ePrint Archive **2013**, 713 (2013), http://eprint.iacr.org/2013/713

[73] Guo, J., Peyrin, T., Poschmann, A., Robshaw, M.J.B.: The LED block cipher. In: Cryptographic Hardware and Embedded Systems - CHES 2011 - 13th International Workshop, Nara, Japan, September 28 - October 1, 2011. Proceedings. pp. 326–341 (2011). doi: 10.1007/978-3-642-23951-9_22, https://doi.org/10.1007/978-3-642-23951-9_22

[74] Gurobi Optimization, L.: Gurobi optimizer reference manual (2018), http://www.gurobi.com

[75] Hall-Andersen, M., Vejre, P.S.: Generating graphs packed with paths estimation of linear approximations and differentials. IACR Trans. Symmetric Cryptol. **2018**(3), 265–289 (2018). doi: 10.13154/tosc.v2018.i3.265-289, https://doi.org/10.13154/tosc.v2018.i3.265-289

[76] Hong, D., Sung, J., Hong, S., Lim, J., Lee, S., Koo, B., Lee, C., Chang, D., Lee, J., Jeong, K., Kim, H., Kim, J., Chee, S.: HIGHT: A new block cipher suitable for low-resource device. In: Cryptographic Hardware and Embedded Systems - CHES 2006, 8th International Workshop, Yokohama, Japan, October 10-13, 2006, Proceedings. pp. 46–59 (2006). doi: 10.1007/11894063_4, https://doi.org/10.1007/11894063_4

[77] Jacob, M., Boneh, D., Felten, E.: Attacking an obfuscated cipher by injecting faults. ACM Workshop on Digital Rights Management (2002)

[78] Jean, J.: TikZ for Cryptographers. https://www.iacr.org/authors/tikz/ (2016)

[79] Jean, J., Nikolic, I.: Efficient Design Strategies Based on the AES Round Function. In: Peyrin, T. (ed.) Fast Software Encryption - 23rd International Conference, FSE 2016, Bochum, Germany, March 20-23, 2016, Revised Selected Papers. Lecture Notes in Computer Science, vol. 9783, pp. 334–353. Springer (2016). doi: 10.1007/978-3-662-52993-5_17, https://doi.org/10.1007/978-3-662-52993-5_17

[80] Jean, J., Nikolic, I., Peyrin, T.: Tweaks and keys for block ciphers: The TWEAKEY framework. In: Advances in Cryptology - ASIACRYPT 2014 - 20th International Conference on the Theory and Application of Cryptology and Information Security, Kaoshiung, Taiwan, R.O.C., December 7-11, 2014, Proceedings, Part II. pp. 274–288 (2014). doi: 10.1007/978-3-662-45608-8_15, https://doi.org/10.1007/978-3-662-45608-8_15

[81] Kales, D., Perrin, L., Promitzer, A., Ramacher, S., Rechberger, C.: Improvements to the linear layer of lowmc: A faster picnic. IACR Cryptology ePrint Archive **2017**, 1148 (2017), http://eprint.iacr.org/2017/1148

[82] Kanda, M.: Practical security evaluation against differential and linear cryptanalyses for feistel ciphers with SPN round function. In: Selected Areas in Cryptography, 7th Annual International Workshop, SAC 2000, Waterloo, Ontario, Canada, August 14-15, 2000, Proceedings. pp. 324–338 (2000). doi: 10.1007/3-540-44983-3_24, https://doi.org/10.1007/3-540-44983-3_24

[83] Karroumi, M.: Protecting white-box AES with dual ciphers. In: International Conference on Information Security and Cryptology. pp. 278–291. Springer (2010)

[84] Khoo, K., Lee, E., Peyrin, T., Sim, S.M.: Human-readable proof of the related-key security of AES-128. IACR Trans. Symmetric Cryptol. **2017**(2), 59–83 (2017). doi: 10.13154/tosc.v2017.i2.59-83, https://doi.org/10.13154/tosc.v2017.i2.59-83

[85] Kim, J., Hong, S., Sung, J., Lee, C., Lee, S.: Impossible differential cryptanalysis for block cipher structures. In: Progress in Cryptology - INDOCRYPT 2003, 4th International Conference on Cryptology in India, New Delhi, India, December 8-10, 2003, Proceedings. pp. 82–96 (2003). doi: 10.1007/978-3-540-24582-7_6, https://doi.org/10.1007/978-3-540-24582-7_6

[86] Kirkpatrick, S., Gelatt, C.D., Vecchi, M.P.: Optimization by simulated annealing. science **220**(4598), 671–680 (1983)

[87] Knudsen, L.: Deal - a 128-bit block cipher. In: NIST AES Proposal (1998)

[88] Knudsen, L.R.: Truncated and higher order differentials. In: Fast Software Encryption: Second International Workshop. Leuven, Belgium, 14-16 December 1994, Proceedings. pp. 196–211 (1994). doi: 10.1007/3-540-60590-8_16, https://doi.org/10.1007/3-540-60590-8_16

[89] Knudsen, L.R., Wagner, D.A.: Integral cryptanalysis. In: Fast Software Encryption, 9th International Workshop, FSE 2002, Leuven, Belgium, February 4-6, 2002, Revised Papers. pp. 112–127 (2002). doi: 10.1007/3-540-45661-9_9, https://doi.org/10.1007/3-540-45661-9_9

[90] Kocher, P.C., Jaffe, J., Jun, B.: Differential power analysis. In: Advances in Cryptology - CRYPTO '99, 19th Annual International Cryptology Conference, Santa Barbara, California, USA, August 15-19, 1999, Proceedings. pp. 388–397 (1999). doi: 10.1007/3-540-48405-1_25, https://doi.org/10.1007/3-540-48405-1_25

[91] Koo, B., Hong, D., Kwon, D.: Related-key attack on the full HIGHT. In: Information Security and Cryptology - ICISC 2010 - 13th International Conference, Seoul, Korea, December 1-3, 2010, Revised Selected Papers. pp. 49–67 (2010). doi: 10.1007/978-3-642-24209-0_4, https://doi.org/10.1007/978-3-642-24209-0_4

[92] Lai, X.: Higher order derivatives and differential cryptanalysis. In: Communications and Cryptography, pp. 227–233. Springer (1994)

[93] Lai, X., Massey, J.L.: A proposal for a new block encryption standard. In: Advances in Cryptology - EUROCRYPT '90, Workshop on the Theory and Application of of Cryptographic Techniques, Aarhus, Denmark, May 21-24, 1990, Proceedings. pp. 389–404 (1990). doi: 10.1007/3-540-46877-3_35, https://doi.org/10.1007/3-540-46877-3_35

[94] Lallemand, V., Naya-Plasencia, M.: Cryptanalysis of KLEIN. In: Fast Software Encryption - 21st International Workshop, FSE 2014, London, UK, March 3-5, 2014. Revised Selected Papers. pp. 451–470 (2014). doi: 10.1007/978-3-662-46706-0_23, https://doi.org/10.1007/978-3-662-46706-0_23

[95] Langford, S.K., Hellman, M.E.: Differential-linear cryptanalysis. In: Advances in Cryptology - CRYPTO '94, 14th Annual International Cryptology Conference, Santa Barbara, California, USA, August 21-25, 1994, Proceedings. pp. 17–25 (1994). doi: 10.1007/3-540-48658-5_3, https://doi.org/10.1007/3-540-48658-5_3

[96] Leander, G., Minaud, B., Rønjom, S.: A generic approach to invariant subspace attacks: Cryptanalysis of robin, iscream and zorro. In: Advances in Cryptology - EUROCRYPT 2015 - 34th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Sofia, Bulgaria, April 26-30, 2015, Proceedings, Part I. pp. 254–283 (2015). doi: 10.1007/978-3-662-46800-5_11, https://doi.org/10.1007/978-3-662-46800-5_11

[97] Leander, G., Poschmann, A.: On the classification of 4 bit s-boxes. In: Arithmetic of Finite Fields, First International Workshop, WAIFI 2007, Madrid, Spain, June 21-22, 2007, Proceedings. pp. 159–176 (2007). doi: 10.1007/978-3-540-73074-3_13, https://doi.org/10.1007/978-3-540-73074-3_13

[98] Lepoint, T., Rivain, M., De Mulder, Y., Roelse, P., Preneel, B.: Two attacks on a white-box AES implementation. In: International Conference on Selected Areas in Cryptography. pp. 265–285. Springer (2013)

[99] Liu, G., Ghosh, M., Song, L.: Security analysis of skinny under related-tweakey settings. IACR Transactions on Symmetric Cryptology **2017**(3), 37–72 (2017)

[100] Luby, M., Rackoff, C.: How to Construct Pseudorandom Permutations from Pseudorandom Functions. SIAM J. Comput. **17**(2), 373–386 (1988). doi: 10.1137/0217022, https://doi.org/10.1137/0217022

[101] Luo, Y., Lai, X., Wu, Z., Gong, G.: A unified method for finding impossible differentials of block cipher structures. Inf. Sci. **263**, 211–220 (2014). doi: 10.1016/j.ins.2013.08.051, https://doi.org/10.1016/j.ins.2013.08.051

[102] Mala, H., Shakiba, M., Dakhilalian, M.: New impossible differential attacks on reduced-round crypton. Computer Standards & Interfaces **32**(4), 222–227 (2010). doi: 10.1016/j.csi.2009.11.011, https://doi.org/10.1016/j.csi.2009.11.011

[103] Matsui, M.: On correlation between the order of s-boxes and the strength of DES. In: Santis, A.D. (ed.) Advances in Cryptology - EUROCRYPT '94, Workshop on the Theory and Application of Cryptographic Techniques, Perugia, Italy, May 9-12, 1994, Proceedings. Lecture Notes in Computer Science, vol. 950, pp. 366–375. Springer (1994). doi: 10.1007/BFb0053451, https://doi.org/10.1007/BFb0053451

[104] Matsui, M.: New block encryption algorithm MISTY. In: Fast Software Encryption, 4th International Workshop, FSE '97, Haifa, Israel, January 20-22, 1997, Proceedings. pp. 54–68 (1997). doi: 10.1007/BFb0052334, https://doi.org/10.1007/BFb0052334

[105] Matsui, M., Yamagishi, A.: A new method for known plaintext attack of FEAL cipher. In: Advances in Cryptology - EUROCRYPT '92, Workshop on the Theory and Application of of Cryptographic Techniques, Balatonfüred, Hungary, May 24-28, 1992, Proceedings. pp. 81–91 (1992). doi: 10.1007/3-540-47555-9_7, https://doi.org/10.1007/3-540-47555-9_7

[106] Michiels, W., Gorissen, P., Hollmann, H.D.L.: Cryptanalysis of a generic class of white-box implementations. In: Selected Areas in Cryptography, 15th International Workshop, SAC 2008, Sackville, New Brunswick, Canada, August 14-15, Revised Selected Papers. pp. 414–428 (2008). doi: 10.1007/978-3-642-04159-4_27, https://doi.org/10.1007/978-3-642-04159-4_27

[107] Minaud, B., Derbez, P., Fouque, P.A., Karpman, P.: Key-recovery attacks on ASASA. In: International Conference on the Theory and Application of Cryptology and Information Security (ASIACRYPT). pp. 3–27. Springer (2015)

[108] Moriai, S., Shimoyama, T., Kaneko, T.: Higher order differential attak of CAST cipher. In: Fast Software Encryption, 5th International Workshop, FSE '98, Paris, France, March 23-25, 1998, Proceedings. pp. 17–31 (1998). doi: 10.1007/3-540-69710-1_2, https://doi.org/10.1007/3-540-69710-1_2

[109] Mouha, N., Wang, Q., Gu, D., Preneel, B.: Differential and linear cryptanalysis using mixed-integer linear programming. In: Information Security and Cryptology - 7th International Conference, Inscrypt 2011, Beijing, China, November 30 - December 3, 2011. Revised Selected Papers. pp. 57–76 (2011). doi: 10.1007/978-3-642-34704-7_5, https://doi.org/10.1007/978-3-642-34704-7_5

[110] Nikolic, I.: Tweaking AES. In: Biryukov et al. [23], pp. 198–210. doi: 10.1007/978-3-642-19574-7_14, https://doi.org/10.1007/978-3-642-19574-7_14

[111] Nikolic, I.: How to use metaheuristics for design of symmetric-key primitives. In: Takagi, T., Peyrin, T. (eds.) Advances in Cryptology - ASIACRYPT 2017 - 23rd International Conference on the Theory and Applications of Cryptology and Information Security, Hong Kong, China, December 3-7, 2017, Proceedings, Part III. Lecture Notes in Computer Science, vol. 10626, pp. 369–391. Springer (2017). doi: 10.1007/978-3-319-70700-6_13, https://doi.org/10.1007/978-3-319-70700-6_13

[112] Nyberg, K.: Generalized Feistel Networks. In: Advances in Cryptology - ASIACRYPT '96, International Conference on the Theory and Applications of Cryptology and Information Security, Kyongju, Korea, November 3-7, 1996, Proceedings. pp. 91–104 (1996). doi: 10.1007/BFb0034838, https://doi.org/10.1007/BFb0034838

[113] Pieprzyk, J.: How to Construct Pseudorandom Permutations from Single Pseudorandom Functions. In: Advances in Cryptology - EUROCRYPT '90, Workshop on the Theory and Application of of Cryptographic Techniques, Aarhus, Denmark, May 21-24, 1990, Proceedings. pp. 140–150 (1990). doi: 10.1007/3-540-46877-3_12, https://doi.org/10.1007/3-540-46877-3_12

[114] Rivest, R.L., Shamir, A., Adleman, L.M.: A method for obtaining digital signatures and public-key cryptosystems. Commun. ACM **21**(2), 120–126 (1978). doi: 10.1145/359340.359342, http://doi.acm.org/10.1145/359340.359342

[115] Sasaki, Y., Todo, Y.: New impossible differential search tool from design and cryptanalysis aspects - revealing structural properties of several ciphers. In: Advances in Cryptology - EUROCRYPT 2017 - 36th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Paris, France, April 30 - May 4, 2017, Proceedings, Part III. pp. 185–215 (2017). doi: 10.1007/978-3-319-56617-7_7, https://doi.org/10.1007/978-3-319-56617-7_7

[116] Sasdrich, P., Moradi, A., Güneysu, T.: White-box cryptography in the gray box. In: International Conference on Fast Software Encryption (FSE). pp. 185–203. Springer (2016)

[117] Schneier, B., Kelsey, J., Whiting, D., Wagner, D., Hall, C., Ferguson, N.: Twofish: A 128-bit block cipher. NIST AES Proposal **15**, 23 (1998)

[118] Shannon, C.E.: Communication theory of secrecy systems. Bell system technical journal **28**(4), 656–715 (1949)

[119] Shibayama, N., Kaneko, T.: A new higher order differential of CLEFIA. IEICE Transactions **97-A**(1), 118–126 (2014). doi: 10.1587/transfun.E97.A.118, https://doi.org/10.1587/transfun.E97.A.118

[120] Shibutani, K.: On the diffusion of generalized feistel structures regarding differential and linear cryptanalysis. In: Biryukov et al. [23], pp. 211–228. doi: 10.1007/978-3-642-19574-7_15, https://doi.org/10.1007/978-3-642-19574-7_15

[121] Shibutani, K., Isobe, T., Hiwatari, H., Mitsuda, A., Akishita, T., Shirai, T.: Piccolo: An Ultra-Lightweight Blockcipher. In: Cryptographic Hardware and Embedded Systems - CHES 2011 - 13th International Workshop, Nara, Japan, September 28 - October 1, 2011. Proceedings. pp. 342–357 (2011). doi: 10.1007/978-3-642-23951-9_23, https://doi.org/10.1007/978-3-642-23951-9_23

[122] Shirai, T., Shibutani, K., Akishita, T., Moriai, S., Iwata, T.: The 128-Bit Blockcipher CLEFIA (Extended Abstract). In: Fast Software Encryption, 14th International Workshop, FSE 2007, Luxembourg, Luxembourg, March 26-28, 2007, Revised Selected Papers. pp. 181–195 (2007). doi: 10.1007/978-3-540-74619-5_12, https://doi.org/10.1007/978-3-540-74619-5_12

[123] Shoup, V.: NTL: A library for doing number theory. www. shoup. net/ntl/ (2001)

[124] Sterndark, D.: Rc4 algorithm revealed. Usenet posting, Message-ID:< sternCvKL4B. Hyy@ netcom. com (1994)

[125] Sugio, N., Aono, H., Hongo, S., Kaneko, T.: A study on higher order differential attack of KASUMI. IEICE Transactions **90-A**(1), 14–21 (2007). doi: 10.1093/ietfec/e90-a.1.14, https://doi.org/10.1093/ietfec/e90-a.1.14

[126] Sugio, N., Aono, H., Sekino, K., Kaneko, T.: A new higher order differential of camellia. In: International Symposium on Information Theory and its Applications, ISITA 2014, Melbourne, Australia, October 26-29, 2014. pp. 478–482 (2014), http://ieeexplore.ieee.org/document/6979889/

[127] Sun, L., Wang, W., Liu, R., Wang, M.: MILP-Aided Bit-Based Division Property for ARX-Based Block Cipher. IACR Cryptology ePrint Archive **2016**, 1101 (2016)

[128] Sun, L., Wang, W., Wang, M.: MILP-Aided Bit-Based Division Property for Primitives with Non-Bit-Permutation Linear Layers. IACR Cryptology ePrint Archive **2016**, 811 (2016)

[129] Sun, L., Wang, W., Wang, M.: Automatic Search of Bit-Based Division Property for ARX Ciphers and Word-Based Division Property. In: Advances in Cryptology - ASIACRYPT 2017 - 23rd International Conference on the Theory and Applications of Cryptology and Information Security, Hong Kong, China, December 3-7, 2017, Proceedings, Part I. pp. 128–157 (2017)

[130] Sun, S., Gerault, D., Lafourcade, P., Yang, Q., Todo, Y., Qiao, K., Hu, L.: Analysis of aes, skinny, and others with constraint programming. IACR Trans. Symmetric Cryptol. **2017**(1), 281–306 (2017). doi: 10.13154/tosc.v2017.i1.281-306, https://doi.org/10.13154/tosc.v2017.i1.281-306

[131] Suzaki, T., Minematsu, K.: Improving the generalized Feistel. In: International Workshop on Fast Software Encryption. pp. 19–39. Springer (2010)

[132] Suzaki, T., Minematsu, K., Morioka, S., Kobayashi, E.: TWINE : A Lightweight Block Cipher for Multiple Platforms. In: Selected Areas in Cryptography, 19th International Conference, SAC 2012, Windsor, ON, Canada, August 15-16, 2012, Revised Selected Papers. pp. 339–354 (2012). doi: 10.1007/978-3-642-35999-6_22, https://doi.org/10.1007/978-3-642-35999-6_22

[133] Tardy-Corfdir, A., Gilbert, H.: A known plaintext attack of FEAL-4 and FEAL-6. In: Advances in Cryptology - CRYPTO '91, 11th Annual International Cryptology Conference, Santa Barbara, California, USA, August 11-15, 1991, Proceedings. pp. 172–181 (1991). doi: 10.1007/3-540-46766-1_12, https://doi.org/10.1007/3-540-46766-1_12

[134] The Sage Developers: SageMath, the Sage Mathematics Software System (Version 8.0) (2017), http://www.sagemath.org

[135] Todo, Y.: Integral cryptanalysis on full MISTY1. In: Advances in Cryptology - CRYPTO 2015 - 35th Annual Cryptology Conference, Santa Barbara, CA, USA, August 16-20, 2015, Proceedings, Part I. pp. 413–432 (2015). doi: 10.1007/978-3-662-47989-6_20, https://doi.org/10.1007/978-3-662-47989-6_20

[136] Todo, Y.: Structural Evaluation by Generalized Integral Property. In: Advances in Cryptology - EUROCRYPT 2015 - 34th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Sofia, Bulgaria, April 26-30, 2015, Proceedings, Part I. pp. 287–314 (2015)

[137] Todo, Y., Morii, M.: Bit-Based Division Property and Application to Simon Family. In: Fast Software Encryption - 23rd International Conference, FSE 2016, Bochum, Germany, March 20-23, 2016, Revised Selected Papers. pp. 357–377 (2016)

[138] Wagner, D.A.: The boomerang attack. In: Fast Software Encryption, 6th International Workshop, FSE '99, Rome, Italy, March 24-26, 1999, Proceedings. pp. 156–170 (1999). doi: 10.1007/3-540-48519-8_12, https://doi.org/10.1007/3-540-48519-8_12

[139] Wang, Q., Hao, Y., Todo, Y., Li, C., Isobe, T., Meier, W.: Improved Division Property Based Cube Attacks Exploiting Algebraic Properties of Superpoly. In: Advances in Cryptology - CRYPTO 2018 - 38th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 19-23, 2018, Proceedings, Part I. pp. 275–305 (2018)

[140] Wheeler, D.J., Needham, R.M.: Tea, a tiny encryption algorithm. In: Fast Software Encryption: Second International Workshop. Leuven, Belgium, 14-16 December 1994, Proceedings. pp. 363–366 (1994). doi: 10.1007/3-540-60590-8_29, https://doi.org/10.1007/3-540-60590-8_29

[141] Wu, H.: Acorn: a lightweight authenticated cipher (v3). Candidate for the CAESAR Competition. See also https://competitions.cr.yp.to/round3/acornv3.pdf (2016)

[142] Wu, S., Wang, M.: Automatic search of truncated impossible differentials for word-oriented block ciphers. In: Progress in Cryptology - INDOCRYPT 2012, 13th International Conference on Cryptology in India, Kolkata, India, December 9-12, 2012. Proceedings. pp. 283–302 (2012). doi: 10.1007/978-3-642-34931-7_17, https://doi.org/10.1007/978-3-642-34931-7_17

[143] Xiang, Z., Zhang, W., Bao, Z., Lin, D.: Applying MILP Method to Searching Integral Distinguishers Based on Division Property for 6 Lightweight Block Ciphers. In: Advances in Cryptology - ASIACRYPT 2016 - 22nd International Conference on the Theory and Application of Cryptology and Information Security, Hanoi, Vietnam, December 4-8, 2016, Proceedings, Part I. pp. 648–678 (2016)

[144] Xiao, Y., Lai, X.: A secure implementation of white-box AES. In: Computer Science and its Applications, 2009. CSA'09. 2nd International Conference on. pp. 1–6. IEEE (2009)

[145] Zhang, W., Bao, Z., Lin, D., Rijmen, V., Yang, B., Verbauwhede, I.: RECTANGLE: a bit-slice lightweight block cipher suitable for multiple platforms. SCIENCE CHINA Information Sciences **58**(12), 1–15 (2015)

[146] Zhang, W., Rijmen, V.: Division Cryptanalysis of Block Ciphers with a Binary Diffusion Layer. IACR Cryptology ePrint Archive **2017**, 188 (2017)

[147] Zheng, Y., Matsumoto, T., Imai, H.: On the Construction of Block Ciphers Provably Secure and Not Relying on Any Unproved Hypotheses. In: Advances in Cryptology - CRYPTO '89, 9th Annual International Cryptology Conference, Santa Barbara, California, USA, August 20-24, 1989, Proceedings. pp. 461–480 (1989). doi: 10.1007/0-387-34805-0_42, https://doi.org/10.1007/0-387-34805-0_42

[148] Zong, R., Dong, X., Wang, X.: Milp-aided related-tweak/key impossible differential attack and its applications to qarma, joltik-bc. Cryptology ePrint Archive, Report 2018/142 (2018), https://eprint.iacr.org/2018/142

UNIVERSITE BRETAGNE LOIRE MATHSTIC

UNIVERSITÉ DE RENNES 1

**Titre :** Optimisation des Principaux Composants des Chiffrements par Bloc

**Mots clés :** Cryptographie, symétrique, conception, cryptanalyse

**Résumé :** La sécurité des chiffrements par bloc évolue constamment au fur et à mesure que de nouvelles techniques de cryptanalyse sont découvertes. Lors de la conception de nouveaux chiffrements par bloc, il est donc nécessaire de considérer ces nouvelles techniques dans l'analyse de sécurité. Dans cette thèse, nous montrons comment construire certaines opérations internes des chiffrements par bloc pour améliorer la résistance à certaines attaques.
Nous commençons par donner une méthode pour trouver les permutations paires-impaires optimales selon un certain critère pour les Réseaux de Feistel Généralisés. Grâce à une nouvelle caractérisation et à un algorithme efficace, nous sommes notamment capables de résoudre un problème ouvert depuis 10 ans.
Nous donnons ensuite de nouvelles techniques de cryptanalyse pour améliorer la *division property*, qui nous permet également de donner un nouveau critère optimal pour la conception de boîtes-S.
Nous continuons avec de nouvelles observations pour un cadencement de clé alternatif pour AES. Ceci nous permet de donner un nouveau cadencement de clé, à la fois plus efficace et augmentant la sécurité face à certaines attaques par rapport à l'original.
Pour finir, nous présentons un algorithme général très efficace permettant d'attaquer la majorité des propositions pour la cryptographie en boîte blanche, ainsi qu'une attaque dédiée sur un schéma non attaqué jusque là, donnant lieu à une attaque qui n'a besoin que de quelques secondes pour retrouver la clé.

**Title :** Optimization of Core Components of Block Ciphers

**Keywords :** Cryptography, symmetric, design, cryptanalysis

**Abstract :** Along with new cryptanalysis techniques, the security of block ciphers is always evolving. When designing new block ciphers, we thus need to consider these new techniques during the security analysis. In this thesis, we show how to build some core operations for block ciphers to improve the security against some attacks.
We first start by describing a method to find optimal (according to some criterion) even-odd permutations for a Generalized Feistel Network. Using a new characterization and an efficient algorithm, we are able to solve a 10-years old problem.
We then give new cryptanalysis techniques to improve the division property, along with a new proven optimal criterion for designing S-boxes.
We continue with new observations for the design of an alternative key-schedule for AES. We thus give a new key-schedule, which is both more efficient and more secure against some attacks compared to the original one.
Finally, we describe a very efficient generic algorithm to break most proposals in white-box cryptography, as well as a dedicated attack on a previously not analyzed scheme, leading to a key-recovery attack in a few seconds.