UNIVERSITÉ DE LORRAINE

École doctorale IAEM Lorraine

Analyse et Conception d'Algorithmes de Chiffrement Légers

THÈSE

présentée et soutenue publiquement le 26 novembre 2020

pour l'obtention du

Doctorat de l'Université de Lorraine
(mention informatique)

par

Paul Huynh

Composition du jury

Président :	Emmanuel Jeandel	Professeur, Université de Lorraine
Rapporteurs :	Pierre-Alain Fouque	Professeur, Université Rennes 1
	François-Xavier Standaert	Professeur, Université Catholique de Louvain
Examinateurs :	Christina Boura	Maîtresse de Conférences, Université de Versailles Saint-Quentin-en-Yvelines
	Virginie Lallemand	Chargée de Recherche CNRS, Nancy
Directrice de thèse :	Marine Minier	Professeure, Université de Lorraine

Laboratoire Lorrain de Recherche en Informatique et ses Applications — UMR 7503

Remerciements

"I don't know half of you half as well as I should like;
and I like less than half of you half as well as you deserve."

— Bilbo Baggins

All work and no play makes Jack a dull boy
All w ork and no play makes Jack a dull boy
All work and no ,play makesJack a dull boy

Contents

Introduction Générale

Ça va être dur de faire plus dur que si dur.

Un homme de lettres

La cryptologie, ou étymologiquement "la science des codes secrets", comprend la *cryptographie*, qui se concentre sur l'ensemble des techniques de communication visant à protéger la confidentialité des données, et la *cryptanalyse*, qui s'attèle à trouver des faiblesses dans les méthodes de chiffrement proposées. Ces deux aspects de la cryptologie sont évidemment complémentaires, une avancée dans un domaine poussant l'autre à proposer des solutions plus efficaces.

De la substitution simple, qui permit à Jules César de communiquer avec ses armées par messages chiffrés, à la machine Enigma utilisée par les forces de l'Axe lors de la seconde guerre mondiale, la cryptologie est restée pendant très longtemps réservée à un usage principalement militaire. Ce n'est qu'avec le développement de l'informatique et l'arrivée des ordinateurs personnels qu'elle est devenue un domaine de recherche actif, s'imposant progressivement dans notre quotidien. On compte aujourd'hui un certains nombre de primitives cryptographiques, tels le chiffrement Rsa [RSA78] ou encore l'Aes [AES01, DR02], standard de chiffrement depuis 2001.

La cryptographie moderne

Avant toute chose, rappelons les objectifs de la cryptographie moderne. Deux protagonistes, Alice et Bob, veulent communiquer en présence d'un adversaire, Eve. Pour que des utilisateurs non-légitimes comme Eve ne puissent pas accéder au contenu du message qu'Alice envoie à Bob, ce message est *chiffré* au moyen d'une transformation, un *cryptosystème*, qui apporte les garanties suivantes :

- *Confidentialité.* Bob doit être le seul à pouvoir *déchiffrer* le message d'Alice. Si Eve venait à intercepter le message chiffré d'Alice, il serait impossible pour elle d'en extraire une quelconque information.
- *Intégrité.* Si Eve change le contenu du message d'Alice, Bob doit être en mesure de le détecter.
- *Authenticité.* Bob doit être en mesure de vérifier que le message provient bien d'Alice. Eve ne peut donc pas communiquer avec Bob en prétendant être Alice.

La Figure 1 montre ce procédé. Les algorithmes de chiffrement se répartissent en deux grandes familles : les algorithmes *symétriques*, aussi appelés algorithmes *à clef secrète*, et les algorithmes *asymétriques*, ou encore *à clef publique* :

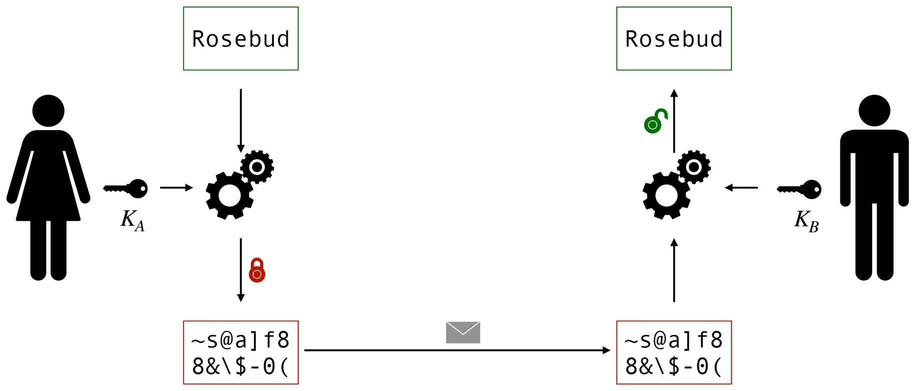


Figure 1: Alice veut envoyer un message à Bob, chiffré avec une clef K_A. Seul Bob peut retrouver le message à l'aide de sa clef de déchiffrement K_B. Pour un chiffrement symétrique, Alice et Bob partagent la même clef secrète, *i.e.* $K_A = K_B$.

- La première famille utilise une seule et même clef pour le chiffrement et le déchiffrement, ce qui nécessite le partage d'un secret entre les deux entités souhaitant communiquer.

- Pour les algorithmes à clef publique, comme RSA, chaque entité dispose d'une paire de clef : une clef de chiffrement, publique, et une clef de déchiffrement, secrète.

La seconde approche permet ainsi d'éviter le problème du partage de secret commun qui est propre aux algorithmes symétriques. Le sécurité des cryptosystèmes asymétriques repose sur des problèmes issus de la théorie des nombres considérés comme difficiles, tels la décomposition en facteurs premiers ou la résolution du logarithme discret dans un corps fini : on démontre en général que casser un tel système est équivalent à résoudre un certain problème mathématique jugé difficile. Les arguments de sécurité s'appuient ainsi sur des résultats de complexité asymptotique. Par ailleurs, certains travaux ont mené à des algorithmes quasi-polynomiaux permettant de résoudre ces problèmes dans certains cas [Jou14, BGJT14, KW19, BGG+20], montrant qu'il est possible que des algorithme rapides pour résoudre ces problèmes sous-jacents finissent par voir le jour. Enfin, il convient aussi d'évoquer la menace d'un éventuel ordinateur quantique, auquel certains de ces problèmes mathématiques ne résisteraient pas.

Un second type de difficulté concerne la mise en pratique de tels systèmes : les problèmes difficiles mentionnés précédemment manipulent de grands nombres d'éléments, dans des corps de grande dimension, ce qui implique des coûts d'implémentation trop élevés ou des débits trop lents pour certaines applications, comme par exemple, le chiffrement de disque dur. Seuls les algorithmes symétriques sont capables d'atteindre ces exigences d'implémentation.

Ainsi, dans la pratique, des systèmes hybrides sont utilisés : un système à clef publique va permettre à deux interlocuteurs Alice et Bob de s'échanger une clef de session, qui servira à son tour à paramétrer un algorithme de chiffrement symétrique. C'est sur cette seconde famille d'algorithmes de chiffrement que se concentre cette thèse.

Plan détaillé de la thèse

Ce manuscrit rassemble les différents travaux effectués durant ma thèse à l'Université de Lorraine, sous la direction de Marine Minier, au sein de l'équipe CARAMBA.

Ces travaux de recherche portent principalement sur un sous-domaine particulier de la cryptologie symétrique : la cryptographie dite *à bas coût* ou encore *légère* (*lightweight cryptography* en anglais), qui a pour but de concevoir des algorithmes de chiffrement dédiés aux objets connectés. Ces nouveaux appareils, apparus au cours des vingt dernières années et regroupés sous le concept d'Internet des Objets (IoT), sont en effet généralement limités en ressources, créant un besoin de nouvelles primitives cryptographiques demandant peu de place en mémoire, ayant une faible consommation d'énergie, tout en conservant une vitesse d'exécution rapide.

Dans les différents chapitres de ce manuscrit, seront donc étudiées les méthodes de conceptions et d'analyse de tels chiffrements légers. Le contenu est divisé en trois partie, détaillées ci-dessous.

Partie I. Contexte général

Cette première partie a pour but de situer le contexte des contributions qui seront ensuite présentées dans les Parties II et III. Ainsi, après une présentation des concepts fondamentaux de la cryptographie moderne, nous nous intéresserons à un des défis ouverts de ce domaine de recherche, la cryptographie légère. Plusieurs techniques de conception d'algorithmes efficaces et légers seront étudiées et nous détaillerons également certaines méthodes utilisées en cryptanalyse symétrique qui seront appliquées dans la suite du manuscrit.

Chapitre 1. Cryptographie symétrique et chiffrements par blocs

Ce chapitre présente les objectifs principaux de la cryptographie et s'intéresse aux solutions mises en oeuvre dans le domaine de la cryptographie symétrique. En particulier, il décrit les méthodes de conception des algorithmes de chiffrement symétrique qui utilisent des *chiffrements par blocs*. Nous verrons en effet que construire des primitives cryptographiques peut se décomposer en deux sous-problèmes :

1. concevoir une famille de 2^k permutations de $\mathbb{F}_2^n$ se comportant comme une permutation aléatoire, où k désigne la taille de clef secrète (typiquement 128 bits), et n est la taille de bloc (128 bits également),

2. décrire un *mode opératoire* permettant de manipuler des messages de tailles quelconque.

Une partie de la thèse s'attardera davantage sur le premier point, à savoir la conception d'une transformation pouvant manipuler des messages de taille fixe. Une technique classique pour concevoir un algorithme de chiffrement par blocs est d'utiliser un chiffrement *itératif* : l'idée consiste à appliquer successivement une permutation paramétrée par une quantité secrète (une *sous-clef*) en partant du message clair d'origine pour obtenir le message chiffré. Ce type de construction est illustrée sur la Figure 2.

Cette structure présente deux avantages. Elle permet tout d'abord d'obtenir des chiffrements complexes en ne construisant qu'une fonction de tour, ce qui réduit considérablement le coût d'implémentation. De plus, elle facilite l'analyse du chiffrement puisqu'il est plus simple de fournir des arguments de sécurité pour une petite fonction simple itérée plusieurs fois que sur une grande transformation.

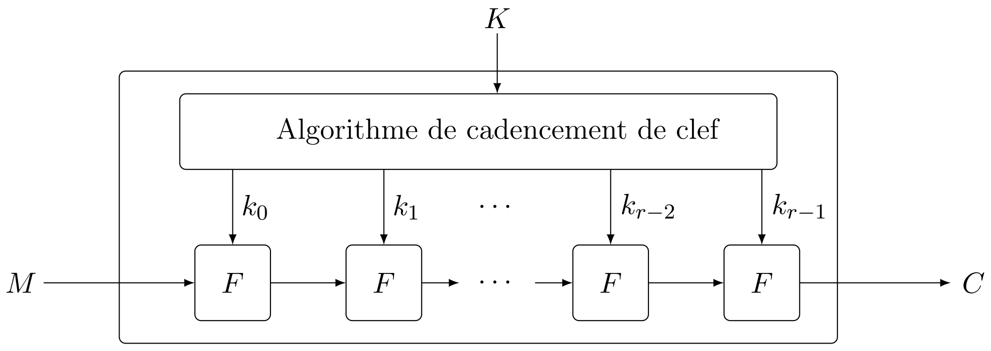


Figure 2: Un chiffrement par blocs itératif: la fonction de tour F est appliquée successivement, à chaque fois avec une sous-clef k_i différente dérivée de la clef de chiffrement K.

Ainsi, la conception d'un chiffrement par blocs est réduit à deux problèmes : la spécification d'un algorithme de cadencement de clef pour dériver des sous-clef de tour et une fonction de tour F.

Pour la fonction de tour, l'approche usuelle consiste à utiliser deux couches, chacune permettant d'obtenir une des propriétés de *confusion* et de *diffusion* énoncées par Claude Shannon dans son article fondateur de 1949 [Sha49] :

- une couche linéaire, qui va créer de la dépendance entre tous les bits de clef, les bits de clairs et les bits de chiffrés,
- une couche non-linéaire, pour casser les structures algébriques, souvent à base de substitution.

Ces techniques de chiffrement ont été grandement utilisées et étudiées et l'apogée de ce genre de construction est le choix du standard actuel de chiffrement symétrique, l'AES [AES01], qui utilise une concaténation d'une même table de substitution—la boîte-S—sur 8 bits dans sa couche non-linéaire, et une multiplication par une matrice dite MDS[1], permettant d'optimiser la diffusion de sa couche linéaire. La fonction de tour de l'AES est illustrée à la Figure 3.

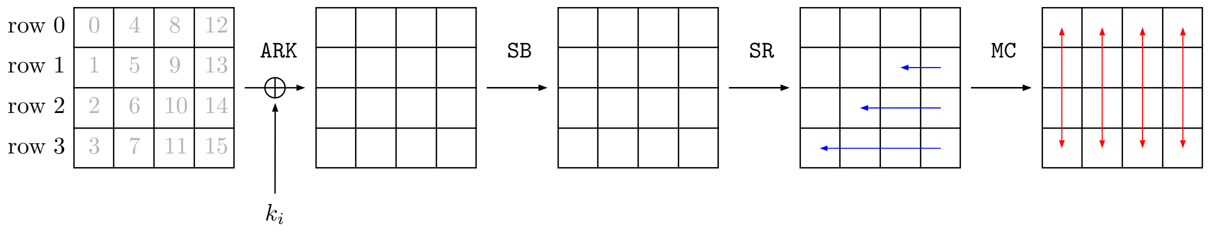


Figure 3: Fonction de tour de l'AES.

Chapitre 2. Cryptographie légère

Dans ce chapitre, nous nous intéresserons davantage à la problématique sous-jacente du coût d'implémentation et des performances, déjà évoquée dans le chapitre précédent.

[1]Maximum Distance Separation. En théorie des codes, les codes MDS ont la meilleure capacité de correction et de détection d'erreurs.

Alors même que l'AES venait d'être choisi comme nouveau standard par le NIST[2], le début des années 2000 vit apparaître un certain nombre d'applications utilisant des plateformes limitées en ressources, telles les cartes de transports ou les systèmes de verrouillage à distance de voitures. Aujourd'hui, la prolifération de ces petits objets connectés et le flux de données transitantes qui en résulte ont créé de nouveaux besoins auxquels les standards actuels comme l'AES ne sont pas adaptés. Depuis quelques années, la communauté cryptographique s'efforce donc de proposer de nouveaux algorithmes consommant très peu d'énergie, utilisant très peu de portes logiques sur un circuit : les algorithmes légers.

Ces primitives doivent répondre à des contraintes différentes selon le support d'intégration visé, qui peut être de nature matérielle ou logicielle. Un ensemble de métriques permet de mesurer l'efficacité d'une implémentation.

- Pour une implémentation logicielle, les paramètres à optimiser sont l'empreinte mémoire, la vitesse d'exécution et la latence.
- Pour une implémentation matérielle, un des critères les plus importants est la taille occupée par l'algorithme sur un circuit, qui est mesurée en nombre de portes logiques (ou GE pour *Gate Equivalent*). Un concepteur cherchera également à optimiser le temps d'exécution, la latence ainsi que la puissance requise.

Il est très difficile d'optimiser toutes ces quantités en même temps et en général, l'application visée déterminera les choix d'optimisation.

Une autre problème soulevé est la protection contre les attaques par canaux auxiliaires, qui tirent parti du fonctionnement de l'appareil sur lequel est implémenté l'algorithme de chiffrement visé. Dans un contexte tel que celui de l'Internet des Objets, beaucoup de systèmes (puces RFID, capteurs sans fil etc.) opèrent dans des environnements non-protégés et sont particulièrement exposés à de telles attaques. Ainsi, il est important de pouvoir implémenter des contre-mesures efficaces, tout en conservant de bonnes performances.

Le chapitre détaillera donc les méthodes de conceptions adoptées pour répondre à ces nouvelles exigences, illustrées par plusieurs exemples d'algorithmes légers :

- PRESENT [BKL+07], aujourd'hui standardisé (ISO/IEC 29192-2 [ISO12b]), fut le premier chiffrement léger. Ses concepteurs ont opté pour un ordonnancement de clef simplifié, un état interne réduit, des boîtes-S agissant sur 4 bits et une permutation simple de bits pour favoriser une implémentation matérielle.
- L'implémentation de l'AES sur des petites cartes restant trop coûteuse du fait des structures mathématiques utilisées (boîte-S sur 8 bits, matrice MDS), une question naturelle qui s'est notamment posé a été de savoir comment alléger ces composants, en ayant des propriétés d'implémentation qui soient très bonnes et en gardant une sécurité correcte. Cette approche a été adoptée par les concepteurs de SKINNY [BJK+16]. D'autres travaux ont également été menés sur la recherche de boites-S [Can05, BP09, CDL16, CR19] ou de matrices MDS [JPS17, DL18] compactes.
- Des constructions basées sur des schémas de Feistel, comme LBLOCK [WZ11a] ou LILLIPUT [BFMT16] permettent de diminuer le surcoût du déchiffrement.

Des implémentation logicielles efficaces peuvent être obtenue par *bit-slicing*, qui consiste à réécrire un algorithme cryptographique au plus bas niveau, en utilisant uniquement des

[2]National Institute of Standards and Technology. Il s'agit d'une agence de standardisation américaine.

opérations logiques bit à bit. Les LS-Designs [GLSV15] ont été conçus pour faciliter ce genre d'implémentation. Les structures ARX [Miy91, WN95, BSS+13, DPU+16], n'utilisent que des opérations simples—addition, rotation et XOR et sont également très efficaces pour les applications logicielles.

La question de l'authentification est également importante en cryptographie légère. Plus particulièrement, alors que la méthode traditionnelle consiste à chiffrer les données, puis faire appel à une autre primitive pour générer un *tag d'authentification* permettant de vérifier l'identité de l'émetteur, la communauté cryptographique a cherché à proposer des modes opératoires permettant de fournir l'authentification en même temps que le chiffrement : on parle alors de *chiffrement authentifié* en une *passe*.

Chapitre 3. Cryptanalyse

La cryptanalyse permet d'évaluer la sécurité des primitives cryptographiques. C'est une composante essentielle de la cryptologie; la seule façon de se convaincre de la sécurité d'un algorithme étant de regarder les grandes familles d'attaques connues et de montrer que celles-ci ne s'appliquent pas sur ce système.

Une notion importante en cryptanalyse symétrique est celle du *distingueur*, qui peut être définie comme une propriété permettant à un attaquant de différencier un chiffrement d'une permutation aléatoire.

Prenons comme exemple le cas de la cryptanalyse différentielle, introduite par Biham et Shamir [BS91a], qui étudie le biais statistique induit par une fonction de chiffrement dans la distribution de différences observées en sortie du chiffrement lorsque la différence d'entrée entre deux messages clairs est fixée. L'attaquant considère deux messages d'entrée M, M', dont la différence est fixée à une valeur δ_0, et observe la différence entre les images C, C' de ces deux entrées, notée δ_t, comme illustré sur la Figure 4.

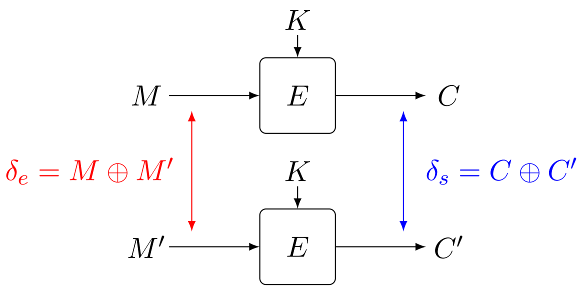


Figure 4: Une differentielle (δ_e, δ_s) sur un chiffrement E, paramétré par une clef K.

Pour une permutation aléatoire, la différence en sortie prendra toutes les valeurs possibles (non-nulles) avec une distribution uniforme. Plus précisément, pour toute paire d'éléments de n bits (δ_e, δ_s) la probabilité :

$$\Pr[f(x \oplus \delta_e) \oplus f(x) = \delta_s].$$

vaudra $\frac{1}{2^n}$ si f est une permutation aléatoire. Cependant, si f correspond à un chiffrement par blocs E, il est possible que pour certains couples de différences d'entrée et de sortie (δ_e, δ_s), cette probabilité soit plus élevée, ce qui donnera un distingueur différentiel. On dit alors que la paire $(\delta_e \rightsquigarrow \delta_s)$ est une *différentielle*. Le but pour un attaquant sera donc de trouver des différentielles de forte probabilité, afin de distinguer le chiffrement d'une permutation aléatoire.

Les analyses présentées dans les Chapitres 5, 6 et 7 se basent sur ce principe. L'attaque différentielle fait partie d'une grande sous-classe d'attaques : les attaques *statistiques*. D'autres attaques de ce type sont décrites dans ce chapitre, comme l'attaque linéaire, l'attaque boomerang ou par interpolation. Une seconde partie sera également consacrée à des attaques dites *structurelles*. La question de l'automatisation de certaines attaques via des outils génériques comme le ILP (Integer Linear Programming), SAT (Boolean Satisfiability problem) ou CP (Constraint Programming) sera évoquée, avant d'être étudiée dans le Chapitre 6.

Partie II : Contributions au processus de standardisation pour la cryptographie légère du NIST

En août 2018, le NIST lança un appel à contribution [NISb] afin de constituer un portfolio d'algorithmes de chiffrement authentifié légers, à l'instar des précédentes compétitions AES en 1998 et SHA-3 en 2007. Parmi les 57 propositions soumises, 56 furent retenues pour le premier tour, en avril 2019. Actuellement, 32 candidats sont encore en lice dans le deuxième tour. C'est dans ce contexte que s'inscrivent les contributions présentées dans cette partie.

Chapitre 4. L'algorithme de chiffrement authentifié léger Lilliput-AE

Ce chapitre présente l'algorithme de chiffrement authentifié léger Lilliput-AE [ABC+18], conçu en collaboration avec Alexandre Adomnicai, Thierry Berger, Christophe Clavier, Julien Francq, Virginie Lallemand, Kévin Le Gouguec, Marine Minier, Léo Reynaud et Gaël Thomas et qui fit partie des candidats du premier tour du processus de standardisation pour la cryptographie légère initié par le NIST.

Ce schéma utilise la primitive Lilliput-TBC, basée sur le chiffrement par blocs léger Lilliput [Tho15, BFMT16]. Il reprend ainsi la fonction de tour, représentée sur la Figure 5. Elle repose sur une structure de Schéma de Feistel Généralisé Étendu (*Extended Generalized Feistel Network* en anglais), ce qui permet d'optimiser son délai de diffusion[3]. L'algorithme de cadencement de clef a en revanche été modifié.

Deux modes de chiffrement authentifiés sont définis : Lilliput-I et Lilliput-II, respectivement basés sur les modes ΘCB3 [KR11b] et SCT-2, tous les deux utilisés dans Deoxys [JNPS16]. Le premier mode garantit une sécurité tant que le nonce est renouvelé entre chaque usage, tandis que le second permet une réutilisation du même nonce. Pour chaque mode, le schéma est défini avec un état interne de 128 bits, ainsi qu'une clef de taille 128, 192 ou 256 bits, conformément aux critères d'évaluation énoncés par le NIST [NISb].

Après avoir donné une spécification complète du schéma, nous proposons une analyse de sécurité justifiant les choix de conceptions. Enfin, le chapitre se conclut par une évaluation des performances logicielles et matérielles. Celles-ci sont comparables à celles d'autre algorithmes légers comme Acorn et Ascon, qui figurent dans le portfolio final de la compétition CAESAR portant sur le chiffrement authentifié.

Chapitre 5. Cryptanalyse de Spook

Spook [BBB+19] est l'un des 32 candidats à avoir été retenu pour le deuxième tour du processus de standardisation pour la cryptographie légère du NIST. Sa permutation interne, `Shadow`, se décline en deux versions : `Shadow-384` et `Shadow-512`, s'appliquant à un état de 384 (respectivement

[3]De manière informelle, la notion de délai de diffusion peut être vue comme le nombre minimal de tours à effectuer pour que tous les blocs en sortie dépendent de tous les blocs en entrée.

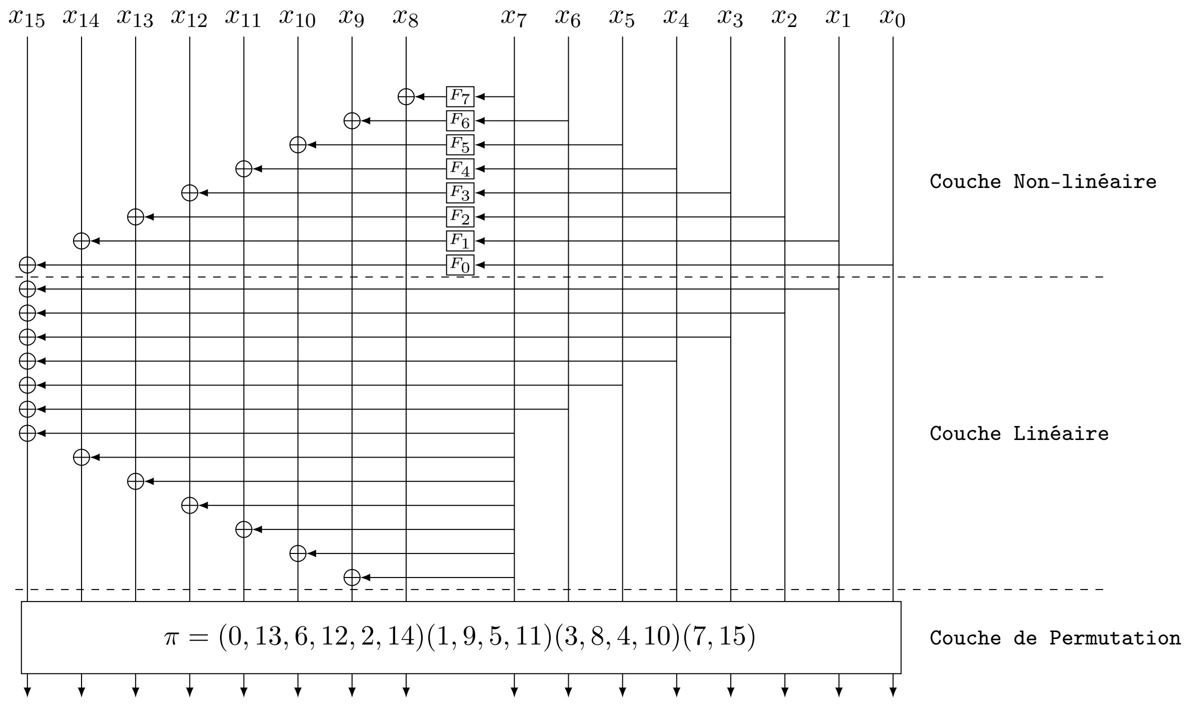

x_{15} x_{14} x_{13} x_{12} x_{11} x_{10} x_9 x_8 x_7 x_6 x_5 x_4 x_3 x_2 x_1 x_0
F_7 F_6 F_5 F_4 F_3 F_2 F_1 F_0
Couche Non-linéaire
Couche Linéaire
$\pi = (0, 13, 6, 12, 2, 14)(1, 9, 5, 11)(3, 8, 4, 10)(7, 15)$
Couche de Permutation

Figure 5: Fonction de tour utilisée dans LILLIPUT-AE.

512) bits vu comme un ensemble de 3 (respectivement 4) tableaux de taille 32 x 4 bits (aussi appelés *bundles*).

Chaque permutation applique 6 étapes (*steps*). Comme le montre la Figure 6, une étape est constituée de deux tours, un tour A et un tour B, séparés par une addition de constante de tour. L'unique différence entre une étape de `Shadow-512` et une étape de `Shadow-384` réside dans la définition de la couche de diffusion D.

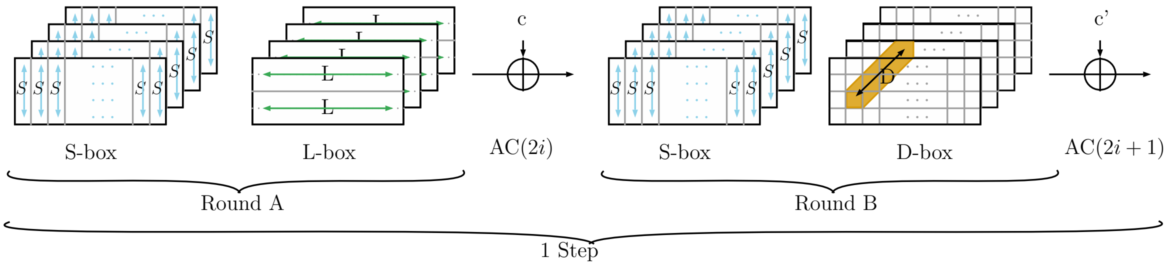


Figure 6: Description d'une étape de `Shadow-512`.

L'analyse présentée dans ce chapitre montre l'existence de distingueurs différentiels sur `Shadow-384` et `Shadow-512`, ainsi qu'une attaque pratique permettant de forger des messages avec un tag d'authentification identique, en utilisant une version réduite de `Shadow`. Ces résultats sont le fruit d'une collaboration avec Patrick Derbez, Virginie Lallemand, María Naya-Plasencia, Léo Perrin et André Schrottenloher, et ont été présentés à Crypto 2020 [DHL+20].

Nos distingueurs reposent sur une propriété structurelle des permutations permettant la préservation après une étape d'états i-identiques, définis comme suit :

Definition 0.1 (État i-identique)**.** *Un état de* **`Shadow`** *est dit i-*identique *si i bundles sont égaux.*

Les équations à satisfaire pour maintenir un état 4-identique après une étape de `Shadow-512` sont détaillées ci-dessous, où S désigne la boîte-S appliquée sur chaque colonne $(y^0, y^1, y^2, y^3)^t$ en début de tour, et c, c' sont les constantes des tours A et B respectivement :

$$\begin{cases} S(y^3 \oplus c) = & S(y^3) \oplus c' \\ S(y^2 \oplus c) = & S(y^2) \oplus c' \\ S(y^1 \oplus c) = & S(y^1) \oplus c' \\ S(y^0 \oplus c) = & S(y^0) \oplus c' \ , \end{cases}$$

ce qui donne les probabilités suivantes :

Table 1: Probabilité que la sortie de l'étape s de `Shadow-512` soit 4-identique sachant que l'entrée l'est.

s	0	1	2	3	4	5
Probability	0	0	0	2^{-12}	2^{-8}	0

Par un raisonnement similaire, il est assez simple de déduire les probabilités que les constantes de tour s'annulent dans le cas d'états 3- et 2-identiques, conservant ainsi ces états après une étape. Ces propriétés permettent d'obtenir un distingueur sur l'intégralité des 6 étapes de `Shadow-512`, ainsi que sur une version réduite à 5 étapes au lieu de 6 pour `Shadow-384`.

En utilisant un principe similaire, il est également possible de générer des collisions sur les tags d'authentifications en très peu de requêtes, dans une configuration de rejeu du nonce. Tous ces résultats ont une complexité très faible et ont été vérifiés expérimentalement. Ils sont disponibles à l'adresse suivante :

https://who.paris.inria.fr/Leo.Perrin/code/spook/index.html

Cette analyse montre l'importance du choix des constantes de tour pour les LS-designs : il faut donc non seulement s'assurer que les attaques par sous-espaces invariants [BCLR17] ne sont pas applicables, mais également éviter d'autres effets indésirables, comme dans ce cas, une annulation dans l'état interne. Nos résultats ont amené les concepteurs de l'algorithme à changer la matrice utilisée dans la couche de diffusion pour la version Spook v2, présentée dans l'édition spécial du journal Transactions of Symmetric Cryptography 2020 consacré à la compétition NIST [BBB+20].

Chapitre 6. Cryptanalyse différentielle automatisée de Skinny

Dans ce chapitre, le chiffrement étudié est Skinny [BJK+16], une primitive sur laquelle sont basées plusieurs propositions d'algorithmes légers soumis au NIST : ForkAE [LPR+19], Remus [IKMP19a], Romulus [IKMP19b] et Skinny-AEAD/Skinny-hash [BJK+19]. Ces travaux ont été effectués en collaboration avec Stéphanie Delaune, Patrick Derbez, Marine Minier, Victor Mollimard et Charles Prud'homme et sont actuellement en cours de soumission.

La recherche de chemins différentiels optimaux est essentielle car elle permet d'évaluer la marge de sécurité des primitive cryptographiques. Son automatisation constitue cependant un problème hautement combinatoire pour lequel le passage à l'échelle s'avère difficile. Une façon de limiter cette explosion combinatoire consiste à utiliser des différentielles tronquées [Knu95], qui indiquent si la différence sur un mot de l'état interne est nulle ou non, sans plus d'information sur la valeur de cette différence. Typiquement, chaque mot de l'état interne est vu comme une variable binaire, et le but recherché n'est plus de déterminer la valeur exacte des différences en entrée et en sortie de chaque tour mais uniquement les positions de ces différences, c'est à dire, déterminer quel mot contient une différence non-nulle. Lorsqu'une différence est présente en entrée d'une boîte-S, on dira alors que cette boîte-S est *active*. L'inconvénient de la représentation tronquée est l'existence de faux-positifs (*i.e.* de chemins non-instanciables).

L'approche usuelle [BN10, AST+17, GLMS20] consiste ainsi à diviser le processus de recherche en deux étapes:

1. une première étape (Step1) qui adopte une représentation abstraite des différences en considérant uniquement des variables binaires et qui cherche donc à trouver des chemins différentiels tronqués minimisant le nombre de boites-S actives;

2. une seconde (Step2) qui essaye d'instancier chaque solution trouvée par l'étape précédente, tout en maximisant la probabilité du chemin.

Plusieurs modèles d'attaques différentielles sont ici considérés : le modèle **SK** se place dans le cadre d'une attaque à clef simple, tandis que le modèle **TK1** (respectivement **TK2** et **TK3**) désigne une attaque à clef liée dans laquelle le tweakey contient une seule (respectivement, deux et trois) composante(s). Le modèle d'attaque à clef liée suppose qu'un adversaire a non seulement le contrôle sur les différences entre les messages clairs M et M' dont il peut demander les chiffrés, mais également sur les différences entre les clés de chiffrement K et K'.

Plusieurs outils automatisés ont été choisis pour modéliser l'étape Step1 : Integer Linear Programming (ILP), Constraint Programming (CP), Boolean Satisfiability Problem (SAT) ainsi qu'un outil Ad-Hoc. Pour l'étape Step2, la programmation par contraintes a été choisie pour son efficacité dans la résolution de ce type de problème.

Ces modèles nous ont permis de retrouver les résultats de [AST+17] en seulement quelques minutes, alors qu'il a fallu environ 16 jours[4] aux auteurs du papier d'origine. De plus, nous avons également pu dériver des bornes optimales jusqu'à 14 tours pour le modèle **TK1** et 12 tours pour le modèle **TK2** de `SKINNY-128`, un nombre de tours encore jamais atteint par les travaux précédents, comme ceux de Liu *et al.* [LGS17]. Enfin, nous montrons l'absence de caractéristique de probabilité supérieure à 2^{-128} pour 15 tours, dans le modèle **TK1**. L'ensemble de nos résultats est résumé dans le Tableau 2.

Ces travaux nous ont également permis de donner une comparaison rapide de différents outils automatisés utilisés. Pour l'étape **Step1**, l'outil Ad-Hoc donne les meilleurs temps dans la plupart de cas et semble donc être la plus efficaces des 4 méthodes étudiées. Cette méthode demande en revanche une grande quantité de mémoire. Aussi, l'approche ILP paraît être une bonne alternative de par sa facilité d'utilisation. Pour l'étape Step2, l'outil CP s'est avéré être le plus rapide, principalement grâce à l'utilisation de contraintes TABLE qui ont permis une modélisation efficace des boîtes-S et de leur DDT[5].

Enfin, nous avons pu constater qu'une meilleure solution de filtrage pour l'étape Step1 serait très bénéfique à l'approche adoptée dans cette analyse. En effet, cette étape renvoie généralement un nombre minimal de boîtes-S actives v^* relativement bas, cependant, les solutions optimales

[4]Il faut néanmoins préciser que nos configurations étaient différentes. Nous aurions souhaité pouvoir faire une comparaison, cependant le code du papier [AST+17] n'est pas publique.

[5]Difference Distribution Table.

	Nb Tours	Obj_{step1}	Nb sol. Step 1	Temps Step 2	Meilleure Pr
SK	9	$41 \to 43$	52	16s	2^{-86}
SK	10	$46 \to 48$	48	11s	2^{-96}
SK	11	$51 \to 52$	15	4s	2^{-104}
SK	12	$55 \to 56$	11	6s	2^{-112}
SK	13	$58 \to 61$	18	2m27s	2^{-123}
SK	14	$61 \to 63$	6	21s	$< 2^{-128}$
TK1	8	$13 \to 16$	14	4s	2^{-33}
TK1	9	$16 \to 20$	6	3s	2^{-41}
TK1	10	$23 \to 27$	6	4s	2^{-55}
TK1	11	$32 \to 36$	531	37s	2^{-74}
TK1	12	$38 \to 46$	186 482	213m	2^{-93}
TK1	13	$41 \to 53$	2 385 482	2 jours	$2^{-106.2}$
TK1	14	$45 \to 59$	11 518 612	20 jours	2^{-120}
TK1	15	$49 \to 63$	7 542 053	25 jours	$< 2^{-128}$
TK2	9	$9 \to 10$	7	3s	2^{-20}
TK2	10	$12 \to 17$	132	11s	$2^{-34.4}$
TK2	11	$16 \to 25$	4203	6m	$2^{-51.4}$
TK2	12	$21 \to 35$	1 922 762	512m	$2^{-70.4}$
TK2	13	$25 \to 44$	-	non résolu	$2^{-89.7}$
TK2	14	$31 \to 54$	-	non résolu	$2^{-108.4}$
TK2	15	$35 \to 56$	-	non résolu	$2^{-113.2}$
TK2	16	$40 \to 63$	-	non résolu	$2^{-127.6}$
TK2	17	$43 \to 63$	-	non résolu	-
TK2	18	$47 \to 63$	62 681 709	non résolu	-
TK2	19	$52 \to 63$	772 163	280m	$< 2^{-128}$

Table 2: Résultats sur `SKINNY-128` dans les 4 modèles d'attaque. "Temps Step 2" correspond au temps pris par Step2 pour toutes les solutions de *Step1-enum* lorsque Obj_{step1} prend la valeur indiquée dans la première colonne. La colonne "Meilleure Pr" indique la meilleure probabilité trouvée parmi les chemins différentiels.

renvoyées par l'étape Step2 en termes de probabilités correspondent parfois à un nombre de boîtes-S actives v beaucoup plus élevé que la valeur v^* trouvée lors de la première étape.

Partie III : Résultats généraux sur les schémas de Feistel

Les chiffrements par blocs peuvent généralement être divisés en deux sous-familles : les schémas de Substitution-Permutation (*Substitution-Permutation Networks* ou SPNs en anglais) et les schémas de Feistel (*Feistel Networks*). Cette partie regroupe des travaux portant sur cette seconde sous-famille.

Chapitre 7. Introduction et analyse de la `FBCT`

En 1999, une nouvelle méthode de cryptanalyse des chiffrements par blocs fut introduite par Wagner : l'*attaque boomerang*. Cette variante de la cryptanalyse différentielle repose également sur l'étude de la propagation de différences dans un chiffrement par blocs, mais pour un ensemble de quatre messages clairs au lieu de deux.

Dans un distingueur basique, illustré sur la partie gauche de la Figure 7, le cryptosystème E est réécrit comme la composition de deux sous-chiffrements ($E = E_1 \circ E_0$), pour lesquels il existe un chemin différentiel allant d'une différence α vers une différence β de forte probabilité $\Pr[\alpha \rightsquigarrow_{E_0} \beta] = p$ et un deuxième chemin différentiel allant d'une différence γ vers une différence δ de forte probabilité $\Pr[\gamma \rightsquigarrow_{E_1} \delta] = q$. Un attaquant cherchera ainsi à maximiser la probabilité d'obtenir l'évènement suivant :

$$E^{-1}(E(M^1) \oplus \delta) \oplus E^{-1}(E(M^1 \oplus \alpha) \oplus \delta) = \alpha.$$

Cette probabilité a dans un premier temps été évaluée à p^2q^2 mais plusieurs résultats ont fini par montrer que cette formule était erronée. En 2010, Dunkelman *et al.* [DKS10], proposent l'*attaque sandwich*, représentée sur la partie droite de la figure 7, dans laquelle le chiffrement E est cette fois divisé en trois parties : un tour de jonction au milieu (*boomerang switch*) est intercalé entre E_0 and E_1.

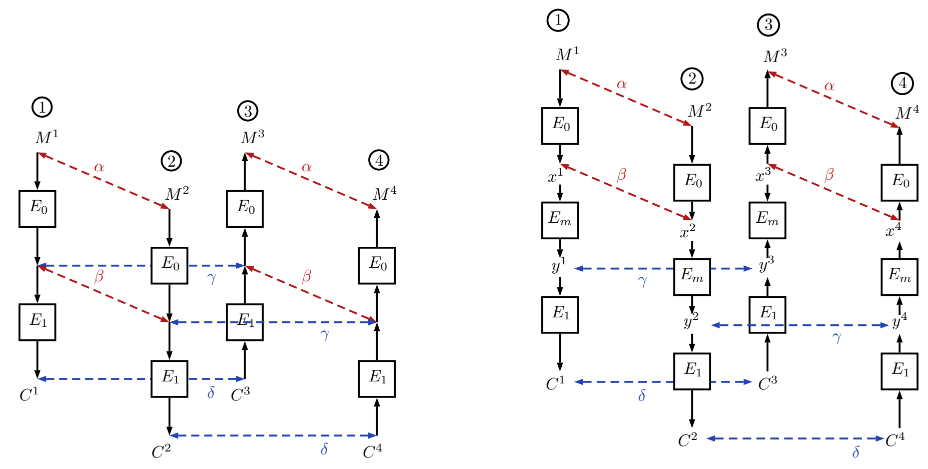


Figure 7: Configurations de l'attaque boomerang dans sa forme de base (gauche) et telle que décrite dans l'attaque sandwich (droite).

En notant r la probabilité que les propagations différentielles requises soient satisfaites pour E_m, la probabilité du distingueur est alors égale à p^2q^2r.

Le calcul de la probabilité r pour les chiffrements SPN a été grandement aidé par l'introduction en 2018 par Cid *et. al* de la *Boomerang Connectivity Table* (BCT) [CHP+18].

Definition 0.2 (Boomerang Connectivity Table [CHP+18])**.** *Soit S une permutation de $\mathbb{F}_2^n$, et $\Delta_i, \nabla_o \in \mathbb{F}_2^n$. La Boomerang Connectivity Table (BCT) de S est donnée par une table de taille $2^n \times 2^n$ dans laquelle l'entrée à la position (Δ_i, ∇_o) est donnée par :*

$$BCT(\Delta_i, \nabla_o) = \#\{x \in \mathbb{F}_2^n | S^{-1}(S(x) \oplus \nabla_o) \oplus S^{-1}(S(x \oplus \Delta_i) \oplus \nabla_o) = \Delta_i\}.$$

Cette nouvelle table a permis de calculer de façon systématique la probabilité r du tour du milieu d'un distingueur pour une attaque boomerang en ramenant le problème à un calcul de probabilité sur les boîtes-S du chiffrement. Cependant, cette étude ne s'est intéressée qu'aux constructions SPN, délaissant les schémas de type Feistel. Le travail présenté dans ce chapitre permet de combler ce manque, en introduisant la FBCT d'une boite-S S, l'équivalent de la BCT pour les constructions de type Feistel. Il a été effectué avec Hamid Boukerrou, Virginie Lallemand, Bimal Mandal et Marine Minier, et nos résultats ont été présentés à FSE 2020 [BHL+20].

La FBCT peut être définie comme suit :

Definition 0.3 (FBCT). *Soit S une fonction de $\mathbb{F}_2^n$ dans $\mathbb{F}_2^m$, et $\Delta_i, \nabla_o \in \mathbb{F}_2^n$. La FBCT de S est une table de taille $2^n \times 2^n$ dans laquelle l'entrée à la position (Δ_i, ∇_o) est donnée par :*

$$\mathit{FBCT}_S(\Delta_i, \nabla_o) = \#\{x \in \mathbb{F}_2^n | S(x) \oplus S(x \oplus \Delta_i) \oplus S(x \oplus \nabla_o) \oplus S(x \oplus \Delta_i \oplus \nabla_o) = 0\}.$$

Il est intéressant de remarquer que la valeur de la table à la position (Δ_i, ∇_o) correspond au nombre d'annulations possibles de la dérivée seconde de S en les points Δ_i, ∇_o. Nous pouvons également noter que contrairement à la définition de la BCT, S n'a pas besoin d'être une permutation. Ceci s'explique par le fait que, dans un structure de type Feistel, le déchiffrement correspond au chiffrement, à une inversion des sous clefs près. Ainsi, l'inverse de la fonction de tour n'intervient à aucun moment.

Nous avons également regardé quelles étaient les propriétés de la FBCT. Le tableau 3 compare certaines propriétés étudiées par Boura and Canteaut [BC18] sur la BCT avec le cas de la FBCT. Dans le cadre des attaques boomerang, le comportement d'une boîte-S peut donc différer en fonction du type de construction dans laquelle elle est utilisée.

Table 3: Comparaison des propriétés de la BCT et de la FBCT (fonctions sur n bits).

Propriété	BCT	FBCT
Uniformité boomerang préservée par équivalence affine	oui	oui
Uniformité boomerang préservée par équivalence affine-étendue	non	oui
Uniformité boomerang préservée par équivalence CCZ	non	non
Uniformité boomerang préservée par inversion	oui	non
Valeur de l'uniformité boomerang d'une fonction APN	2	0
Valeur de l'uniformité boomerang pour la fonction inverse (n pair)	4 ou 6	4

Enfin, nous proposons dans la dernière section du chapitre une formule générique permettant de calculer la probabilité pour une section E_m couvrant un nombre arbitraire de tours, dans le cas où la caractéristique d'arrivée est la même que la caractéristique de départ.

Chapitre 8 : Recherche de structures d'EGFN légères dérivées de la fonction de tour de Lilliput

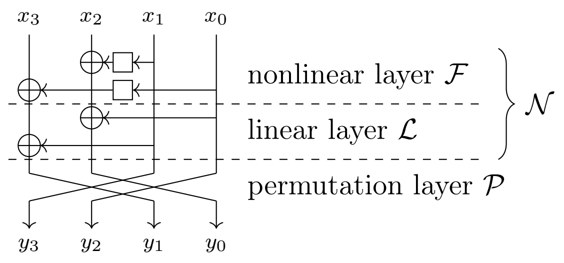
Lors de la conception de Lilliput-AE, différentes structures de type EGFN ont été étudiées, l'idée étant de partir de la fonction de tour du chiffrement Lilliput [BFMT16] et d'alléger cette structure tout en conservant un délai de diffusion optimal ainsi qu'une résistance suffisante à la cryptanalyse différentielle et intégrale. Ce chapitre regroupe les observations que nous avons pu faire lors de nos expériences.

Pour cette étude, l'approche matricielle développée dans [BMT14, Tho15] et rappelée dans la Définition 0.4, a été adoptée. Cette représentation permet de voir la fonction de tour comme étant le produit de deux matrices, correspondant à une couche non-linéaire et à une couche de permutation.

Definition 0.4. *Une matrice $\mathcal{M}$ à coefficients dans $\{0, 1, F, I\} \subset \mathbb{Z}[F, I]$ est la matrice d'un schéma de Feistel généralisé étendu quasi-involutif s'il existe une matrice de permutation $\mathcal{P}$ et une matrice $\mathcal{N}$ avec $\mathcal{M} = \mathcal{P}\mathcal{N}$ tel que :*

1. *la diagonale est remplie de un,*
2. *les coefficients non-nuls en dehors de la diagonale sont soit F, soit I,*
3. *pour tout $0 \leq i \leq k-1$, la ligne i et la colonne i ne peuvent pas toutes deux contenir un coefficient non-nul en dehors de la diagonale,*
4. *pour tout $0 \leq i \leq k-1$, si la ligne i contient un I alors elle contient aussi un F.*

Un exemple de EGFN avec $k = 4$ branches et ses matrices associées sont représentées sur la Figure 8.


x_3
x_2
x_1
x_0
nonlinear layer $\mathcal{F}$
linear layer $\mathcal{L}$
$\mathcal{N}$
permutation layer $\mathcal{P}$
y_3
y_2
y_1
y_0

$$\mathcal{M} = \begin{pmatrix} I & F & 1 & \\ F & I & & 1 \\ 1 & & & \\ & 1 & & \end{pmatrix} \quad \mathcal{P} = \begin{pmatrix} & & 1 & \\ & & & 1 \\ 1 & & & \\ & 1 & & \end{pmatrix}$$

$$\mathcal{L} = \begin{pmatrix} 1 & & & \\ & 1 & & \\ I & & 1 & \\ & I & & 1 \end{pmatrix} \quad \mathcal{F} = \begin{pmatrix} 1 & & & \\ & 1 & & \\ & F & 1 & \\ F & & & 1 \end{pmatrix}$$

Figure 8: Exemple d'EGFN et les matrices associées.

Nous sommes partis du constat que le Théorème 0.5, énoncé ci-dessous— et vérifié par la fonction de tour du chiffrement LILLIPUT—reste vrai tant que la dernière ligne et la dernière colonne contiennent uniquement des coefficients qui sont non-nuls. Cette observation a laissé entrevoir la possibilité d'obtenir des variantes plus légères de LILLIPUT (avec, au maximum, une réduction de 6 coefficients), tout en garantissant un délai de diffusion optimal. Ainsi, nous avons cherché à identifier toutes les structures satisfaisant les conditions du Théorème 0.5 présentant de bonnes propriétés vis à vis des attaques différentielles et intégrales.

Theorem 0.5. *Pour un entier k pair, $k \geq 4$, soit un EGFN tel que défini dans la Définition 0.4 à k branches, $\mathcal{M}$, $\mathcal{N}$ et $\mathcal{P}$ ses matrices associées, où $\mathcal{N}$ est la représentation matricielle des couches linéaire et non-linéaire de l'EGFN et $\mathcal{P}$ la matrice de la couche de permutation, correspondant à une permutation π,*

1. *si la matrice $\mathcal{N}$ est égal à $\mathcal{N} = \left(\begin{smallmatrix} \mathcal{I} & 0 \\ \mathcal{A} & \mathcal{I} \end{smallmatrix}\right) \in \mathbb{Z}[F, I]$ où $\mathcal{I}$ est la matrice identité de taille $\frac{k}{2} \times \frac{k}{2}$ et $\mathcal{A}$, le quart inférieur gauche de $\mathcal{N}$, est égal à $\mathcal{A} = \begin{pmatrix} & (0) & & F & \begin{smallmatrix} F \\ I \end{smallmatrix} \\ & & \cdot^{\cdot^{\cdot}} & & I \\ & F & & (0) & \vdots \\ F & I & I & \cdots & I \end{pmatrix}$*
2. *et si $\mathcal{P}$ échange globalement les émetteurs (branches x_0 à $x_{k/2-1}$) et les receveurs (branches $x_{k/2}$ à x_{k-1}) avec $\pi(k/2-1) = k-1$ et $\pi(k-1) = k/2-1$,*

alors le délai de diffusion d est égal à 4.

Plus particulièrement, nous nous sommes restreints à l'ensemble des permutations involutives vérifiant la Condition 2 du Théorème 0.5, noté $\mathcal{P}_{invol}$, afin de minimiser le surcoût de l'opération de déchiffrement. Pour toutes les configurations possibles de $\mathcal{A}$ et toutes les matrices de $\mathcal{P}_{invol}$ satisfaisant le Théorème 0.5, nous avons donc regardé les propriétés intégrales et différentielles des schémas $\mathcal{M}$ correspondants, avec pour objectif de réussir à caractériser les permutations les plus intéressantes et ainsi définir des critères sur le choix des permutations.

Introduction

Karma police
Arrest this man
He talks in maths

Radiohead

This thesis covers the research work in the field of symmetric cryptography that I carried out between 2017 and 2020 at Loria, Nancy, where I was a Ph.D. student in the Caramba team, under the supervision of Marine Minier. The main focus of my research was lightweight cryptography—that is, cryptography intended to be used on resource-constrained devices, such as wireless sensors—from a design and a cryptanalysis standpoint. The content of this thesis is split into three parts: after a first introductory part, Part II and Part III describe my contributions.

Part I: Background and Preliminaries

This first part aims at giving the reader a grasp of the fundamental concepts of modern cryptology.

Chapter 1. This chapter recalls the main goals of this research field and reports how these objectives are being pursued in symmetric cryptography. More specifically, it delves into the design of block ciphers, one of the two major classes of cryptosystems that can be found in this field.

Chapter 2. This chapter discusses a new research direction in symmetric cryptography that has emerged in the recent years: lightweight cryptography. It investigates the challenges faced in the context of the Internet of Things (IoT) and presents some of the design trends introduced to address these constraints and the algorithms that have resulted from these efforts, the several competitions and standardizations attempts, such as the one initiated by the National Institute of Standards and Technology (NIST).

Chapter 3. Cryptanalysis is a field of cryptology dedicated to the security evaluation of cryptographic primitives. Even though designers are expected to make sound security analyses, external cryptanalysis is instrumental for the adoption of any new proposal. A primitive that has been subject to many third-party analysis and for which no security breach has been found is likely to gain trust from the cryptographic community. This chapter offers an overview of some of the most prominent cryptanalysis techniques used to study encryption schemes and introduces the basic notions that will later be used in Chapter 5, Chapter 6 and Chapter 7.

Part II: Contributions to the NIST Lightweight Cryptography Standardization Process

The second part of this thesis regroups all the works related to the NIST Lightweight Cryptography Standardization Process.

Chapter 4. This chapter presents LILLIPUT-AE, a new Authenticated Encryption with Associated Data (AEAD) scheme designed in collaboration with Alexandre Adomnicai, Thierry Berger, Christophe Clavier, Julien Francq, Virginie Lallemand, Kévin Le Gouguec, Marine Minier, Léo Reynaud and Gaël Thomas as a proposal to the NIST Lightweight Cryptography Standardization Process. After describing this algorithm based on an Extended Generalized Feistel Network (EGFN), a security analysis is provided to support our design choices. An evaluation of the performances of LILLIPUT-AE for both software and hardware platforms is discussed at the end of the chapter.

Chapter 5. SPOOK is one of the 32 candidates that has made it to the second round of the NIST LWC standardization process. This chapter demonstrates a distinguisher on the full underlying permutation of SPOOK, `Shadow-512`, and a practical forgery attack against SPOOK in the nonce-misuse setting, with a reduced version of `Shadow`.

Chapter 6. SKINNY serves as a basis for several candidates of the NIST LWC standardization process. This chapter compares existing automatic tools—Integer Linear Programming (ILP), Constraint Programming (CP) and Boolean Satisfiability Problem (SAT)—to find the best differential characteristics on SKINNY, a task that is decisive when it comes to evaluating the security margin of cryptographic primitives.

Part III: General Results on Feistel Constructions

This part focuses on Feistel schemes which, together with Substitution-Permutation Networks (SPNs), are the main constructions used for block ciphers.

Chapter 7. In this chapter a new tool to compute the probability of the middle round of a boomerang distinguisher for ciphers following a Feistel construction is introduced: the FBCT. Its properties are studied and compared to the BCT, its counterpart for SPNs. A generic formula to compute the probability of a boomerang switch over multiple rounds is also provided. Finally, this approach is applied to LBLOCK-S, the lightweight block cipher used in the CAESAR candidate LAC, to build a 16-round distinguisher.

Chapter 8. When designing LILLIPUT-AE, several configurations of Extended Generalized Feistel Networks (EGFNs) were explored, inspired by the one used in the round function of the LILLIPUT block cipher. The aim was to find lighter constructions that provided good resistance to differential and integral cryptanalysis, while preserving an optimal diffusion delay. This chapter recalls some properties of Generalized Feistel Networks (GFNs) and their extensions, then summarizes small results derived from the experiments that were conducted.

List of Publications

1. Lilliput-AE: a New Lightweight Tweakable Block Cipher for Authenticated Encryption with Associated Data
Alexandre Adomnicai, Thierry P. Berger, Christophe Clavier, Julien Francq, Paul Huynh, Virginie Lallemand, Kévin Le Gouguec, Marine Minier, Léo Reynaud, and Gaël Thomas.
Submission to the NIST Lightweight Cryptography project. Available online `https://csrc.nist.gov/CSRC/media/Projects/Lightweight-Cryptography/documents/round-1/spec-doc/LILLIPUT-AE-spec.pdf`, 2018.

2. On the Feistel Counterpart of the Boomerang Connectivity Table–Introduction and analysis of the FBCT.
Hamid Boukerrou, Paul Huynh, Virginie Lallemand, Bimal Mandal, and Marine Minier.
IACR Trans. Symmetric Cryptol., 2020(1):331–362, 2020. `doi:10.13154/tosc.v2020.i1.331-362`.

3. Cryptanalysis Results on Spook–Bringing Full-round Shadow-512 to the Light.
Patrick Derbez, Paul Huynh, Virginie Lallemand, María Naya-Plasencia, Léo Perrin, and André Schrottenloher.
In Daniele Micciancio and Thomas Ristenpart, editors, *Advances in Cryptology - CRYPTO 2020 - 40th Annual International Cryptology Conference, CRYPTO 2020, Santa Barbara, CA, USA, August 17-21, 2020, Proceedings, Part III*, volume 12172 of *Lecture Notes in Computer Science*, pages 359–388. Springer, 2020. `doi:10.1007/978-3-030-56877-1_13`.

4. Skinny with Scalpel: Comparing Tools for Differential Cryptanalysis.
Stéphanie Delaune, Patrick Derbez, Paul Huynh, Marine Minier, Victor Mollimard, and Charles Prud'homme. (submitted)

List of Figures

Part I

Background and Preliminaries

1

Design of Symmetric Encryption Algorithms

What immortal hand or eye,
Dare frame thy fearful symmetry?

William Blake

Cryptology is the science that studies secret communication. It encompasses both *cryptography* and *cryptanalysis*. Cryptography studies the design of algorithms to ensure secure communication in the presence of malevolent third parties. Cryptanalysis is the offensive side, trying to find flaws in the aforementioned designs. The evolution of cryptology has always been driven forward through a sort of cat-and-mouse game: each time an algorithm is compromised—or *broken*—by cryptanalysts, a new one which is computationally more challenging to break is put forward by cryptographers.

1.1 From Ancient to Modern Cryptography

Ciphers have been used since ancient times, allowing people to transmit and protect sensitive information. A cipher (or *cryptosystem*) defines the set of rules used for *encryption*—the process of converting an original message (*plaintext*) into an unreadable form (*ciphertext*)—and for *decryption*—the process of recovering the plaintext from the ciphertext.

Ancient cryptography. Some of the earliest forms of secret writing date back to antiquity [Kah96]. Although occurrences of symbol replacement can be traced back to 1900 BC, in ancient Egypt, the first known example of cryptography being used to protect sensitive information occurred around 3500 years ago, when a Mesopotamian scribe concealed a formula for pottery glaze by omitting letters and changing the spellings of some common words.

By later periods of antiquity, cryptography was widely used to protect important military information. In ancient Greece, during times of battle, the Spartans used *scytales* to perform an early form of a *transposition cipher*. Messages were written on a strip of leather wound around a cylinder of a particular size, making them undecipherable until they were wrapped around a similar-sized cylinder by the recipients.

Substitution ciphers. A more advanced cryptography method was invented by the Romans around 60 BC, known as the *Caesar cipher*, where letters of the encrypted message were shifted by 3 places down the Latin alphabet, turning A into D, B into E, etc. The Caesar cipher is the most famous example of a larger class of techniques called *substitution ciphers*, which replace every letter in a message with something else according to a translation. More specifically, a Caesar cipher is a *monoalphabetic substitution cipher*, a class of ciphers for which each letter of the alphabet stands for another letter consistently throughout the whole message. A variant of the Caesar cipher is still nowadays widely found on Internet forums to obscure jokes or story spoilers: the ROT13 algorithm. It replaces each of the 26 letters of the basic Latin alphabet, with the letter positioned 13 places down and it is therefore its own inverse.

After the Caesar cipher, there were no major breakthroughs in encryption for over a thousand years but major contributions in cryptanalysis were made in the meantime. Around 800 AD, mathematician Al-Kindi wrote the first text on cryptanalysis and introduced the *frequency analysis* technique, determining which letters occurred most frequently in Arabic, as well as which letters could not occur together. In *monoalphabetic substitution ciphers*, such properties of the natural language are preserved in the ciphertext, allowing an attacker to determine the appropriate shift and recover the original message.

This technique was rendered ineffective after the invention of *polyalphabetic substitution ciphers*. One of the earliest polyalphabetic ciphers was the *Vigenère cipher*, developed in the 16th century. This cipher used a shifting substitution determined by a keyword repeated multiple times, spanning the entire message.

Kerckhoffs' principle. The advent of the telegraph in the 19th century motivated the use of more secure ciphers. In 1883, Dutch cryptographer Auguste Kerckhoffs laid out a series of six design principles for military ciphers in his essay titled *La Cryptographie Militaire* ("Military Cryptography") [Ker83]. The second one is known as *Kerckhoffs' principle* and states the following about a cryptosystem:

> *"2. Il faut qu'il n'exige pas le secret, et qu'il puisse sans inconvénient tomber entre les mains de l'ennemi ;"*

which can be translated to

> *"2. It must not require secrecy and should not cause any inconvenience if it were to fall into the hands of the enemy;"*

Central to Kerckhoffs' principle is the idea that secrecy in itself is no effective guarantee of security. This fundamental design guideline gave rise to modern cryptography algorithms.

World War II. By the 1900s, cryptography was mechanized in the form of encryption machines. In 1918, Arthur Scherbius invented one of the most famous cryptographic machines in the world: *Enigma*, an encryption device heavily used by the Axis powers during World War II with about 2^{67} possible keys defined by the settings of its internal rotors.

Polish mathematicians led by Marian Rejewski reverse-engineered the details of Enigma and found the first attacks against it using an electromechanical device called the *bomba kryptologiczna* ("cryptologic bomb"). This early computer technology was later transferred to British mathematician Alan Turing along with the rest of his team at Bletchley Park who succeeded in breaking more recent versions of Enigma.

The modern era. In 1949, Claude Shannon authored a landmark paper [Sha49] that established the mathematical basis of information theory and formalized the main goals of modern cryptography.

Notably, this paper introduced the concept of *perfect secrecy* (also referred to as *unconditional security*). A ciphertext maintains perfect secrecy if the attacker's knowledge of the contents of the message is the same both before and after the adversary inspects the ciphertext, attacking it with unlimited resources. That is, the message gives the adversary precisely no information about the message contents. To achieve unconditional security, the key must be at least the size of the message. This is the idea behind the *One-Time Pad*—also known as the *Vernam cipher*—an encryption algorithm invented in the 20th century that simply consists in XORing[1] every plaintext with a *single-use* key of the same length. The One-Time Pad provides perfect secrecy if the keys used are fully random, as proven by Shannon, however, this cipher is impractical due to the key management problem.

In addition to his work on perfect secrecy, Shannon also identified the principles of *confusion* and *diffusion* as two major properties of a secure cipher. Confusion means that the key does not relate in a simple way to the ciphertext and can be enforced using substitutions.Diffusion is enforced using permutations or transpositions and spreads the plaintext statistics through the ciphertext. Simply put, if one character of the plaintext is changed, then the ciphertext should change drastically.

Towards civilian applications. Up to World War II, encryption was mostly exploited for military purposes. However, with the advent of the personal computer, cryptography started to gain commercial attention. Businesses trying to secure their data from competitors as well as the development of the Internet and the massive amount of shared information all called for a widespread use of cryptography. This gave rise to modern cryptographic algorithms like DES [DES77] in 1975, RSA [RSA78] in 1977, AES [AES01, DR02] in 2001.

1.2 General Concepts of Cryptography

1.2.1 Purpose

The narrative in cryptography is simple: two entities, Alice and Bob, are trying to communicate securely over an insecure channel. Eve is an *attacker* trying to interact malevolently with the communication. Modern cryptography is meant to provide the following security services:

[1] In Boolean algebra, the exclusive or is a logical operation that outputs true (1) only when inputs differ *i.e.* one is true (1) and the other is false (0).

- *Secrecy.* Only authorized parties—Alice and Bob here—can access the content of the information. It must be impossible for Eve to obtain any meaningful information from intercepted communication.
- *Integrity.* Data integrity provides a means to detect whether data has been tampered with by an unauthorized party. If Alice is sending a message to Bob, it must be impossible for Eve to change its content.
- *Authenticity.* The aim here is twofolds. If Bob receives a message from Alice, he must be able to tell if Alice was indeed the original sender. Moreover, it must be impossible for Eve to communicate with Bob while pretending to be Alice.

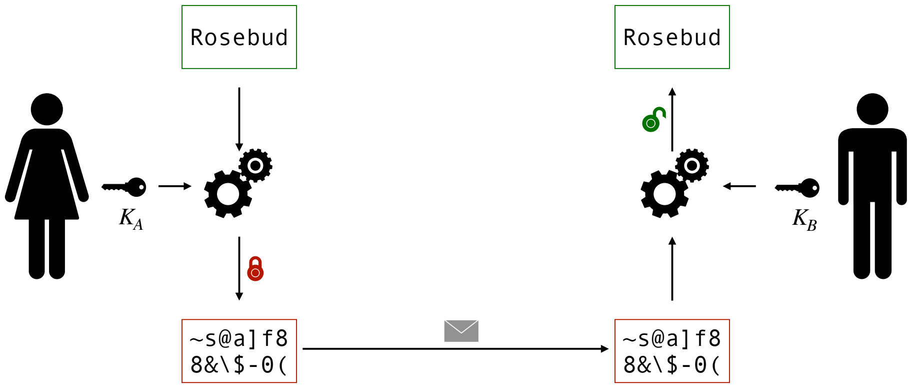


Figure 1.1: A classic setting in cryptography: Alice wants to send a message to Bob. In symmetric cryptography, Alice and Bob share the same secret key, *i.e.* $K_A = K_B$.

Modern cryptography can be divided into two main branches: *private-key* (or *symmetric*) cryptography and *public-key* (or *asymmetric*) cryptography.

In symmetric cryptography, a *unique* key is used for both encryption and decryption. Symmetric cryptography also studies algorithms that do not require any key at all, such as *hash functions*.

Asymmetric cryptography requires a *pair of keys*: one key set up in advance is used to encrypt and another one to decrypt. The key for encrypting plaintexts, called the *public key*, can be freely distributed while only the decryption key, called the *private key*, is kept secret. In other words, the sender and the receiver do not need to share a common secret in order to communicate securely. This solves the complex key sharing problem of symmetric algorithms, which are in contrast faster and more power-efficient than public-key algorithms.

As a result, a combination of both is most commonly used in practice: a secret key is first exchanged using public-key cryptography, then communication is encrypted with a symmetric algorithm. This thesis focuses on the design and the analysis of symmetric encryption primitives.

1.2.2 Symmetric Cryptography

In symmetric cryptography, the studied algorithms can be divided into four categories: *block ciphers*, *stream ciphers*, *hash functions* and *message authentication codes*. Block ciphers and stream ciphers provide secrecy while hash functions and message authentication codes—which will not be discussed in this thesis—are used for integrity and authentication.

Stream ciphers. A *stream cipher* uses the same encryption principle as the One-Time Pad—messages are XORed with a sequence of bits of the same length—while offering more practical requirements on key management: the stream of bits, called the *keystream*, is generated in a *pseudorandom*[2] manner from a shorter secret key that needs to be transmitted only once. However, since the keystream is not truly random, the proof of security associated with the one-time pad no longer holds.

A stream cipher can be seen as a finite state automaton whose initial internal state is derived from the short secret key. At each clock cycle, the keystream bits are obtained by applying a *filtering function* to the internal state before it is updated using a *transition function*. Both filtering function and transition function must be chosen carefully.

Linear-Feedback Shift Registers (LFSR) are very popular building-blocks for transition functions since they can generate sequences with good statistical properties at high speed and have low hardware implementation cost. However, schemes based on a simple combination of LFSRs such as A5/1—a stream cipher used to encrypt over-the-air transmissions in the GSM standard—are insecure. Most notably, an LFSR should never be used by itself as a keystream generator since the *Berlekamp-Massey algorithm* [Mas69] can recover the entire keystream given enough consecutive bits of it. Moreover, LFSR-based keystream generators can be vulnerable to *algebraic attacks* [CM03] due to the linearity of the transition function. This weakness can be avoided by combining LFSRs with nonlinear mappings such as Nonlinear-Feedback Shift Registers (NLFSRs) or more sophisticated filtering functions.

Stream ciphers are well-suited for software applications requiring a very high throughput or with limited buffering, since each bit of a stream of data is encrypted individually. They also have lower hardware complexity than block ciphers, which makes them interesting for hardware applications that need a low-cost implementation, as illustrated by Trivium [De 06] and Grain [HJMM08] for instance.

Block ciphers. While a stream cipher encrypts one bit at a time, a block cipher is a family of permutations operating on blocks of n bits (usually 64/128/256 bits) that is parameterized by a key. Block ciphers must be combined with *modes of operation* which describe how to repeatedly apply a single-block operation to encrypt plaintexts of arbitrary size. Further details will be provided in Section 1.3.

Hash functions. A cryptographic hash function takes input strings of arbitrary length and maps these to short fixed length output strings called *hash values* or *digests*. This construction requires no secret key and must satisfy the following properties.

- *Collision resistance*: it should be computationally infeasible to build two messages with the same hash value.
- *Preimage resistance*: it should be computationally infeasible to reverse a hash function (i.e. given a hash value, it should be difficult to find an input message resulting in that value).
- *Second-preimage resistance*: given an input and its digest it should be difficult to find a second input with the same digest.

[2] A pseudorandom number generator (PRNG) is an algorithm that can generate predictable sequences of numbers that exhibit statistical randomness.

The most common cryptographic uses of hash functions are in digital signatures and data integrity verification. To sign a message, a hash function is applied and only the digest is signed, which brings both performance and security benefits. The recipient then hashes the received message, and verifies that the received signature is valid for this newly-computed hash-value. For data integrity verification, a document is hashed at two different points in time—before and after being sent for instance—and the corresponding digests are compared to detect any alteration that may have occurred within this timeframe.

Message authentication codes. A message authentication code (MAC) algorithm is a family of functions parameterized by a secret key that generates a *tag* of fixed length from an arbitrarily long message, which is used to verify the authenticity and the integrity of the message. MACs can be seen as keyed hash functions. The main security requirement for a MAC algorithm is the *computation-resistance* property; that is, it should be computationally infeasible for an attacker to forge a message and tag pair without knowing the secret key.

1.2.3 Security and adversarial models

Following Kerckhoffs' principle, most cryptographic algorithms are published in the open literature, exclusively relying on the secrecy of the key to provide security. As a result, any attacker can access the full specification of an algorithm and study it to provide a security assessment. The more an algorithm has resisted cryptanalysis attempts, the more it is trusted by the community. A description of the main cryptanalytic attacks is given in Chapter 3.

Security level. The *security level* of cryptosystem is defined as the minimum number of operations that would be necessary to break the cipher. It is usually expressed in bits: a cryptosystem offers i-bit security and is said to have security level i if the best attack requires 2^i operations. Cryptographic primitives are designed to achieve a *target security level.* When attacks are found that have lower cost than these *security claims*, the primitive is considered broken.

The security level of an algorithm is highly correlated to the key size. More precisely, since the security of any algorithm can be violated by *brute-force attack* (also called *exhaustive key search*), the key length gives an upper bound: for a k-bit key, enumerating all possible key values until the correct one is found requires up to 2^k operations. At the time of writing, it is regarded that 2^{80} operations is an affordable figure for a handful of actors with major computing resources and a large budget, such as government agencies. Consequently, the current recommendations suggest a minimum of 128 bits for the key size of symmetric ciphers [ANS14, ECR18, Bar20], as this provides *computational security*: no current technology could break the system[3].

In asymmetric cryptography on the other hand, the designs rely on mathematical problems that are known to be easy to compute in one direction but hard to reverse, such as *integer factorization* or the *discrete logarithm problem*; solving these underlying arithmetic problems is time-consuming but usually faster than exhaustive search of the keyspace. Thus, asymmetric keys must be longer than symmetric algorithm keys, for an equivalent security level. As an example, a 2048-bit minimum is recommended for the key size of the RSA cryptosystem [RSA78], which is based on the multiplication of two prime numbers. Breaking this primitive can be reduced to factoring a 2048-bit integer, which is simpler than testing all 2^{2048} possibilites, although still computationally unreachable in reasonable time to this day. The complexity of solving such problems is subject to many refinements and has generated much interest [Jou14, BGJT14, KW19, BGG+20].

[3]However, it might become vulnerable in the future.

Attack models. In general terms, an *attack* is a method that provides some information about the decryption key (*key recovery attack*) and/or the plaintext (*decryption attack*). There are four main types of attacks, depending on the information made available to the cryptanalyst.

- *Ciphertext-only attack.* The attacker only has access to a set of ciphertexts produced using the same encryption key. This is the weakest assumption and thus, the hardest setting for the attacker.
- *Known-plaintext attack.* The attacker has access to several pairs plaintexts together with their corresponding ciphertexts produced using the same encryption key.
- *Chosen-plaintext attack.* The attacker can choose the plaintexts to be encrypted and obtain the corresponding ciphertexts.
- *Chosen-ciphertext attack.* The attacker can choose ciphertexts and obtain their decryption.

Usually the attacker has no information on the encryption key and cannot interact with it. However, in the *related-key* setting, the attacker can choose the plaintexts (or the ciphertext) to be encrypted (or to be decrypted) under multiple keys that are kept secret, yet linked to the target key in some mathematically defined way that is known. For instance, for two keys k and k', the attacker might know the value $c = k \oplus k'$. Although less realistic at first glance, this setting is still plausible in the context of block-cipher-based hash functions.

All these *cryptanalytic attacks* use a *black-box* model: the attacker cannot access the internal state of the cipher; only its input, output or key can be controlled. On the other hand, *side-channel attacks* can gain partial information about the internal state of a primitive by exploiting the physical effects caused by the operation of a cryptosystem, such as variations in the power consumption.

1.3 On the Design of Block Ciphers

Block ciphers are some of the most prominent components in symmetric cryptography. A block cipher E is an algorithm parameterized by a key K of k bits acting on plaintext blocks of n bits and producing ciphertext blocks of the same size. The block size n usually equals 64, 128 or 256 bits. Formally, a block cipher can be defined as follows.

Definition 1.1 (Block cipher). *A block cipher E taking as inputs an n-bit block and a k-bit key is a family of permutations of $\mathbb{F}_2^n$ of size 2^k denoted by $(E_K)_{K \in \mathbb{F}_2^k}$.*

In other words, given a block cipher E, any secret key $K \in \mathbb{F}_2^k$ defines a unique permutation on n bits denoted by E_K.

Ideal block cipher. One of the desired properties of a block cipher is that it needs to be a reversible mapping for decryption to be possible, meaning that each input block is mapped to a unique output block, for all possible input states. Ideally, a block cipher would allow every possible permutation of the plaintext values, with each permutation being selected by an encryption key K. For an n-bit block, there are $2^n!$ such transformations in total. Since the key specifies which mapping to use, the key space should be as big as the number of possibilities, resulting in a key that would be $log_2(2^n!)$ bits long. This size is absolutely not practical as it would be impossible to store and to distribute in an efficient manner. As an example, for $n = 32$, it would require approximately 16 GB to store the key. In practice, the key size k is close to the block size n and never exceeds a couple of hundreds of bits.

1.3.1 Modes of operation

A block cipher only acts on blocks of fixed length n, but messages come in a variety of lengths. To operate on an input which is larger, the data is first split into separate n-bit blocks. Whenever required, the message is *padded*: a specific sequence of bits is added to it in order to bring its length up to a multiple of the block size n. *Padding* thus ensures that messages of arbitrary length can be partitioned into blocks of the right size. These blocks are then processed by the block cipher and combined together to form the final output in a specific manner that is described by a mode of operation.

Ecb (Electronic Code Book). This mode is very simple: an $\ell \times n$-bit plaintext M is broken into ℓ n-bit blocks $M_0, \cdots, M_{\ell-1}$ and each block is encrypted independently at a time. The ciphertext blocks are then concatenated together to obtain the resulting ciphertext $C = C_0||C_1||\cdots||C_{\ell-1}$, as depicted in Section 1.3.1. While ECB is parallelizable in both encryption and decryption, this mode does not hide patterns in the plaintext: for instance, two identical blocks will result in the same ciphertext block, thus revealing information on the inner structure of the original message. Figure 1.3 shows that even after encryption, the outlines of the content of a picture remain visible.

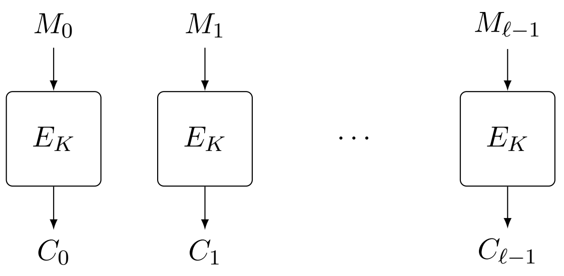
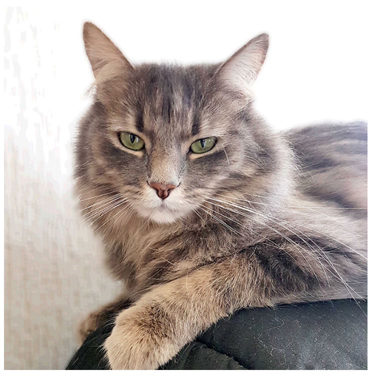
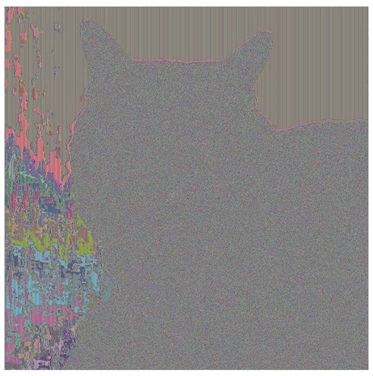


Figure 1.2: Encryption in the ECB mode of operation.



(a) A picture of a cat.



(b) The same picture, encrypted using Aes-128 in ECB mode: the cat remains visible.

Figure 1.3: Comparison of a picture before and after encryption using the ECB mode.

CBC (Cipher Chaining Block). In this mode of operation, each plaintext block is XORed with the preceding ciphertext block before applying the block cipher. As a result, each cipher block depends on all plaintext blocks previously processed. To ensure that ciphertexts remain distinct even when the same plaintext message is encrypted multiple times under the same key, the first plaintext block is XORed with a randomly chosen block of bits called an *initialization vector* (IV), as shown in Figure 1.4.

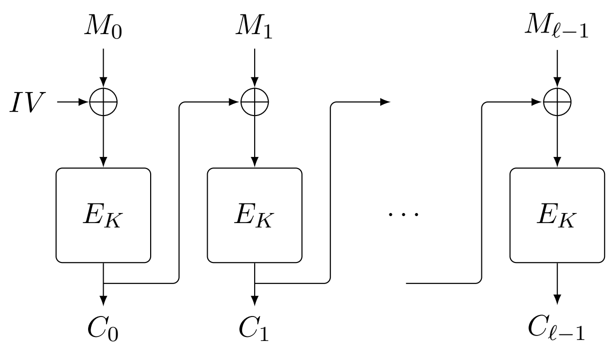


Figure 1.4: Encryption in the CBC mode of operation.

Despite being a sequential encryption mode—decryption, however, allows for parallelism—CBC is a very popular way of using block ciphers.

For some applications, the ciphertext should have exactly the same length as the original plaintext; thus preventing the use of padding. Other modes of operations address this problem: CFB (Cipher Feedback), OFB (Output Feedback) and CTR (Counter), depicted in Section 1.3.1, Figure 1.6 and Figure 1.7 respectively. These modes transform a block cipher into a stream cipher—they work by XORing the plaintext with the output of the block cipher. In order to treat an incomplete last block, one simply selects the required number of bits from the output of the block cipher. All three modes only use the encryption operation of the block cipher.

CFB. This mode is very similar to CBC as it also uses chaining (or *feedback*) to induce diffusion between the blocks and destroy patterns in the plaintext data. In CFB, an initialization vector is used as initial state. The state is encrypted and the encryption result is used as keystream. The resulting ciphertext is then fed back to the state.

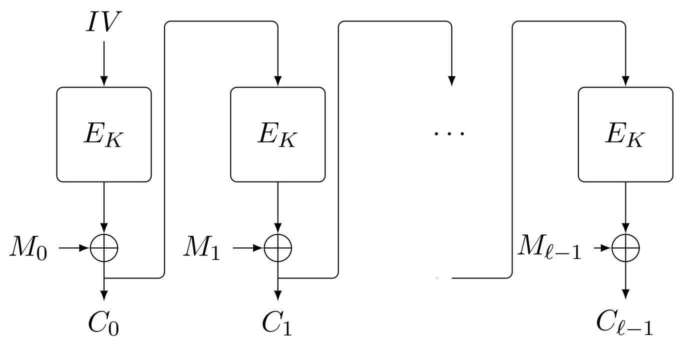


Figure 1.5: Encryption in the CFB mode of operation.

OFB. This mode differs from CFB in the way feedback is accomplished: instead of feeding the previous ciphertext back to the state, OFB directly uses the keystream.

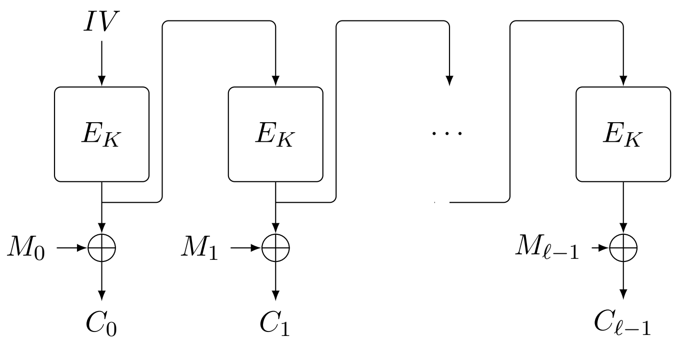


Figure 1.6: Encryption in the OFB mode of operation.

CTR. The CTR mode uses a counter to produce the keystream—this counter can be as simple as an ascending number. Consequently, this mode allows for parallelism in both encryption and decryption.

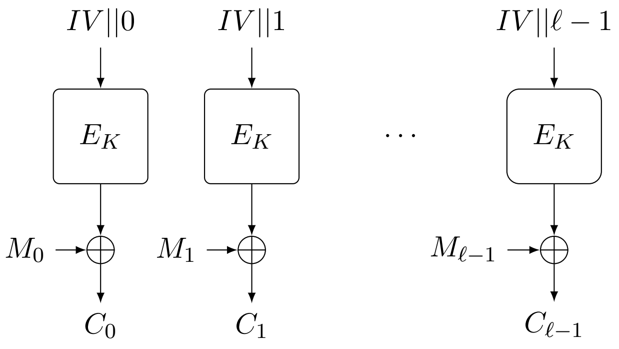


Figure 1.7: Encryption in the CTR mode of operation.

While block ciphers only provide secrecy on their own, some modes of operation combine confidentiality and authenticity such as OCB (Offset Codebook, EAX (Encrypt-then-authenticate-then-translate), CCM (Counter with CBC-MAC), and GCM (Galois Counter Mode). These modes are suited for *authenticated encryption with associated data* (AEAD) schemes, which will be discussed in Chapter 2.

1.3.2 Iterated block ciphers

Generally, each permutation E_K, for $K \in \mathbb{F}_2^k$, should be complex enough for it to be hard to distinguish from a random permutation—this is called PRP[4] security. Specifying the entire codebook[5] for an entire family of 2^k permutations on the set $\{0,1\}^n$ with such property is quite challenging, especially since the size of a block usually ranges between 64 and 256 bits (for security and implementation reasons).

[4]Pseudo-Random Permutation.

[5]A codebook is a lookup table for coding and decoding. It shows the relationship between the input blocks and the output blocks.

For this reason, all block ciphers are designed using a simple key-dependent transformation called the *round function* that is applied r times in an iterative fashion to compute the output—iterations may vary by the addition of *round constants*.

Definition 1.2 (Iterated block cipher)**.** *An iterated block cipher is a block cipher obtained by iterating r times an invertible transformation F called the round function, each time with a round key k_i derived from the master key K by means of a key scheduling algorithm. Denoting the n-bit plaintext with M, the encryption operation can be written as:*

$$E_K(M) = F_{k_{r-1}} \circ F_{k_{r-2}} \circ \cdots \circ F_{k_0}(M).$$

Figure 1.8 shows the general structure of an iterated block cipher. While the individual round function might be cryptographically weak, its repeated application allows for the construction of a complex encryption function. The choice of the proper number of iterations (or *rounds*) is performed via cryptanalysis of the cipher. The round function itself is made easy to study by composing several basic operations—such as transpositions, XORs, simple substitutions etc. In addition to the facilitated analysis, another advantage of this iterative structure is the compact hardware, software implementation.

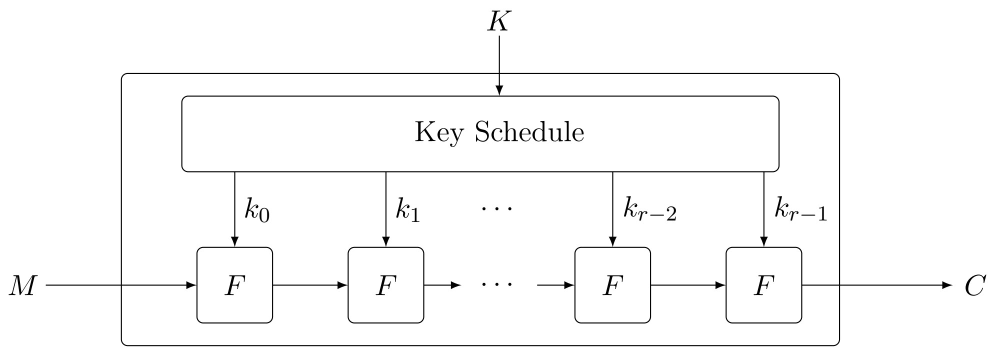


Figure 1.8: An iterated block cipher: the round function F is repeatedly applied, each time using a different round-key k_i derived from the master key K.

Iterated block ciphers may be divided into two large classes: *Feistel networks* and *Substitution-Permutation networks* (SPN) , both described in the following.

Feistel Networks

Named after cryptographer Horst Feistel, who introduced them in 1974 during the design of IBM's Lucifer block cipher [Fei74], *Feistel networks* are widely used structures in iterated block ciphers. The most prominent example of a Feistel network is the DES[6] [DES77], which was adopted by the U.S. government as an official standard from 1977 to 2000.

The round function of a Feistel network uses a keyed function F_k called the *Feistel function* and maps an n-bit input split into two halves (x_1, x_0), with x_1 and x_0 in $\mathbb{F}_2^{n/2}$, to an output (y_1, y_0) such that:

$$\begin{cases} y_1 = x_0 \\ y_0 = x_1 \oplus F_k(x_0), \end{cases}$$

[6]Data Encryption Standard.

as depicted in Figure 1.9. By construction, a Feistel round is a permutation over $\mathbb{F}_2^n$ regardless of whether F_k is bijective or not, and inverting such a round is simple; the function is the same, up to the permutation of the two halves:

$$\begin{cases} x_0 = y_1 \\ x_1 = y_0 \oplus F_k(y_1). \end{cases}$$

As a result, implementing decryption on top of encryption is very cost-efficient in the case of Feistel networks. Furthermore, the left and right branches are usually not exchanged in the last round. That way, the scheme becomes an involution[7] and the same algorithm can be used to both encrypt and decrypt up to a reversal of the round-key order.

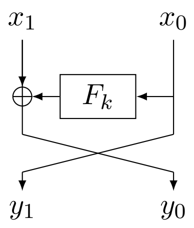


Figure 1.9: A Feistel round.

Examples of block ciphers relying on a Feistel structure in the literature include LBlock [WZ11a], Lilliput [BFMT16] or the NSA[8]'s Simon [BSS+13]. Variants of this construction exist, such as *unbalanced Feistel networks* [SK96], where the two branches are of different sizes. In the case of *Misty-like structures*—named after the Misty block cipher [Mat97]—the Feistel function has to be a permutation and is directly applied to a branch rather than a copy of it. Finally, *Generalized Feistel Networks*, which are the main focus of Chapter 8, extend the Feistel network to a larger number of branches.

Substitution-Permutation Networks (SPN)

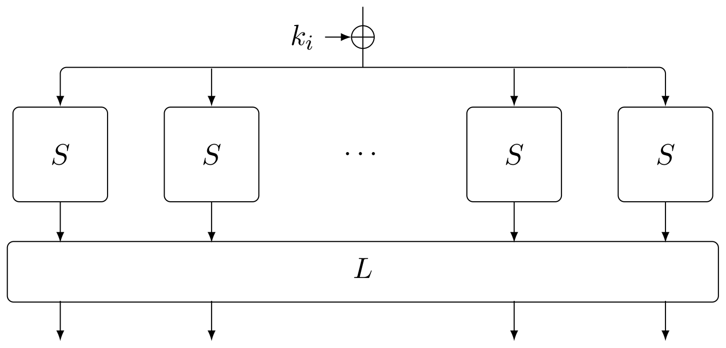


Figure 1.10: One SPN round.

In an SPN round, the internal state is first XORed with the round key k_i. As shown in Figure 1.10, it then goes through two layers to satisfy Shannon's confusion and diffusion properties:

[7]An involution is a function f that is its own inverse. Namely, $f(f(x)) = x$.
[8]National Security Agency, American intelligence agency.

1. A *substitution layer* consisting of parallel applications of small nonlinear functions—usually operating on 4 or 8 bits—called *S-boxes*[9]. This layer adds to local confusion.

2. A *permutation layer* acting on the entire block state, that mixes the entire internal state. Modern ciphers tend to use linear mappings instead of simple permutations to reach better diffusion in fewer rounds.

As opposed to Feistel ciphers, the SPN structure requires that all inner components be invertible. The most prominent SPN is the current NIST[10] standard, the Advanced Encryption Standard [AES01] (AES), but many other algorithms exist in the open literature, such as KLEIN [GNL11], MIDORI [BBI+15], NOEKEON [DPAR00], PRESENT [BKL+07], ROBIN [GLSV15], SKINNY [BJK+16] and ZORRO [GGNS13].

A focus on the AES. In the two decades following the design of the DES, computers had dramatically increased in power, making the 56-bit key of the algorithm—the encryption standard at the time—vulnerable to exhaustive search. In 1997, NIST initiated an open process to find the successor to the DES. All submissions had to be block ciphers supporting a block size of 128 bits and key sizes of 128, 192, and 256 bits. As a result of this effort, after three years of analysis, RIJNDAEL [DR02], designed by Belgian cryptographers Joan Daemen and Vincent Rijmen, was selected among 15 candidates as the AES [AES01] (Advanced Encryption Standard) to replace the DES.

The algorithm operates on a 128-bit internal state s which can be represented as a 4×4 array of bytes, as depicted in Figure 1.11. The size of the internal state and the size of the round keys k_i are both fixed to 128 bits, however, the number of rounds r (10, 12 and 14) and the key scheduling algorithm can change according to the key size (128, 192 and 256 bits respectively).

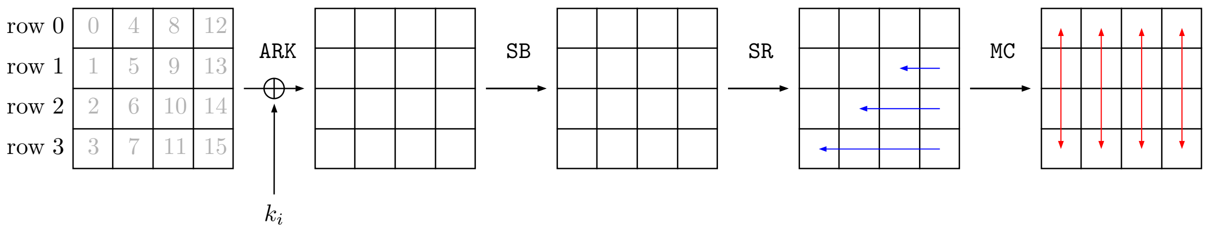


Figure 1.11: One round of the AES.

Round function The round function combines four operations:

1. `AddRoundKey` (`ARK`). At the beginning of each round, the state s is XORed with the round key k_i: $\texttt{AddRoundKey}(s) = s \oplus k_i$.

2. `SubBytes` (`SB`). The same S-box S is applied to each of the 16 bytes. This S-box is derived from the multiplicative inverse over the finite field $\mathbb{F}_{2^8}$, which is known to have good cryptographic properties.

3. `ShiftRows` (`SR`). Row i is cyclically shifted by i positions to the left, for $i = \{0, \cdots, 3\}$. This step is instrumental in preventing the columns from being encrypted independently.

[9]Substitution Box.
[10]National Institute of Standards and Technology, American standardization authority.

4. `MixColumns` (`MC`). This step combines the four bytes of each column of s using an invertible linear transformation. It can be seen as a multiplication by the circulant matrix $\mathcal{M}$, defined as follows:

$$\mathcal{M} = \begin{pmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{pmatrix}$$

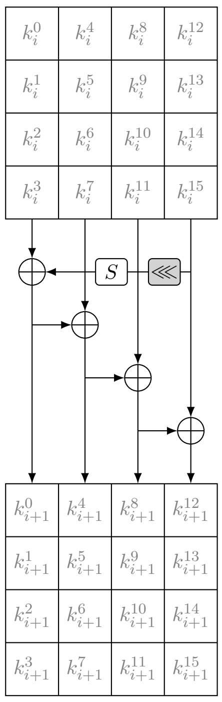
   where each byte is viewed as an element of the finite field

$$\mathbb{F}_{2^8} = \mathbb{F}_2[X]/(X^8 + X^4 + X^3 + X + 1).$$

The matrix $\mathcal{M}$ is defined by a so-called MDS[11] code and has a very interesting diffusion property; it ensures that each byte in the input column influences all four output bytes.

The `MixColumns` operation is omitted in the last round, an additional round key is added instead.

Key schedule The 128-bit keys version of the key schedule will be described in what follows. For more details regarding the other versions, the reader is referred to the complete specification [AES01].



k_i^0 k_i^4 k_i^8 k_i^{12}
k_i^1 k_i^5 k_i^9 k_i^{13}
k_i^2 k_i^6 k_i^{10} k_i^{14}
k_i^3 k_i^7 k_i^{11} k_i^{15}
S
$\lll$
k_{i+1}^0 k_{i+1}^4 k_{i+1}^8 k_{i+1}^{12}
k_{i+1}^1 k_{i+1}^5 k_{i+1}^9 k_{i+1}^{13}
k_{i+1}^2 k_{i+1}^6 k_{i+1}^{10} k_{i+1}^{14}
k_{i+1}^3 k_{i+1}^7 k_{i+1}^{11} k_{i+1}^{15}

Figure 1.12: Key schedule for the AES in the 128-bit keys version.

The 11 round keys k_i, $i \in \{0, \cdots, 10\}$, are generated from the master key K. The key state at round i is represented as a 4×4 matrix whose coefficients are denoted k_i^j, as shown in Figure 1.12.

[11]Simply put, in coding theory, Maximum Distance Separable codes have the greatest error correcting and detecting capabilities.

The first round key k_0 simply equals K, then for $i \in \{1, \cdots, 10\}$, the key state is updated as follows.

$$k_i^j = k_{i-1}^j \oplus S(k_{i-1}^{12+(j+1) \mod 4}) \oplus rc_i^j \qquad j \in \{0,1,2,3\} \tag{1.1}$$

$$k_i^j = k_{i-1}^j \oplus k_i^{j-4} \qquad j \notin \{0,1,2,3\} \tag{1.2}$$

Equation (1.1) describes how to compute the first column of k_i: the last column of k_{i-1} is rotated by one position—top byte is moved to the bottom etc.—then, each byte is ran through the AES S-box S. Next, the column is XORed with a 4-byte round constant $rc_i = [rc_i^3 rc_i^2 rc_i^1 rc_i^0]$ and finally with the first column of the previous round key k_{i-1}. Equation (1.2) states that the other columns of k_i are obtained by XORing the previous column with the same column of the previous round key k_{i-1}.

2

Lightweight Cryptography

Light as a feather and hard as dragon scales.

Bilbo Baggins

Modern cryptographic algorithms are fairly useless on their own: for real-life applications, they have to be implemented in actual devices that process information encoded in a binary representation. Implementation can either be done in *software*—the cipher is written as a code that is then turned into a program that can be executed on a general-purpose processor—or the algorithm can be built directly in real *hardware*, for instance on an FPGA[1] or an ASIC[2], depending on the specific requirements of the use cases.

While most modern symmetric ciphers such as the AES were designed for high-end machines such as personal computers or web servers, the last two decades saw the proliferation of a variety of interconnected devices in our environment—all encompassed within the very broad concept of the *Internet of Things* (IoT)—sometimes with very limited resources, thus only devoting a small fraction of their computing power to security. Yet, the expansion of the IoT in our daily lives clearly raises new security concerns with regard to privacy and availability of service. Indeed, sophisticated sensors and chips embedded in the physical devices that surround us continuously gather and transmit large amounts of data to offer a diversity of services, from connected watches tracking our daily physical efforts to sensors networks monitoring the production chain of a factory and many more examples extending to healthcare and agriculture among others. The IoT is a highly heterogeneous environment: although some of the devices use powerful processors and can be expected to use the same cryptographic algorithms as standard desktop PCs, many of them use extremely low power microcontrollers, such as RFID[3] tags or wireless sensors networks. Classical encryption schemes are not adapted to these platforms, which calls for new dedicated algorithms.

[1]Field Programmable Gate Arrays. FPGAs are integrated circuits typically found in small embedded devices. They are designed to be configured after manufacturing using a hardware description language (HDL).

[2]Application-Specific Integrated Circuits. An ASIC is an integrated circuit manufactured for a specific use or application. For this reason, they can provide high speed and optimized implementations, but they are more expensive to design.

[3]Radio-Frequency IDentification.

2.1 Lightweight Encryption Primitives

The AES marked the culmination of a four-year effort involving the cooperation between the U.S. Government, private industry and academia from around the world to develop a standard encryption primitive. Twenty years later, its security is well-established in the cryptographic community and it remains the most popular algorithm used in symmetric-key cryptography. Furthermore, hardware [BJM+14, BBR16, UMHA16] and software optimization [OBSC10, SS16] allow for decent—although not optimal—performances on a wide variety of platforms, making the AES an honorable candidate for most use cases. However, due to its large S-box and large block size, it remains a suboptimal choice for more stringent constraints.

Lightweight cryptography aims at providing security for these highly constrained systems, where cost, power consumption, energy, and available resources are limited. Many primitives have been proposed in the recent years[4], including CLEFIA [SSA+07], KLEIN [GNL12], LBLOCK [WZ11a], LED [GPPR11], LILLIPUT [BFMT16], PRESENT [BKL+07], SKINNY [BJK+16], SIMON and SPECK [BSS+13] and TWINE [SMMK13]. When compared to traditional ciphers, these new algorithms—in their lightest variants—sometimes operate with a smaller block size (64 bits) and key length (80 bits). Likewise, their components are simplified: smaller S-box (4 bits) when they use any, linear diffusion layer over a smaller alphabet ($\mathbb{F}_2$ for instance) and a simplified key-schedule.

2.1.1 Design criteria

It should be stressed that lightweight algorithms differ from their conventional counterparts in that they are tailored to satisfy specific constraints related to their use cases. Usually, aiming for both

[4]See the Lightweight Cryptography Lounge at `https://www.cryptolux.org/index.php/Lightweight_Block_Ciphers`

software and hardware efficiency is hard and trade-offs between security margins, performance and costs are common.

The efficiency of an algorithm can be assessed using appropriate cost and performance metrics, which are the memory usage, the implementation size and the speed of the algorithm. More details are given below, for hardware implementations and software implementations.

Software implementations

Algorithms can be implemented in software, typically for use on microcontrollers. In this case, designers tend to optimize the memory consumption of the algorithm during computation (in RAM[5]) and the code size of the algorithm itself (stored on a ROM[6] for instance). In regard to time complexity, the throughput of the algorithm, expressed in bytes per CPU[7] cycle, is an important metric. Latency, which is the time required for a single operation, should also be taken into account.

Microcontrollers do not operate at bit level as opposed to the hardware case; instead, they process bits in small batches, called *words*. These words usually contain 8, 16 or 32 bits. Consequently, one operation acting at bit level like seen sometimes in hardware, will mostly display very bad performance in software if it breaks this word structure. For this reason, software-friendly primitives rely exclusively on word-wide logical operations and rotations.

Hardware implementations

In hardware platforms, such as RFID tags, the gate area measures both the memory consumption and the size of the circuit implementing the algorithm. It can be expressed in μm^2, although this value is specific to the manufacturing technology of the circuit. For this reason, another unit is often used: the *Gate Equivalent* (GE), which is the silicon area of a NAND gate, in the considered technology. In addition, performance metrics such as the execution speed and the latency also have to be optimized; some applications indeed require a quick response time. Finally, the power consumption, measured in Watts should not be neglected, especially when considering devices running on batteries that cannot be replaced or recharged frequently.

Side-channel resistance

Side-channel attacks (SCAs) are physical attacks exploiting the implementation of a cipher to leak information about its internal state and break its security. In the context of the IoT, embedded devices are particularly exposed to such threats as they might operate in unprotected environments, allowing any attacker to easily gain access to them. For this reason, implementation of counter-measures such as reduction of information leakage, data shuffling or masking is not to be neglected when designing lightweight algorithms. Masked implementations have a substantial cost that is correlated to the number of nonlinear logic gates (AND gates, OR gates, etc.) in the circuit representation of the functions that need masking. As a result, reducing the complexity of nonlinear components such as S-boxes is crucial to the resistance to side-channel analysis.

[5]Random-Access Memory. RAM is a form of a volatile memory used to store the data of a program and its execution stack.

[6]Read-Only Memory.

[7]Central Processing unit.

An optimal trade-off?

All the criteria previously mentioned compete with one another and achieving full versatility is extremely hard. Consequently, the optimization sometimes focus on particular metrics at the expense of others, depending on the context. For instance, PRINCE [BCG+12] and MANTIS [BJK+16] target hardware implementation for applications such as data encryption on SSDs, and have thus been optimized for latency. The designers of MIDORI [BBI+15] focused on energy and power efficiency and mention applications such as battery-operated medical implants.

2.1.2 Overview of various design strategies

Low implementation cost is at the core of lightweight cryptography. In the case of block ciphers, this is achieved by minimizing the implementation costs of the two main components: the nonlinear layer and the linear layer.

Nonlinear operations

Nonlinearity brings confusion into a cipher. In practice, it is provided by S-boxes or through the use of nonlinear arithmetic operations.

Compact S-boxes. S-boxes are usually the most expensive components of a cipher. Many software implementations use S-boxes in the form of precalculated *look-up tables* (LUTs) to optimize throughput (retrieving a value from memory is faster than computing the S-box). However, this requires storing all possible outputs which, for an 8-bit S-box such as the one used by the AES, has a significant cost. In hardware, look-up tables would not fit any device and instead, the S-box is represented as a binary circuit using elementary logic gates. For an efficient hardware implementation, this circuit needs to be as compact as possible in terms of GE. While 4-bit S-boxes are rather well understood, finding a low-cost implementation of a given 8-bit S-box that is known to have good cryptographic properties is a hard problem, even though some convincing works have been made in this direction for the AES [Can05, BP09]. Another approach is to build small circuits that instantiate S-boxes with sufficient cryptographic properties [CDL16, CR19].

An alternative to LUT-based implementation for software platforms is called *bitslicing* [Bih97a] in which bitwise operations (such as XORs or ANDs) are performed on words of w bit, resulting in w parallel applications of the S-box. For a bitslice implementation to be efficient, the S-boxes need to be designed with this implementation strategy in mind from the start. They usually require only a limited number of logical operations to be evaluated, which makes their software implementation particularly easy to mask. The so-called LS-Designs [GLSV15] are examples of bitslice-based algorithms.

ARX structures. ARX-*based* ciphers were first introduced in [Miy91]. These algorithms are designed using only three operations: modular addition, which provides nonlinearity, with rotation and XOR for diffusion. Examples include SPARX [DPU+16], SPECK [BSS+13] and TEA [WN95].

ARX-based algorithms are particularly well-suited to software implementation because of the very simple operations they rely on. Moreover, they run in constant time, and therefore, are inherently immune to timing attacks. In contrast, modular addition is fairly expensive to implement in hardware.

Linear operations

Bit permutations. For diffusion, something as simple as bit permutation can be used. Bit-wise permutations come virtually for free in hardware since they are realized by simple wiring and require no transistors; they are thus a popular choice. For instance, PRESENT [BKL+07] is a hardware-oriented cipher that use a simple bit-wise permutation in its diffusion layer (see Section 2.4.1). The implementation cost of bit-wise permutations in software is, however, quite high, unless they describe rotations inside a word.

Compact MDS matrices. For more complex diffusion layers, MDS matrices can be used. Several works [JPS17, DL18] focus on global optimization of an MDS matrix. In [DL18], a search through a space of circuits is ran to find optimal circuits yielding MDS matrices. In particular MDS matrices with only 67 bitwise XORs were found, while a direct implementation of the MixColumns matrix of the AES requires 152 bitwise XORs.

Key schedule

A trend in lightweight cryptography consists in applying a very simple key schedule, which merely selects different bits of the master key adds them to some round constants to produce the round keys [GPPR11, BCG+12, BBI+15, BJK+16]. Some attacks has shown to be quite effective against such schemes, namely, *invariant subspace attack* [LAAZ11, LMR15] and *nonlinear invariant attack* [TLS16]. However, these structural attacks heavily rely on the choice of round constants and can be thwarted if the round constants are carefully chosen. In [BCLR17], Beierle *et al.* studied the mathematical nature of these so-called invariants and provided certain conditions under which an iterated block cipher could be resistant against invariant attacks. Wei *et al.* later showed in [WYWP18] that these criteria appeared to be necessary but not sufficient and presented additional design guidelines.

2.2 On Authentication

Enforcing security of IoT systems is a challenging task and this quest for "lightweightness" can not be reduced to encryption primitives: authentication and integrity solutions must also be provided, in spite of the resource constraints.

2.2.1 Authenticated encryption

Authenticated encryption (AE) aims at providing confidentiality, integrity and authentication at once. Authentication allows communicating entities to ensure that their communication has not been modified or tampered with. This verification is based on the computation of a so-called *tag* associated to the transmitted data which cannot be generated in reasonable time unless a secret is known.

Authenticated encryption with associated data (AEAD), introduced by Rogaway [Rog02], is an extension of AE that can ensure authenticity and integrity of additional *associated data* (AD) that does not require privacy protection. A common use case for such schemes is the transmission of network packets: the header, which may contain an IP address, has to remain unencrypted for the packet to reach its correct destination and the payload needs confidentiality, but both need authentication and integrity.

More formally, an AEAD scheme can be seen as a tuple $\Pi = (\mathcal{K}, \mathcal{E}, \mathcal{D})$, where $\mathcal{K}$ is a key scheduling algorithm, $\mathcal{E}$ is an encryption function and $\mathcal{D}$ a decryption/verification function. The function $\mathcal{K}$ takes as input an integer k and picks a secret key K uniformly at random from $\mathbb{F}_2^k$. The encryption function is specified as

$$\begin{aligned}\mathcal{E}\colon \mathbb{F}_2^k \times \mathbb{F}_2^n \times \mathbb{F}_2^* \times \mathbb{F}_2^* &\mapsto \mathbb{F}_2^* \times \mathbb{F}_2^t\\ (K, N, A, M) &\mapsto (C, T)\end{aligned}$$

where K is a secret key, N a nonce[8], A associated data, M a plaintext message, C a ciphertext and T an authentication tag. Decryption, on the other hand is defined by

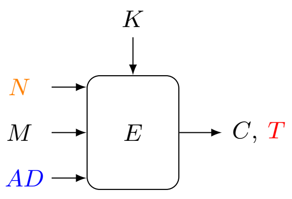
$$\begin{aligned}\mathcal{D}\colon \mathbb{F}_2^k \times \mathbb{F}_2^n \times \mathbb{F}_2^* \times \mathbb{F}_2^* \times \mathbb{F}_2^t &\mapsto \mathbb{F}_2^* \cup \{\perp\}\\ (K, N, A, C, T) &\mapsto \begin{cases} M & \text{if } T = T' \\ \perp & otherwise \end{cases}\end{aligned}$$

where T' denotes the authentication tag that was computed and T is the one that was received. Figure 2.1 depicts encryption and decryption/verification processes for AEAD. AE can be viewed as a special case where A is left empty.

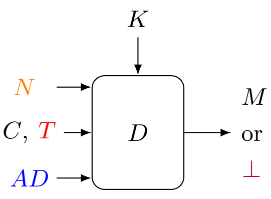
A communication between Alice and Bob is conducted as follows, assuming they have already agreed on a secret key K using a key exchange protocol:

1. Alice computes $\mathcal{E}_K(N, A, M) = (C, T)$ and sends (N, A, C, T) over the communication channel to Bob.

2. Bob uses the decryption function $\mathcal{D}_K$ on (N, A, C, T), which computes an authentication tag T' and compares it to the tag T that has been received. If the tag is valid *i.e.* $T = T'$, then $\mathcal{D}_K$ returns the message M. If tag verification fails, $\mathcal{D}_K$ outputs an error symbol denoted by $\perp$ and securely erases all intermediate results.



(a) AEAD encryption.



(b) AEAD decryption

Figure 2.1: Authenticated Encryption with Associated Data.

2.2.2 Generic composition

AE can be achieved by combining two separate primitives under independent keys: a conventional encryption algorithm $\mathcal{E}_{K_e}$ is used for encryption and a MAC $\mathcal{T}_{K_m}$ generates the authentication tag. Such constructions obtained through so-called *generic composition* were formalized by Bellare and Namprempere [BN00], and Katz and Yung [KY01]. They are widely used in practice and can be found in many standards such as the Secure Shell (SSH) protocol, Secure Sockets Layer/Transport Layer Security (SSL/TLS) or IPsec. These three protocols all use different types of composition that we detail below:

[8] A number that can be used only once.

Encrypt-and-MAC. The plaintext M is first encrypted using the encryption algorithm $\mathcal{E}$ and the MAC is applied on the plaintext to obtain the tag T. Finally, $C||T$ is sent, where

$$T = \mathcal{T}_{K_m}(M),\ C = \mathcal{E}_{K_e}(M).$$

MAC-then-Encrypt. The authentication tag T is produced based on the plaintext M, then it is appended to the plaintext before encryption. The resulting ciphertext C is sent.

$$T = \mathcal{T}_{K_m}(M),\ C = \mathcal{E}_{K_e}(M||T)$$

Encrypt-then-MAC. The plaintext M is first encrypted using the encryption algorithm $\mathcal{E}$ and the tag T is produced based on the resulting ciphertext C. Finally, $C||T$ is sent, where

$$C = \mathcal{E}_{K_e}(M),\ T = \mathcal{T}_{K_m}(C).$$

All three approaches can be extended to include associated data A in a rather straightforward manner by prepending A to the input of the MAC before sending the associated data, the ciphertext and the tag together.

These constructions were examined by Krawczyk [Kra01] and Bellare and Namprempre [BN00]. Both analyses recommended Encrypt-then-MAC for achieving AE, as this construction would always be secure as long as the underlying primitives were provably secure while the security of the Encrypt-and-MAC and MAC-then-Encrypt approaches could not be ensured generically. However, a more recent study by Namprempre, Rogaway, and Shrimpton [NRS14] gives a more extensive analysis of generic composition and shows that Encrypt- then-MAC constructions can fail, when AE is formalized differently, underlining that one must proceed with extreme caution when attempting to instantiate concrete schemes using generic composition.

Besides the danger of insecure instantiations of generic compositions [BKN02, Vau02, DR11, AP13, BBB+19], these approaches are also computationally suboptimal, as they require the use of two independent keys and two passes over the data; one providing confidentiality and the other, integrity assurance—they are said to be *two-pass* constructions.

The next section gives a a short overview of dedicated AEAD block cipher modes, which include several *one-pass* schemes.

2.2.3 Dedicated AE(AD) solutions

Dedicated solutions were conceived to achieve AE more efficiently, without using two different algorithms, two different keys, or making two separate passes over the message, potentially leading to schemes with a lower overhead for implementation and higher performance. Some of these developments include block cipher modes. The main authenticated encryption modes based on block ciphers are briefly detailed in the following.

Single-pass modes

The efficiency of AE(AD) schemes may be significantly improved if the constructions can, in a single pass over the data, provide both confidentiality and integrity simultaneously. In 2001, several single-pass provably secure AE designs were proposed: IACBC, IAPM by Jutla [Jut01]; XCBC, XECB by Gligor-Donescu [GD02]; and OCB, by Rogaway *et al.* [RBBK01, Rog04, KR11b, KR14]. All these proposals are patented.

IAPM (Integrity-Aware Parallelizable Mode). In 2001, Jutla proposed two seminal single-pass AE modes: IACBC (Integrity-Aware Cipher Block Chaining) and IAPM (Integrity-Aware Parallelizable Mode) [Jut01], built on the traditional CBC and ECB modes respectively, with the addition of a simple checksum[9] and masking of the outputs and inputs. As opposed to IACBC, IAPM is fully parallelizable and therefore, generated more interest. Compared to generic composition, which requires about $2 \times \ell$ block-cipher calls per ℓ-block long plaintext to achieve authenticated encryption, this scheme only needs around $\ell + \log_2(\ell)$ invocations. Further refinements to IAPM exist [Jut08] and reduce the number of block cipher calls down to $\ell + 1$. One of the variants of IAPM described in [Jut08] is depicted in Figure 2.2. IACBC operates in an analogous manner, except that CBC encryption is applied to the plaintext instead of ECB.

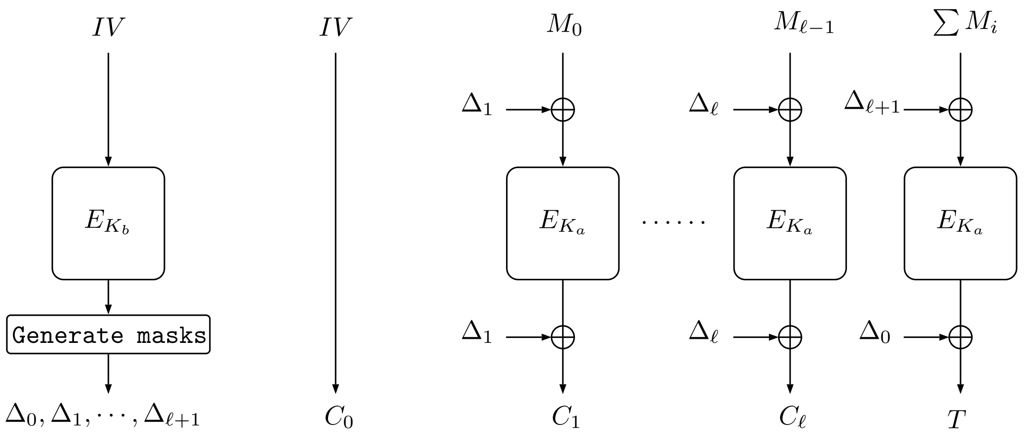


Figure 2.2: Integrity-Aware Parallelizable Mode (IAPM) as shown in [Jut08].

OCB (Offset Codebook). OCB [KR14] is a one-pass nonce-based AEAD block cipher mode originally proposed by Rogaway *et al.* in [RBBK01] as a follow up to IAPM. It allows parallelization of data processing and provides several improvements over IAMP such as encryption of arbitrary-length plaintexts with minimal ciphertext expansion—the resulting ciphertext has the same length as the plaintext plus the length of the authentication tag—and the use of a single block cipher key for both data encryption and masks (or *offsets*) generation. A second version, OCB2 [Rog04], was developed by Rogaway to support associated data. Finally, a third version introduced some minor changes regarding offset computation and brought some performance improvements [KR11b]. This version is illustrated by Figure 2.3 and Figure 2.4. OCB is an efficient mode of operation, that uses $\ell + 2$ calls to the block cipher in the worst case to encrypt and authenticate ℓ-block long messages. Whenever associated data A is included, ℓ_a additional block cipher calls are necessary, where ℓ_a is the block length of A. OCB achieves security up to the *birthday bound*—the mode is proven secure up to $2^{n/2}$ queries if the underlying block cipher uses blocks of n bits—and Ferguson described a collision attack matching this bound [Fer02]. Despite its attractive features, OCB never found widespread adoption due to patent restriction. The scheme is also not completely without flaws: in October 2018 a practical existential forgery attack on OCB2 was presented by Inoue and Minematsu [IM18], followed by several improvements [Poe18, Iwa18] leading up to a plaintext recovery attack [IIMP19]. Authors have stated that these attacks do not extend to OCB1 and OCB3.

[9]The simplest checksum algorithm consists in breaking the data into n-bit words and then computing the XORof all those words.

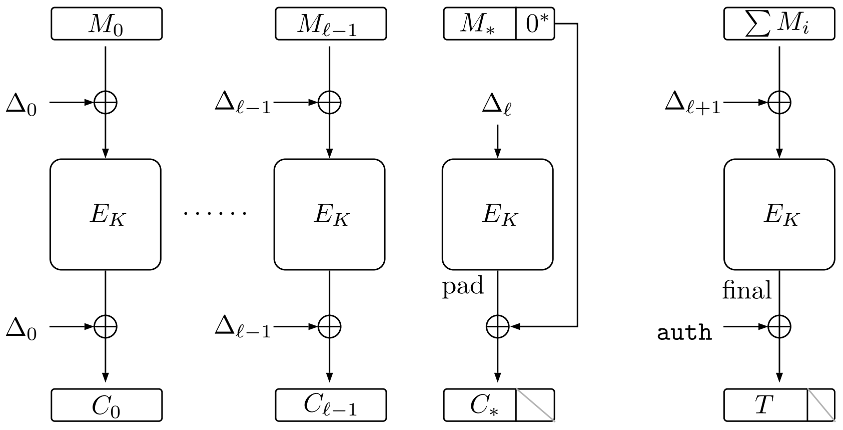


Figure 2.3: Message processing with padding in OCB3, the final version of OCB described in [KR11b]. The offsets Δ_i are computed from a nonce N and a simple increment function. This step requires a single call to the block cipher E under the same key K as the one used for encryption. Figure 2.4 shows how to compute the `auth` value.

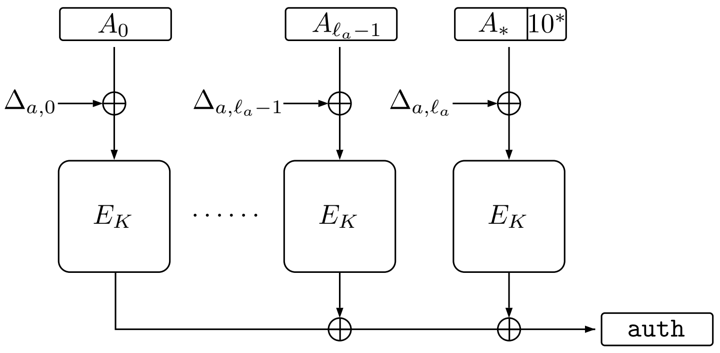


Figure 2.4: Associated data processing with padding in OCB3, the final version of OCB described in [KR11b]. The `auth` value is determined by the associated data A and the same key K as the one used for message processing. The offsets $\Delta_{a,i}$ are computed using the same increment function as before, starting from an initial value $\Delta = 0_{(128)}$.

XCBC (Extended Ciphertext Block Chaining). XCBC and XECB are two classes of schemes presented by Gligor and Donescu in [GD02], respectively similar to CBC and XECB modes of operation. Both schemes provide AE at a cost very close to that of encryption alone. Akin to IACBC and IAPM, they apply a mask to each message block before and after a block cipher invocation. However, the masks are generated using arithmetic modulo 2^n—n being the block size in bits—which is very fast on most processors.

Two-pass modes

The usage of fast one-pass schemes was hindered by intellectual property claims, which spurred researchers to develop further efficient AEAD patent-free algorithms. Theses schemes include CCM [WHF03], EAX [BRW04], CWC [KVW04] and GCM [MV04]—CCM and GCM are both recommended in NIST's special publications SP 800-38C [Dwo04] and SP 800-38D [Dwo07]. Although not as fast as the single-pass modes, they still offer significant performance improvements over generic composition schemes. Most importantly, for all the schemes listed below, a single block cipher key suffices for the entire scheme.

CCM (Counter with CBC-MAC). The first patent-free proposed was CCM by Ferguson, Housley, and Whiting [WHF03]. This AEAD mode only supports block ciphers with a 128-bit block size. It combines CBC-MAC—a MAC construction technique based on CBC mode—for authentication with CTR mode of encryption, using a MAC-then-Encrypt composition: CBC-MAC is first computed on the message to obtain a tag which is then encrypted together with the message using counter mode. CTR makes the scheme effectively a stream cipher that requires unique nonces for initialization as long as the key is fixed. This is necessary, as confidentiality can not be guaranteed for CTR if nonces are repeated.

CCM uses a MAC scheme and an encryption scheme, namely, CBC-MAC and CTR, which are both well known and provably secure modes. CCM does offer advantages over the generic composition of these two primitives; in particular, it uses the same key K for both the MAC and the encryption steps as previously stated. CTR mode requires no decryption function per se, since encryption and decryption are done simply by XORing the plaintext and ciphertext with a keystream, so hardware implementations do not need to implement the decryption functionality of the block cipher.

A notable inefficiency of CCM is that the length of the processed data must be known in advance, consequently it is not an *online* scheme and cannot be used for data streams.

EAX (Encrypt-then-Authenticate-then-Translate). EAX is a nonce-based AEAD block cipher mode designed in part by two researchers from the OCB team to provide another patent-free mode that addressed the several drawbacks to CCM [RW03]. It combines a variant of CBC-MAC called OMAC [IK03], which does not have the security deficiencies of raw CBC-MAC, and CTR encryption.

EAX is a very flexible mode; it has no restrictions on the underlying block cipher or on the block size n and supports arbitrary-length messages as well as authentication tag sizes $\tau \in \{0, \cdots, n\}$. Many other desirable features come with EAX: ciphertext expansion is minimal, it can pre-process static associated data, which results in performance savings and it is online for both the plaintext and the associated data, *i.e.* the total data length does not need to be known up front. Similar to CCM, only the forward cipher function of the block cipher algorithm is used.

GCM (Galois Counter Mode). Akin to CCM, GCM is an AEAD mode defined for block ciphers with a block size of 128 bits. It combines CTR mode of encryption and a MAC for authentication into a scheme which uses a single secret key. First the plaintext is encrypted using CTR mode, the MAC then processes the resulting ciphertext together with the associated data, in a Encrypt-then-MAC manner. The authentication process is based on multiplication in the Galois field $\mathbb{F}_{2^{128}}$ and does not require any block cipher invocation. GCM mode can reach high throughput rates and benefits from a structure that allows parallel processing and efficient use of pipelining, making it suitable to high-speed hardware-based applications.

CWC (Carter-Wegman + Counter). CWC is a mode of operation that combines the Carter-Wegman MAC algorithm with the CTR mode of operation under a common key. Contrary to EAX and CCM, which use sequential CBC-MAC type algorithms, CWC is parallelizable. NIST considered this mode for standardization, but eventually GCM was chosen instead in 2007.

Authentication modes for tweakable block ciphers

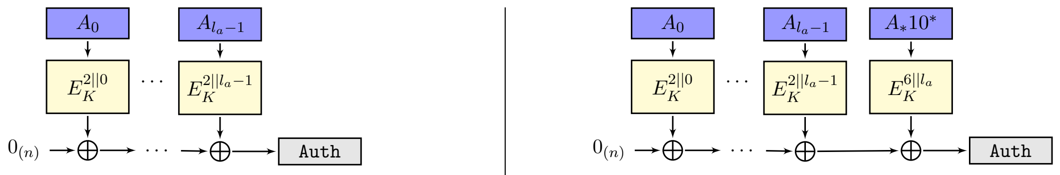
In the design authenticated encryption schemes, *tweakable block ciphers* (TBCs), are valuable building blocks. A tweakable block cipher is a generalized block cipher with an additional *public* input, the *tweak*, that is meant to provide variability—not security. If the tweak can be changed with little cost, then some interesting new operation modes become possible, such as AEAD modes that can provide authenticity and confidentiality with just one pass over the data; leading to schemes with a lower overhead for implementation and higher performance.

Tweakable block ciphers. The notion of tweakable block ciphers and their application to modes of operations was introduced by Liskov, Rivest and Wagner [LRW02]. This theme was developed by Rogaway in [Rog04], which described the first efficient constructions of TBC, namely XE and XEX, as well as several modes of operations. Formally, a tweakable block cipher is a function $\mathcal{E}\colon \mathcal{K} \times \mathcal{A} \times \mathcal{X} \mapsto \mathcal{X}$, where $\mathcal{E}_K(A, \cdot) = \mathcal{E}_K^A(\cdot)$ is a permutation with inverse $\mathcal{D}_K(A, \cdot) = \mathcal{D}_K^A(\cdot)$ for all $K \in \mathcal{K}$ and $A \in \mathcal{A}$. Here $\mathcal{X}$ is finite, and $\mathcal{A}$ is the set of tweaks, which might consist of variable-length strings. Whereas block ciphers only give access to a single permutation per key, tweakable block ciphers give access to an entire family, with the requirement that each member of the family looks uniform and independent of all other members.

The TBC-based AE(AD) modes include TAE [LRW02], ΘCB3 [KR11b], OTR [Min14], SCT [PS16], ZAE [IMPS17], PAEF, SAEF and RPAEF [ALP+19] and FBAE [NS19]. Some of these modes are detailed below.

TAE. TAE was introduced by Liskov, Rivest and Wagner in the same paper which formalized tweakable block ciphers [LRW02]. This mode essentially recasts OCB1 [RBBK01] using a tweakable blockcipher instead of a masked block cipher and thus does not define the handling of associated data.

ΘCB3. ΘCB3 is the TBC-based generalization of OCB3, introduced by Krovetz and Rogaway [KR11b]. As the same key material is used for different purposes, *domain separation* is required. Here, a 4-bit string prepended in the tweak ensures independence of the tweakable block cipher calls for different kinds of computations. Figure 2.5 describes the handling of associated data, with and without padding. The processing the plaintext is depicted in Figure 2.6.



(a) Without padding. (b) With padding.

Figure 2.5: Handling of the associated data for ΘCB3.

SCT. Contrary to ΘCB3, SCT is *nonce-misuse resistant* [RS06] *i.e.* repeating a nonce in encryption queries does not harm authenticity nor confidentiality. A slight variant, SCT-2, was described in the specification of DEOXYS [JNPS16]. This mode is also secure in the nonce-misuse scenario and is depicted in Figure 2.7, Figure 2.8 and Figure 2.9.

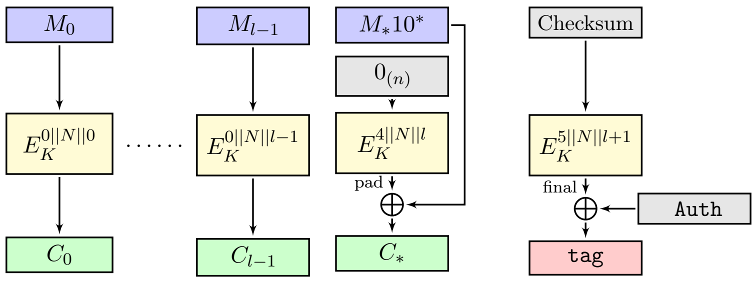


Figure 2.6: Message processing for ΘCB3 with padding when the message length is not a multiple of the block size.

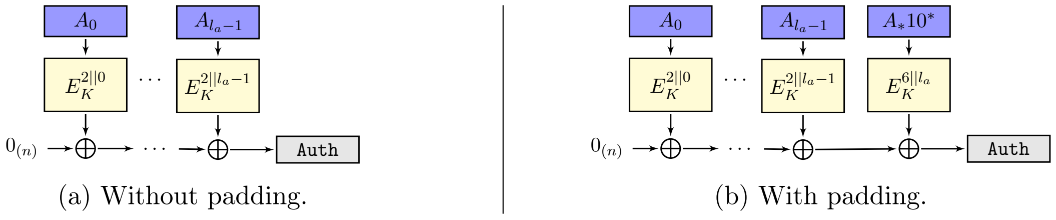


Figure 2.7: Handling of the associated data for SCT-2.

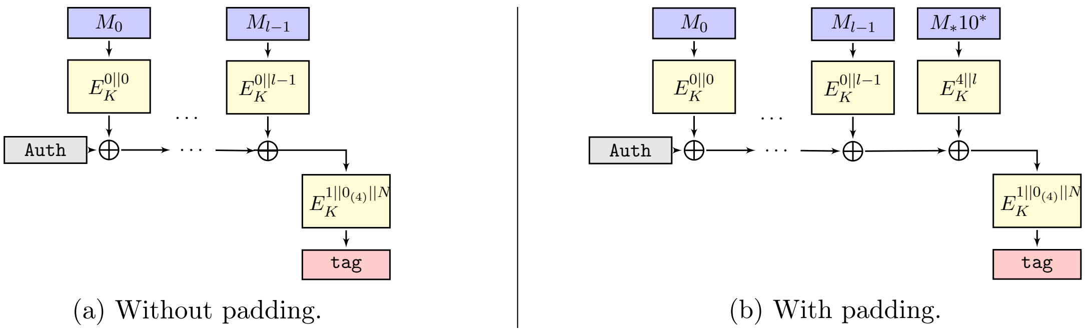


Figure 2.8: Message processing in the authentication part for SCT-2.

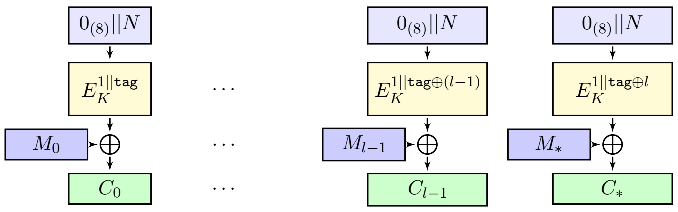


Figure 2.9: Message processing with padding in the encryption part for SCT-2.

2.3 Towards New Standards

Lightweight cryptography and its direct applications in the real world has drawn a lot of attention in the last two decades. Proprietary algorithms have been proposed by the industry, however, many were weak, mostly relying on their secrecy and these $ad - hoc$ solutions sustained various attacks once their specification was leaked or reverse-engineered. For instance, the specification of the KeeLoq [BSK14] algorithm – used in the remote car keys of many manufacturers—was leaked in 2006 which resulted in numerous attacks [CBW08, IKD+08, EKM+08, KKMP09, ABD+12]. The need for standard lightweight algorithms is well established and major standardization organizations are closely following the evolution of this research area.

2.3.1 Iso/iec cryptographic standards

Iso/iec 29192. The International Organization for Standards (iso) and the International Electrotechnical Commission (iec) are two organizations tasked with issuing and maintaining standards regarding information and communication technology. They have already standardized several lightweight primitives under the iso/iec 29192 reference [ISO12a], including block ciphers, such as Present [BKL+07] and Clefia [SSA+07], stream ciphers, as illustrated by Trivium [De 06], as well as hash functions and mac algorithms. Other primitives are currently considered for inclusion.

2.3.2 Open competitions

Open competitions are held when a need for new primitives is well acknowledged by the cryptographic community. Multiple proposals are submitted and analyzed by academic and private organizations, with the main objective of identifying a portfolio of algorithms that are suitable for widespread use, and sometimes defining new standards, as first done for the Aes[10] [AES01], then for the Sha-3 hash function [oST15]. Below are listed the international competitions in which lightweight applications have been considered.

The eSTREAM project (2004 - 2008). In 2004, the European Network of Excellence in Cryptology (ecrypt) announced the eSTREAM project[11], with the objective of selecting new dedicated stream ciphers instead of using the Aes with a mode of operation such as ctr. The call for submission was published in 2004, with two targeted profiles: stream ciphers for software applications with high throughput requirements and hardware applications with limited resources. The competition received 34 submissions, and resulted in a portfolio of several stream ciphers announced in 2008: HC-128 [Wu08], Rabbit [BVP+03], Salsa20/12 [Ber08], and Sosemanuk [BBC+08] for high-throughput software applications; Grain v1 [HJMM08], Mickey2.0 [BD08], and Trivium [De 06] for highly restricted hardware.

The Caesar competition (2014 - 2019). The launch of the cryptographic competition called Caesar (Competition for Authenticated Encryption: Security, Applicability, and Robustness) in 2014 demonstrated the need for new designs providing secure and efficient ae schemes. This competition was a move towards selecting a portfolio of ae schemes that improved upon the state of the art of the time, such as nist's Aes-gcm [Dwo07]. It received 57 proposals [CAE], with a variety of designs, from purely *ad-hoc* designs to tweakable block cipher operating modes. Three use cases were defined, including one regarding lightweight applications for resource-constrained environments. In February 2019, the competition ended with a total of 7 algorithms selected in the final portfolio. For the lightweight use case, Acorn [Wu16] and Ascon [DEMS16] were chosen, with Ascon being the main recommendation.

[10] A competition was also held by the National Bureau of Standards (nsb) for the Des, although it was only reserved to invited designers.

[11] https://www.ecrypt.eu.org/stream/-

The NIST LWC standardization process (2018 -). As part of their *Lightweight Cryptography* (LWC) project [NISa], launched in 2013, NIST has recently initiated a standardization effort in order to select symmetric encryption primitives that are lighter than the established NIST standards. In a report published in 2017 [MBTM17] NIST had indeed concluded that these were not well-suited for a range of resource-constrained devices. The requirements for their portfolio of lightweight algorithms were announced in August 2018 [NISb] and as a result, 57 proposals were submitted, of which 56 were accepted as first round candidates in April 2019. At the time of writing, round 2 is ongoing, with 32 candidates left.

2.4 Some Existing Lightweight Algorithms

The lightweight cryptography ecosystem has a plethora of designs, as evidenced by the variety of schemes recently submitted to the NIST standardization process. Most of the proposals can be divided into two main design families, namely substitution-permutation networks (SPNs) and Feistel networks. In the following section, four ciphers are presented, each with a distinct structural property:

1. PRESENT, an SPN with a bit-based permutation layer;
2. SKINNY, a family of SPN with an AES-like design;
3. LBLOCK, a two-branched Feistel scheme;
4. LILLIPUT, an extended generalized Feistel network.

Other structures exist among these families: ARX-based designs are fairly popular [DPU+16, BSS+13], some SPNs use a bitsliced S-box [GLSV15] and some are designed to minimize the overhead of decryption (reflection ciphers) [BCKL17], to name a few. For a more complete and in-depth listing, the interested reader can refer to [BP17] or [BP15] by Biryukov and Perrin. The same authors also contributed to the development of the FELICS benchmarking framework [DBG+15, DCK+19], which was designed to provide a unified evaluation of the software performances of lightweight cryptographic primitives.

2.4.1 PRESENT

PRESENT was introduced at CHES 2007 [BKL+07] and is one of the earliest lightweight encryption algorithm ever published. It has been standardized for applications requiring lightweight cryptographic implementations and is specified in ISO/IEC 29192-2 [ISO12b]. PRESENT adopts a simple SPN structure with a block size of 64 bits and a key size of 80 or 128 bits.

Round function. As shown in Figure 2.13, each of the 31 rounds of PRESENT is made of one subkey addition, a parallel application of a 4-bit S-box and a bit-oriented permutation.

Key-Schedule. PRESENT supports keys of either 80 or 128 bits. In the 80-bit keys version, the master key K is stored in a register denoted by $K_{79}K_{78}\cdots K_0$. At round i, the 64-bit round key k_i consists of the 64 leftmost bits of the current key state:

$$k_i = K_{79}K_{78}\cdots K_{16}.$$

Then the key state is updated as follows.

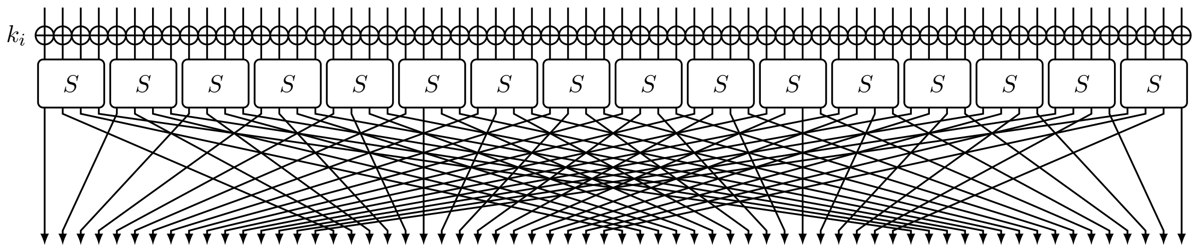


Figure 2.10: One round of the PRESENT cipher.

1. The key register is rotated by 61 bit positions to the left:

$$[K_{79}K_{78}\cdots K_1K_0] = [K_{18}K_{17}\cdots K_0K_{79}\cdots K_{20}K_{19}].$$

2. The leftmost four bits are passed through the PRESENT S-box:

$$[K_{79}K_{78}K_{77}K_{76}] = S([K_{79}K_{78}K_{77}K_{76}]).$$

3. The binary representation of the round counter value i, denoted by $[i]_2$, is XORed with bits $K_{19}K_{18}K_{17}K_{16}K_{15}$ of K with the least significant bit of i on the right:

$$[K_{19}K_{18}K_{17}K_{16}k_{15}] = [K_{19}K_{18}K_{17}K_{16}K_{15}] \oplus [i]_2.$$

Because of its diffusion layer, this cipher clearly favors hardware implementations. Indeed, bit-oriented permutations can be implemented in hardware using simple wiring but they are not software-friendly operations. Its S-box is also very compact (only 28 GE) and has good cryptographic properties as well. Being one of the first lightweight proposals, PRESENT has been extensively studied [Wan08, ÖVTK09, NSZW09, YWQ09, Lea11, CS09, KCS11, BKLT11, KSK13, WSTP12, SHW+14, BPW15, BTV18]. Various cryptanalysis techniques (see Chapter 3) have been used and some of the notable results include a statistical saturation attack which targets up to 24 rounds [CS09], a truncated differential attack on 26 rounds [BN14] and a 28-round multi-dimensional linear attack with time complexity 2^{122} for PRESENT-128, and $2^{77.4}$ for PRESENT-80 [FN20].

2.4.2 SKINNY

SKINNY [BJK+16] is a family of AES-like tweakable block ciphers designed to reach performances similar to those of NSA's SIMON [BSS+13] with additional strong security guarantees.

SKINNY follows the TWEAKEY framework [JNP14], which offers a simple way of designing tweakable block ciphers by treating the tweak and the key materials as a single input called the *tweakey*. Various tweakey and block sizes are supported and the resulting variants are denoted SKINNY-n-t, where n represents the block size (64 or 128 bits) and t is the tweakey size (n, $2n$ or $3n$).

Round function. All members of the family use an AES-like round function depicted in Figure 2.11. This function operates on a 4×4 matrix of cells of s bits, where $s = 4$ in the 64-bit variant and $s = 8$ in the 128-bit variant. The operations are listed below.

1. `SubBytes` (`SB`). The same S-box—acting on 4 bits in the 64-bit version and on 8 bits in the one with a 128-bit state—is applied to each of the 16 cells.

2. `AddConstants` (`AC`). Round constants derived using a 6-bit LFSR are added into to state.

3. `AddRoundTweakey` (`ART`). Round keys are dependent on both the master key and the tweak are XORed to the first two rows of the internal state at each round.

4. `ShiftRows` (`SR`). Row i is cyclically shifted by i positions to the right, for $i = \{0, \cdots, 3\}$. This operation is similar to the one performed in the AES, however, the direction of the rotation is inverted.

5. `MixColumns` (`MC`). Each column is multiplied by a binary matrix M defined as follows:

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

All these operations are summarized in Figure 2.11.

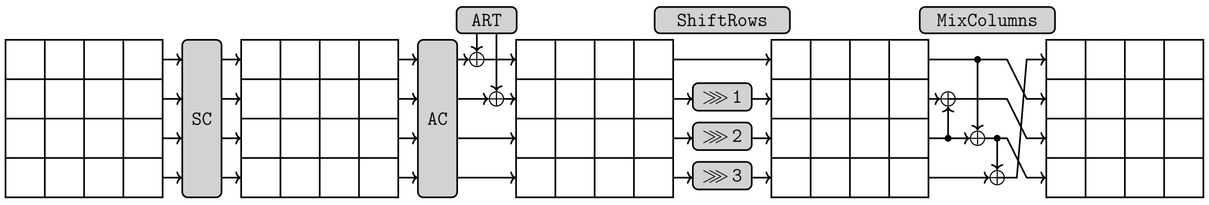


Figure 2.11: One round of the SKINNY cipher.

Tweakey schedule. As for the internal state, the tweakey state is arranged in $z = t/n$ 4×4 matrices, denoted by $TK_1, \cdots, TK_z$. At each round, for $i \in \{1, \cdots, z\}$, the first two lines of the TK_i matrices are extracted and XORed together to produce a *subtweakey* that is used as a round key in the `ART` operation.

Next, each of the z matrices are transformed by the permutation defined as

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

where indices are taken row-wise.

Finally, when $z = 2$ or $z = 3$, the first two rows of TK_i, where $i \neq 1$, are further modified by means of LFSRs given in Table 2.1, as illustrated by Figure 2.12.

Due to its good hardware/software performances, the SKINNY family of block ciphers has received much attention from the cryptographic community such as [ZR17, ABC+17, SGL+17, ST17, AST+17, BCLR17, AK19, EKKT19, HAV18, ABI+18, LTW18, CHP+18, CSSH19, DLU19, SQH19, ZCGP19]. Among all the cryptanalysis results, the best key-recovery attack in the single-key model uses impossible differentials to target 22 rounds out of 40 for SKINNY-64-192, 22 rounds out of 48 with time complexity $2^{245.72}$, for SKINNY-128-256 and 20 rounds out of 56

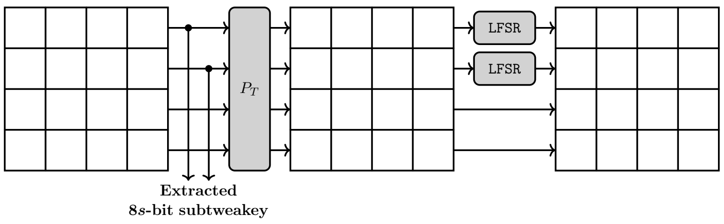


Figure 2.12: The SKINNY tweakey schedule. Each tweakey word is updated in a similar way, except that no LFSR is applied to TK_1.

TK	***s***	**LFSR**
TK_2	4	$(x_3\|\|x_2\|\|x_1\|\|x_0) \mapsto (x_2\|\|x_1\|\|x_0\|\|x_3 \oplus x_2)$
	8	$(x_7\|\|x_6\|\|x_5\|\|x_4\|\|x_3\|\|x_2\|\|x_1\|\|x_0) \mapsto (x_6\|\|x_5\|\|x_4\|\|x_3\|\|x_2\|\|x_1\|\|x_0\|\|x_7 \oplus x_5)$
TK_3	4	$(x_3\|\|x_2\|\|x_1\|\|x_0) \mapsto (x_0 \oplus x_3\|\|x_3\|\|x_2\|\|x_1)$
	8	$(x_7\|\|x_6\|\|x_5\|\|x_4\|\|x_3\|\|x_2\|\|x_1\|\|x_0) \mapsto (x_0 \oplus x_6\|\|x_7\|\|x_6\|\|x_5\|\|x_4\|\|x_3\|\|x_2\|\|x_1)$

Table 2.1: The LFSRs used in the variants of SKINNY with 2 or 3 tweakey words. Each word is made of 16 s-bit cells (x_0 is the least significant bit of the cell).

with time complexity $2^{373.48}$ for SKINNY-128-384 [TAY17]. In [SSD+18], authors mounted a Demiric-Selçuk meet-in-the-middle attack on 22 rounds of SKINNY-128-384 in the single-key model with time complexity $2^{382.46}$ using a 10.5-round distinguisher. In the related-tweakey settings, some of the best results were obtained using rectangle attacks [LGS17, ZDM+19]: 27 rounds of SKINNY-64-192 were attacked with time complexity $2^{165.5}$ and a 28-round attack with a complexity of 2^{315} in time was found for SKINNY-128-384. Authors of [LGS17] also investigated impossible differential attacks. In particular, they obtained a 23-round related-tweakey impossible differential attack with time complexity $2^{251.47}$ for SKINNY-128-256, using a truncated related-tweakey impossible differential over 12 rounds. This result was later improved in [SMB18]. The authors were able to find a 15-round related-tweakey impossible differential which allowed them to decrease the time complexity of the attack to $2^{243.41}$. All in all, no practical attacks have been found so far and SKINNY still offers a security margin above 30% for a 64-bit block size and 50% for the 128-bit version.

Several candidates of the NIST lighweight competition are based on SKINNY. Namely, FORKAE [LPR+19], REMUS [IKMP19a], ROMULUS [IKMP19b], SKINNY-AEAD/SKINNY-HASH [BJK+19] use SKINNY as a building block. Other submissions leverage some components derived from SKINNY: GIFT-COFB [BCI+19], HYENA [CDJN19], LOTUS-AEAD [CDJ+19], SUNDAE-GIFT [BBP+19] and TRIFLE [DGM+19] apply the same LFSR to generate their round constants and in SPOOK [BBB+19], the S-box is a variant of the one used in SKINNY.

2.4.3 LBLOCK

At ACNS 2011, Wu and Zhang proposed the LBLOCK lightweight block cipher [WZ11a]. LBLOCK is a block cipher with a block length of 64 bits and a key length of 80 bits that iterates 32 times a Feistel-based round function.

Round function. The round function is a modified classical two-branched Feistel structure, where the receiving branch of the Feistel function is first rotated by 8 bits. The inner round function is made of one round key addition, a parallel application of 8 different 4-bit S-boxes and a nibble[12]-wise permutation.

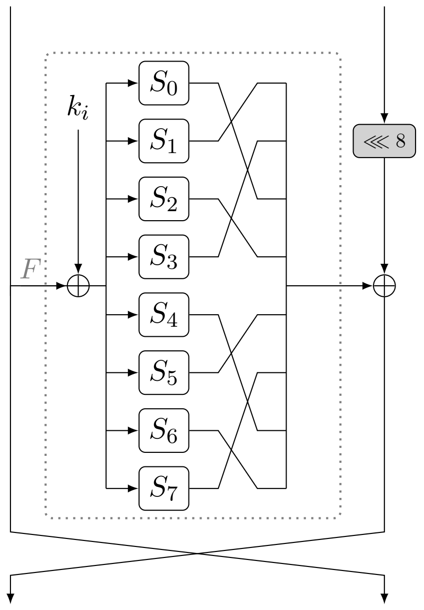


Figure 2.13: One round of the LBlock cipher.

Key schedule. The 80-bit master key K is stored in a key register and denoted as $K = K_{79}K_{78}\cdots K_1K_0$. At round i, the 32-bit round key k_i consists of the 64 leftmost bits of the current key state:

$$k_i = K_{79}K_{78}\cdots K_{48}.$$

Then the key state is updated as follows.

1. The key register is rotated by 29 bit positions to the left:
$$[K_{79}K_{78}\cdots K_1K_0] = [K_{50}K_{49}\cdots K_0K_{79}\cdots K_{52}K_{51}].$$
2. The leftmost eight bits are modified using two S-boxes S_8 and S_9:
$$[K_{79}K_{78}K_{77}K_{76}] = S_9([K_{79}K_{78}K_{77}K_{76}])$$
$$[K_{75}K_{74}K_{73}K_{72}] = S_8([K_{75}K_{74}K_{73}K_{72}]).$$
3. The binary representation of the round counter value i, denoted $[i]_2$, is XORed with bits $K_{50}K_{49}K_{48}K_{47}K_{46}$ of K with the least significant bit of i on the right:
$$[K_{50}K_{49}K_{48}K_{47}k_{46}] = [K_{50}K_{49}K_{48}K_{47}K_{46}] \oplus [i]_2.$$

Interestingly, LBlock's diffusion layer is equivalent to that of the decryption of Twine [SMMK13], although the two proposals are independent. A simplified version of LBlock called LBlock-s

[12]A nibble is half a byte *i.e.* 4 bits.

is used as underlying primitive in the authenticated encryption cipher LAC [ZWW+14], which was part of the first-round candidates of the CAESAR competition until it was attacked by Leurent [Leu16]—not to be confused with the eponym public key encryption scheme [LLJ+19], a former candidate to NIST's post-quantum cryptography standardization process. To reduce the cost in software and hardware implementation, a single 4-bit S-box is used instead of the 10 different ones. Additionally, the new key schedule discards the 29-bit rotation and replaces it with a nibble-friendly 24-bit rotation, which yields better performances on 8-bit platforms. It also features added XORs to improve diffusion. Some studies of LBLOCK and its variant LBLOCK-S can be found in [SN12, SW13, LGW12, ZNS12, EMPS13, BMNS14, WW14, SHW+14, BNS14, XJHL15, WWJ16, LWZ16, Lu16]. Some attacks worth mentioning include an impossible differential attack by Boura *et al.* [BMNS14], which breaks 23 rounds of LBLOCK with time complexity $2^{75.36}$ and data complexity 2^{59} in the single-key model, a 22-round zero-correlation linear cryptanalysis [SN12] and an integral analysis [SW13] that targets 22 rounds of LBLOCK as well. Chapter 7 presents additionnal cryptanalysis results on LBLOCK-S.

2.4.4 LILLIPUT

LILLIPUT [BFMT16] was designed by Berger *et al.* in 2015. It is the first instantiation of the so-called *Extended Generalized Feistel Network* [BMT14] (EGFN) construction, a generalization of Feistel networks with an arbitrary number of branches that uses a linear mapping instead of a simple permutation of the branches so as to achieve fast diffusion.

The state of LILLIPUT is made of 64 bits, seen as 16 nibbles denoted $x_{15}, \cdots, x_0$ and the key length is 80 bits.

Round function. The round function, called `OneRoundEGFN`, is iterated 30 times. In more details, `OneRoundEGFN` is composed of three layers:

- `NonLinearLayer`: this layer only updates the first half of the state. After the round key is added to $x_7, \cdots, x_0$, a 4-bit S-box S is applied in parallel to the 8 nibbles. Namely, each F_j seen in Figure 8.11 is defined as $F_j = S(x_j \oplus k_j^i)$ where k_j^i is the j-th nibble of the 32-bit round key k^i of round i, for $j \in \{0, \cdots, 7\}$.

- `LinearLayer`: this layer helps to increase the diffusion speed with a small additional cost. It consists in having x_7 propagate to all nibbles of the second half of the state while having x_{15} influenced from all nibbles of the first half of the state: block x_7 is XORed with blocks x_9 to x_{15} and blocks x_1 to x_6 are XORed with block x_{15}.

- `PermutationLayer`: finally, the permutation π is applied to the bytes. The permutation was chosen to achieve the highest number of active S-boxes after 18, 19 and 20 rounds. This layer is omitted in the last round for involution reasons.

Key schedule. The key schedule expands the 80-bit master key K to 32-bit round keys for round i, with $j \in \{0, \cdots, 29\}$ denoted by k^j.

The key schedule depicted in Figure 2.15 produces the 30 round keys k^0 to k^{29} from the 80-bit master key K.

To do so, it uses an 80-bit Linear Finite State Machine (LFSM) whose inner state Y is initialized with the master key K. The first round key k^0 is extracted from the LFSM initial state (*i.e.* the

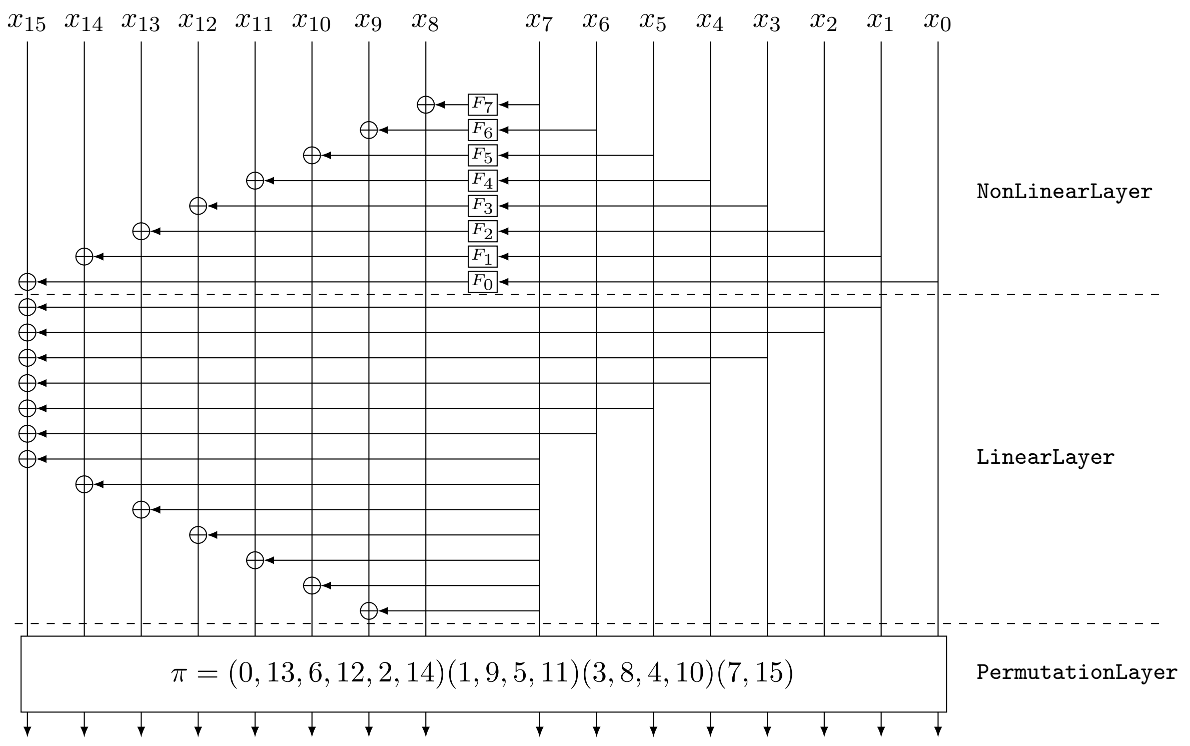

x_{15} x_{14} x_{13} x_{12} x_{11} x_{10} x_9 x_8 x_7 x_6 x_5 x_4 x_3 x_2 x_1 x_0
F_7 F_6 F_5 F_4 F_3 F_2 F_1 F_0
NonLinearLayer
LinearLayer
$\pi = (0, 13, 6, 12, 2, 14)(1, 9, 5, 11)(3, 8, 4, 10)(7, 15)$
PermutationLayer

Figure 2.14: The `OneRoundEGFN` function of LILLIPUT.

master key K). The LFSM state Y is then updated using `RoundFnLFSM` and the next round key is extracted using `ExtractRoundKey` until all 30 round keys are generated.

The state Y is made of 20 nibbles $Y_{19}, \cdots, Y_0$ split among four LFSRs, $\mathcal{L}_0$ to $\mathcal{L}_3$, acting on 5 nibbles each: $\mathcal{L}_0$ acts on Y_0 to Y_4, $\mathcal{L}_1$ on Y_5 to Y_9 and so on. These four LFSRs used in the LILLIPUT key schedule are Feistel-like word-oriented LFSRs acting on 5 nibbles, inspired by the results of [BMP09] and [ABMP11].

The `RoundFnLFSM` function seen as 4 Feistel-like word-oriented LFSRs can be divided into two transformations: `MixingLFSM` which holds the feedbacks, followed by `PermutationLFSM` which is the word-wise cyclic shift, as depicted in Figure 2.15.

- `MixingLFSM`: For each of the 4 parallel LFSRs $\mathcal{L}_0$ to $\mathcal{L}_3$, it consists in XORing some nibbles to some others nibbles in a Feistel-like manner:
  - for $\mathcal{L}_0$: $Y_0 \leftarrow Y_0 \oplus (Y_4 \ggg 1)$ and $Y_1 \leftarrow Y_1 \oplus (Y_2 \gg 3)$,
  - for $\mathcal{L}_1$: $Y_6 \leftarrow Y_6 \oplus (Y_7 \ll 3)$ and $Y_9 \leftarrow Y_9 \oplus (Y_8 \lll 1)$,
  - for $\mathcal{L}_2$: $Y_{11} \leftarrow Y_{11} \oplus (Y_{12} \ggg 1)$ and $Y_{13} \leftarrow Y_{13} \oplus (Y_{12} \gg 3)$,
  - for $\mathcal{L}_3$: $Y_{16} \leftarrow Y_{16} \oplus (Y_{15} \ll 3) \oplus (Y_{17} \lll 1)$.
- `PermutationLFSM`: For each of the 4 parallel LFSRs $\mathcal{L}_0$ to $\mathcal{L}_3$, it consists in a left cyclic shift of its 5 nibbles, i.e. $Y_i \leftarrow Y_{i-1 \bmod 5}$.
- `ExtractRoundKey`: For each round of encryption, the subkey k^i is extracted from the LFSM inner state using a non linear extracting function in a bitsliced way. First, some nibbles of the state are extracted:

$Z_{(32)} \leftarrow Y_{18}||Y_{16}||Y_{13}||Y_{10}||Y_9||Y_6||Y_3||Y_1$ and let $Z_{31}, \cdots, Z_0$ be the bits of Z then $k^i_j = S(Z_j||Z_{8+j}||Z_{16+j}||Z_{24+j})$

where S is the same S-box as in the datapath. Finally, the current round number $i \in \{0, \cdots, 29\}$ is xored to the last 5 bits of the subkey: $k^i \leftarrow k^i \oplus i_{(5)}||0_{(27)}$.

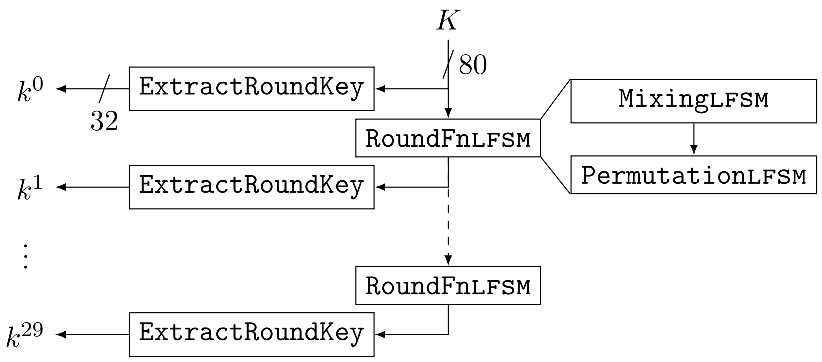


Figure 2.15: Lilliput Key Schedule

Lilliput is the underlying primitive used in the tweakable block cipher Lilliput-AE [ABC+18] that was submitted as a candidate to the NIST lightweight cryptography standardization process. This cipher is presented in Chapter 4.

The properties of Lilliput are studied in [ST16, ST17, MNV18, ST18]. The best attack, presented in [ST16, ST18], uses the division property [Tod15b], a generalization of integral and higher-order differential distinguishers to find an integral distinguisher on 13 rounds (out of 30) that can be extended to a 17-round key recovery attack with time complexity 2^{77}.

With regard to Lilliput-AE, Dunkelman *et al.* discovered a practical forgery attack [DKLS19] which exploited a weakness in the tweakey schedule, and thus does affect Lilliput. This resulted in an updated design of Lilliput-AE (see Section 4.5.2).

3

Cryptanalysis

Please! God damn it! I hate this hacker crap!

Ray Arnold

Cryptanalysis is the science that aims at finding attacks compromising the security of a cryptographic algorithm, without the knowledge of the secret key. Nowadays, the adoption of a scheme is for a large part based on the cryptanalytic efforts of the community. The security assessment of a cryptographic primitive consists in a careful study of its properties, in an attempt to retrieve the secret key or break the security claims. A successful attempt can discard an algorithm, while unsuccessful attempts tend to increase the confidence in the security of said algorithm. The most naive key recovery approach is brute-force and consists in testing all possible values. This attack is impossible to carry out on modern cryptosystems; the keyspace is indeed too large to be fully explored in practice, as illustrated by Table 3.1 and the following excerpt from [LKT13]:

> *"Boiling all water on the planet (including all starfish) amounts to about 2^{24} lakes of Geneva and leads to* global security*: 114-bit symmetric cryptosystems, 228-bit cryptographic hashes, and 2380-bit* RSA*. This needs to be done 16 thousand times to break* AES*-128,* SHA*-256, or 3064-bit* RSA*."*

Number of sand grains in the Sahara desert	$10^{23} \approx 2^{77}$
Number of atoms on Earth	$10^{51} \approx 2^{170}$
Number of atoms in the Universe	$10^{77} \approx 2^{265}$

Table 3.1: Some significant orders of magnitude. It would take more time than the age of the universe to break 128-bit encryption.

In fact, many cryptanalytic attacks are wildly impractical because of their tremendous *complexity* in terms of time[1], memory[2] needed or amount of data[3] required. For instance, the

[1]Processing complexity is the total number of operations required to perform the attack. It is measured in some agreed upon unit, usually the number of elementary operations or the number of calls to the encryption function.

[2]Storage complexity is the memory space required for the attack.

[3]Data complexity is the total number of data (either ciphertexts or plaintext-ciphertext pairs) the attacker can access.

best published attack against the Aes-128 at the time of writing requires 2^{126} operations [TW15], which is a slight improvement over brute-force but still remains infeasible in practice. For this reason, the term *attack* might seem confusing to some of the readers who are unfamiliar with cryptography. An actual attacker will not be able to exploit them to recover the secret key; instead, they should be thought of as security analyses illustrating unusual behaviors overlooked by the designers.

In some cases, the weaknesses found by cryptanalysts do not even cover the full version of a cryptographic primitive but rather a weaker version of it, such as a round-reduced variant in the case of an iterated block cipher. The smaller the difference between the attacked version and the full cipher, the less the cipher is considered secure. The number of rounds that remain "unbroken" for the worst-case attack defines the *security margin* of the cipher. There are two main types of attacks against symmetric ciphers.

– The first encompasses the more *classical attacks* that consider the ciphers as mathematical objects. In these attacks, the cryptanalyst exploits a weakness in the algorithm to recover the key in a more efficient way than exhaustive search. The main classes of attacks are described in Section 3.1.3.

– *Physical attacks*, which will not be covered in this thesis, are the second type of attacks. These attacks make use of the physical properties of the implementation of a cryptographic algorithm. For instance, some key-dependent computations might induce observable physical phenomena such as difference in timings, power consumption, temperature variations or electromagnetic radiations. A careful study of these correlations can leak information on the internal state of the cipher. Physical attacks can either be *passive—side-channel analysis* (SCA)—or *active* in the case of *fault injection attacks* (FIA). Side-channel analysis is based on the observation of the circuit while it operates the computations of an encryption process; fault injection analysis studies the propagation of a fault throughout the system.

3.1 General Principles of Classical Cryptanalysis

Before presenting the main classes of generic attacks that can be found in the literature, a major component of cryptanalysis is discussed in this section: *distinguishers* and how they can be used to gain some information about the secret key.

3.1.1 Distinguishers

Ideally, a block cipher should be indistinguishable from a random permutation. That is, given a permutation picked uniformly at random from the set of all permutations and a block cipher operating with an unknown key, an adversary should not be capable of successfully telling them apart with a probability higher than $\frac{1}{2}$. Any property allowing them to successfully differentiate the two transformations can be exploited to build a distinguisher.

Definition 3.1 (Distinguisher)**.** *A distinguisher is an algorithm $\mathcal{A}$ that can decide with a probability higher than $\frac{1}{2}$ whether a set of plaintext/ciphertext pairs was generated by a random permutation or by a cryptosystem.*

More precisely, a distinguisher has access to an auxiliary function, called an *oracle*, to which it can make queries. For any message m received, the oracle answers randomly but consistently by returning a ciphertext c produced either by the targeted cipher C or by a random permutation P_n. The distinguisher $\mathcal{A}$ must be able to determine wether the oracle function is C or P_n using q requests. Conventionally, the distinguisher returns 1 if it thinks that the messages received were encrypted using C and 0 otherwise. The probability that the distinguisher returns 1, while the oracle actually uses C is denoted by $p = \Pr[\mathcal{A}^C = 1]$. Similarly, the probability that the distinguisher returns 1, while the oracle actually uses P_n is denoted by $p^* = \Pr[\mathcal{A}^{P_n} = 1]$. The efficiency of the distinguisher is measured by the *advantage* of $\mathcal{A}$:

$$Adv_{\mathcal{A}}(C, C^*) = |p - p^*|.$$

The higher the advantage, the better the distinguisher. In other words, to obtain a good distinguisher, the cryptanalyst has to look for a property of the cipher between the plaintexts and the ciphertexts that has a probability that is significantly higher than what would be observed for a random permutation. Several methods for building such objects are described in Section 3.2 and Section 3.3.

3.1.2 Key recovery and last round attack

While a distinguisher in itself is not an attack, such an object can be used to retrieve some bits of the secret key in a *key recovery attack*. Indeed, an iterated block cipher with r rounds can be attacked using a distinguisher on $r - 1$ rounds. To do so, the general idea is to brute-force the last-round subkey and, for each guess, decrypt the ciphertext to know the internal state of the cipher at round $r - 1$. If a subkey is such that the internal state at round $r - 1$ is coherent with the distinguisher, then it is likely the correct guess. If not, it can be discarded.

This type of attack relies on the *wrong-key randomization hypothesis* [HKM95], which states that a wrong guess of the key does not yield any bias and will result in values that are random and uniformly distributed.

Definition 3.2 (Wrong-key randomization hypothesis). *Let $E_K \colon \mathbb{F}_2^n \to \mathbb{F}_2^n$ a block cipher parameterized by a key K with a round function F. Denoting k_{r-1} the subkey used in the last round, the wrong-key randomization hypothesis assumes the following:*

$$Pr[F^{-1}_{\bar{k}_{r-1}}(E_K(X)) \oplus F^{-1}_{\bar{k}_{r-1}}(E_K(X \oplus \delta)) = \Delta] = \begin{cases} p & \text{if } \bar{k}_{r-1} = k_{r-1} \\ \frac{1}{2^n-1} & \text{otherwise.} \end{cases}$$

If an adversary succeeds in determining the value of the last-round subkey, they can reapply a similar attack to find the second-last round subkey etc. Then, inverting the key scheduling algorithm reveals some bits of the master key, although in practice, this can require a sophisticated analysis. An example is provided for differential cryptanalysis in Section 3.2.1.

3.1.3 Overview of cryptanalytic techniques

Most of the cryptanalytic methods can be divided into two classes: *statistical attacks* and *structural* or *deterministic attacks*.

Statistical attacks

In the early 1990s, cryptanalysis of symmetric primitives made a leap forward as two powerful cryptanalytic techniques were published, namely *differential cryptanalysis* by Biham and Shamir [BS91a] and *linear cryptanalysis* [TG92, Mat94] by Matsui, followed by many variants. These techniques have greatly influenced the design of block ciphers and the approaches developed to protect ciphers against them are still used to this day [DR01, DPU+16]

Differential and linear cryptanalysis are both *statistical attacks*, meaning that they study statistical relations between the plaintexts and ciphertexts, where the probability is determined by components of the cipher as well as the secret key.

- Differential cryptanalysis studies the *propagation of differences* in a cipher, exploiting the statistical bias in the output difference distribution of the cipher when the difference between two plaintexts is fixed. The cryptanalysis results on SKINNY and on SPOOK presented in Chapter 6 and Chapter 5, respectively, rely on this technique.

- In linear cryptanalysis, the bias is induced by linear approximations of the nonlinear round function involving bits of the plaintext, the ciphertext and the key that are verified with a probability that is significantly higher than for a random permutation.

The variants of differential and linear cryptanalysis include:

- Truncated differential cryptanalysis [Knu95]
- Impossible differential cryptanalysis [Knu98, BBS99]
- Multiple differential cryptanalysis [BG11]
- Higher-order differential cryptanalysis [Knu95, Lai94]
- Boomerang cryptanalysis [Wag99]
- Differential-linear cryptanalysis [LH94]
- Multiple linear cryptanalysis [BDQ04]

– Multidimensional linear cryptanalysis [CHN09]

– Zero-correlation linear cryptanalysis [BR11].

An example of boomerang attack against Feistel networks is studied in Chapter 7.

Structural attacks

Structural attacks are mostly based on algebraic properties of a cipher. For instance, *algebraic attacks* [CP02] exploit the relations between bits of the plaintext, the ciphertext and the key. To retrieve the key in a way that is faster than exhaustive search, the goal here is to collect a sufficient number of relations that are simple enough to create a system of equations that can be solved efficiently—such systems either have low degree, small size or a specific structure. This type of attack is very useful against stream ciphers but can be valuable in the case of block ciphers as well, especially when combined with other types of attacks— see Albrecht and Cid's algebraic differential attack [AC09]. One could also mention *meet-in-the-middle* (MITM) attacks [DH77]. This type of attack requires the knowledge of some plaintext and their corresponding ciphertexts. The attacker then tries to compute part of the internal state—usually the middle state of the cipher during the encryption process—from both ends, by partial key-guessing: one pattern starts from the plaintext, going forward to the middle state, the second one, going backward from the ciphertext. If the values of the middle state do not match, then the key-guess is wrong and can be discarded. Finally, the *division property* [Tod15b] (Section 3.3.1) is an example of the more novel structural attacks. This technique extends *integral attacks* [DKR97] and has shown to be effective against SPNs and Feistel constructions, as demonstrated by Todo's attack [Tod15a] against the full version of MISTY1 [Mat97].

Statistical distinguisher	Structural distinguisher
Differential cryptanalysis	Higher-order differential cryptanalysis
Linear cryptanalysis	Impossible differential cryptanalysis
Truncated differential cryptanalysis	Integral cryptanalysis
Differential-linear cryptanalysis	Zero-correlation linear cryptanalysis
Boomerang attack	Division property
Rectangle attack	Interpolation attack
Multiple linear cryptanalysis	Zero-sum distinguisher
Multiple differential cryptanalysis	Algebraic attack
Multidimensional linear cryptanalysis	

Table 3.2: Some of the main cryptanalytic techniques.

The major cryptanalytic techniques are summarized in Table 3.2. In the following, some of the main attacks are detailed. Section 3.2 focuses on statistical attacks and their variants while Section 3.3 presents some structural attacks. A panorama of existing cryptanalytic techniques against block cipher is provided in [SPQ03] or in [KR11a].

3.2 Statistical Attacks

3.2.1 Differential attacks

Differential cryptanalysis was first introduced by Biham and Shamir [BS91a] and was used in 1991 for the first attack against the DES which was faster than exhaustive search [BS93]. For the remaining, let E be a block cipher operating on blocks of n bits using a key K of k bits and r rounds of a function F. M and M' are two plaintext blocks, and their intermediate state throughout the encryption process under K is denoted by $M = X_0, X_1, \ldots, X_{r-1}, X_r = E_K(M) = C$ and $M' = X'_0, X'_1, \ldots, X'_{r-1}, X'_r = E_K(M') = C'$.

Simply put, differential cryptanalysis is a chosen plaintext attack that studies how an input difference evolves through the various rounds of the cipher. More precisely, this technique analyses the probability of observing an output difference δ_{out} given an input difference δ_{in} of a function f:

$$\Pr[f(x \oplus \delta_{in}) \oplus f(x) = \delta_{out}].$$

When f is a permutation picked uniformly at random, any difference might occur and the probability has an expected value of 2^{-n} for any pair of nonzero n-bit elements $(\delta_{in}, \delta_{out})$, and any n-bit input x. However, for a block cipher with a key picked uniformly at random, some output differences may have higher probabilities than others, which leads to a distinguisher. The input/output difference pair $(\delta_{in} \rightsquigarrow \delta_{out})$ is called a *differential*.

Definition 3.3 (Differential). *A differential on t rounds of an iterated cipher E, with $t \leq r$, is a pair $(\delta_{in}, \delta_{out}) \in \mathbb{F}_2^n \times \mathbb{F}_2^n$ such that $X_0 \oplus X'_0 = \delta_{in}$ and $X_t \oplus X'_t = \delta_{out}$.*

The cipher E reduced to t rounds will be denoted by E^t in what follows, as shown in Figure 3.1. In its simplest form, differential cryptanalysis aims at finding *high-probability* differentials. The probability of a differential can be defined as follows.

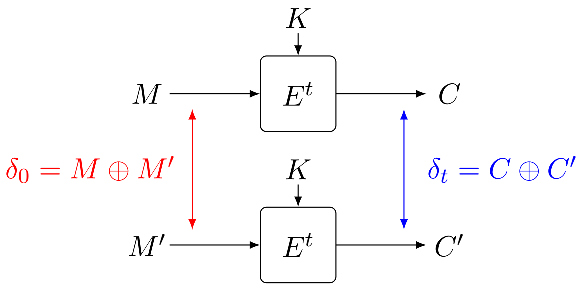


Figure 3.1: A differential (δ_0, δ_t) on t rounds of a cipher E.

Definition 3.4 (Probability of a differential). *The probability of a differential $(\delta_0 \rightsquigarrow \delta_t)$, denoted by $Pr[\delta_0 \rightsquigarrow \delta_t]$, is defined by:*

$$Pr[\delta_0 \rightsquigarrow \delta_t] = Pr_{X,K}[E_K^t(X) \oplus E_K^t(X \oplus \delta_0) = \delta_t],$$

where $Pr_{X,K}$ is the probability computed on all possible input plaintexts $X \in \mathbb{F}_2^n$ and all possible keys $K \in \mathbb{F}_2^k$.

In practice, computing the probability of a differential on t rounds becomes very difficult as t increases. A simpler approach consists in studying *differential characteristics* (or *differential trails*), which specify the evolution of the differences after each round.

Definition 3.5 (Differential characteristic). *A differential characteristic on t rounds of a cipher E is a sequence $(\delta_0, \delta_1, \ldots, \delta_t) \in (\mathbb{F}_2^n)^{t+1}$ that specifies the input/output difference for* *each round:*

$$\begin{aligned}\delta_0 &= X_0 \oplus X'_0\\ \delta_1 &= X_1 \oplus X'_1\\ &\vdots\\ \delta_t &= X_t \oplus X'_t\end{aligned}$$

In the single key setting (see Chapter 1, Section 1.2.3), a *right pair* refers to two input messages whose generated differences match the ones predicted by the characteristic, otherwise it is called a *wrong pair*. The probability of a differential trail is defined as follows.

Definition 3.6 (Probability of a differential characteristic). *The probability of a differential characteristic $(\delta_0, \delta_1, \dots, \delta_t) \in (\mathbb{F}_2^n)^{t+1}$ is given by:*

$$Pr[\delta_0, \delta_1, \dots, \delta_t] = Pr_{X,K}[E_K^i(X) \oplus E_K^i(X \oplus \delta_i) = \delta_t,\ \forall i \in \{1, \dots, t\}].$$

This theoretical probability is computed under the assumption that the round transitions are independent—this is called the *Markov assumption* [LMM91a]. As a result, the average differential probability over all possible keys of an t-round characteristic can be expressed as the product of the probabilities of its corresponding 1-round characteristics. The Markov assumption is usually justified by showing that the analyzed primitive is a *Markov cipher* and by assuming that the probability of any characteristic is roughly the same for the large majority of keys[4].

Definition 3.7 (Markov cipher). *An iterated cipher with round function $Y = F_k(X)$ is a Markov cipher if there is a group operation $\otimes$ for defining differences such that, for all choices of α $(\alpha \neq 0)$ and β $(\beta \neq 0)$,*

$$Pr[Y \otimes Y' = \beta \mid X \otimes X' = \alpha, X = \gamma]$$

is independent of γ when the subkey k is uniformly random.

In simple terms, a Markov cipher is an iterative cipher for which the average differential probability over one round is independent of the input of the round. Assuming that round keys are independent and uniformly distributed in the key space , the probability of a characteristic can therefore be expressed as:

$$\Pr[\delta_0, \delta_1, \dots, \delta_t] = \prod_{i=0}^{t-1} p_i,$$

where $p_i = \Pr[\delta_i \rightsquigarrow \delta_{i+1}]$ for $i \in \{0, \dots, t-1\}$.

As a way of building a differential distinguisher, an attacker has to study the round function of a cipher E to find high-probability characteristics on one round, which they can later combine into a full differential characteristic. The resulting probability p is the product of the probability of the characteristics on one round. Several characteristics can compose a differential, and the average probability can be computed—in principle—as the sum of the probabilities of all characteristics sharing the same input and output differences with the differential. However, most

[4]This is called the *hypothesis of stochastic equivalence* [LMM91b]. While this hypothesis is not always accurate for key-alternating ciphers [DR05, DR07, SWW18], several experimental results have shown that the difference between the expected value of a differential and the value obtain by the mean over the keys is small, making this a reasonable assumption [BG10, BBL13].

of time, the attacker will simply focus on high-probability characteristics, with no consideration for this clustering effect, as it is customarily assumed that the best characteristic gives a good approximation of the actual probability of the corresponding differential.

Search of characteristics

In order to compute the probability of a differential on one round of a cipher E, an attacker has to study its evolution through the several operations of the round function F. For the linear layer L of the cipher, this computation is trivial since any difference δ is transformed into a difference $L(\delta)$. However, for a nonlinear operation such as an S-box application, predicting the evolution of a nonzero difference with certainty is not possible if only the input and output differences—and not the real values—are known.

The *difference distribution table* (DDT) of an S-box is a table that lists the number of pairs that fulfill each possible input and output differences.

Definition 3.8 (Difference Distribution Table)**.** *The difference distribution table of an $n\times n$-bit S-box is a matrix of size $2^n \times 2^n$ such that*

$$DDT[a,b] = \#\{X \in \mathbb{F}_2^n \mid S(X \oplus a) \oplus S(X) = b\}.$$

The coefficient at row a, column b of this table is the number of inputs $X \in \mathbb{F}_2^n$ for which the difference a transitions to the difference b through the S-box. The corresponding probability is obtained by dividing this value by the total number of possible inputs X:

$$\Pr[a \rightsquigarrow_S b] = \frac{DDT[a,b]}{2^n}.$$

When there is there is no difference between the two inputs of an S-box, this S-box is said to be *passive* and the differential transition occurs with probability 1. Therefore, in order to obtain a high-probability characteristic, an attacker has to minimize the number of *active* S-boxes—S-boxes with nonzero input differences and ensure that their probabilities of transition are as high as possible. This maximum probability of transition is derived from the *differential uniformity* of an S-box.

Definition 3.9 (Differential uniformity)**.** *The differential uniformity of an S-box is the highest coefficient of its* DDT*, with the first row and the first column excluded.*

Once a differential $(\delta_{in} \rightsquigarrow \delta_{out})$ that has a probability p that is significantly higher than 2^{-n} has been exhibited, the attacker can build a distinguisher. For instance, they can ask for the encryption of $m \times p^{-1}$ pairs of messages whose difference equals δ_{in}, with m an integer; they should observe about m occurrences of δ_{out} in the resulting output differences.

Attacks on the last round

As introduced in Section 3.1.2, attacks on the last round make use of a statistical distinguisher to recover the encryption key. The remaining section focuses on their application to differential cryptanalysis.

In a differential attack on the last round, illustrated in Figure 3.2, the attacker has a differential denoted $(\delta \rightsquigarrow \Delta)$ of high probability p, that covers the first $r-1$ rounds of the attacked cipher E, parameterized by an encryption key K:

$$E_K = F_{k_{r-1}} \circ F_{k_{r-2}} \circ \cdots \circ F_{k_0},$$

where each round function F_{k_i} depends on a subkey k_i derived from the master key K.

This distinguisher can be turned into an attack covering the full cipher as follows:

1. *data collection phase*: ask for the corresponding ciphertexts $(C_{0,i}, C_{1,i})$ of q pairs of messages $(M_{0,i}, M_{1,i})$ whose difference equals δ, for $i \in \{0, \dots, q-1\}$;

2. *data analysis phase:*

   (a) guess the bits of the last round key $\bar{k}_{r-1}$;

   (b) using the guessed subkey, decipher the last round and observe the resulting differences $F^{-1}_{\bar{k}_{r-1}}(C_{0,i}) \oplus F^{-1}_{\bar{k}_{r-1}}(C_{1,i})$;

   (c) if the number of occurrences of Δ matches the theoretical bias (about $q \times p$), then the guess on $\bar{k}_{r-1}$ is likely to be correct *i.e.* $\bar{k}_{r-1} = k_{r-1}$, else go back to step 2a.

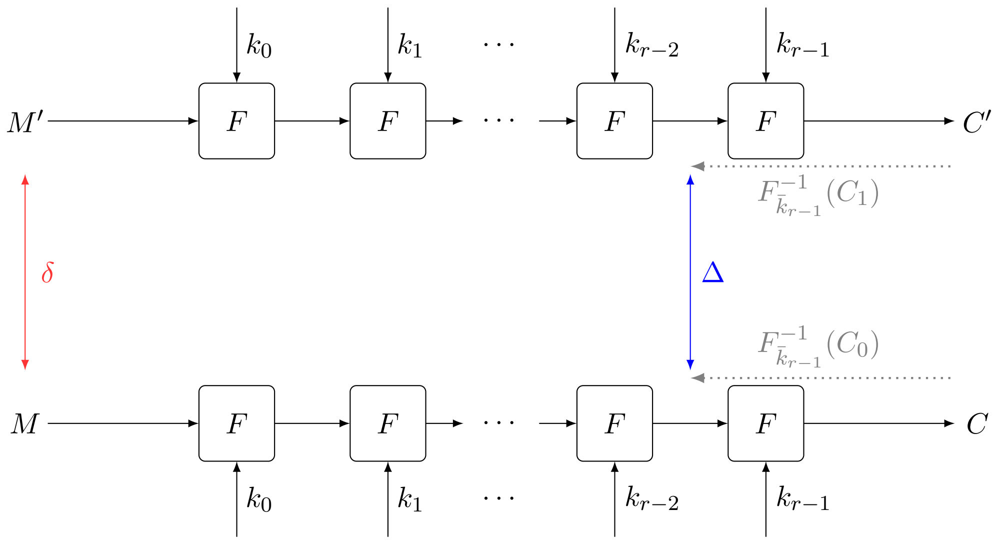


Figure 3.2: A differential attack on the last round of an iterated block cipher. $F^{-1}_{\bar{k}_{r-1}}$ denotes the inverse of the function F parameterized by the guessed key $\bar{k}_{r-1}$.

Extensions and variations

Truncated differential cryptanalysis. *Truncated differentials* are an extended form of differentials introduced by Knudsen [Knu95], in which not all the bits of the difference are fixed. For example, it might be possible to exhibit some differential patterns for which some bits are equal to zero, while the other bits can take different values and are left undetermined. One particular type of truncated differences that is widely used is word[5]-wise truncated differences. In such cases, the analysis usually focuses on predicting whether the difference of a word is zero or not. As a result, for the nonlinear layer, the precise value of the transitions of the S-boxes are not considered anymore; the sole information that can be obtained is wether an S-box is active—the input difference is nonzero—or passive—no difference on the inputs. The distinguishers on SPOOK presented in Chapter 5 rely on this technique.

[5]The word size depends on the native structure of the cipher.

Related-key differential cryptanalysis. In 1993, Biham proposed to consider related-key differential attacks [Bih94], in which the attacker can not only learn the encryption of two plaintexts M and M' under the original key K but also under some derived keys K' whose relations to the original key K are known—typically $K' = K \oplus c$, for some constant c that is known to the attacker. Although Figure 3.3 depicts the related-key setting for differential attack, this model can be applied to numerous cryptanalysis techniques. Both the single-key and the related-key models are considered in the differential analysis of SKINNY presented in Chapter 3

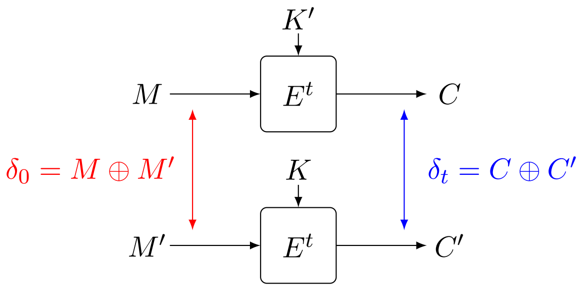


Figure 3.3: A related-key differential (δ_0, δ_t) on t rounds of a cipher E.

Impossible differential cryptanalysis. A differential $(\delta_{in} \rightsquigarrow \delta_{out})$ which has probability zero can be used in *impossible differential attacks* [Knu98, BBS99]. The most common approach is to combine two differentials of probability 1 for the cipher top and bottom that conflict in the middle when concatenated together; this technique is called *miss-in-the-middle* (see Section 3.3.2).

Higher-order differential cryptanalysis. Differential cryptanalysis can be extended to higher degrees [Lai94, Knu95]. Some ciphers are indeed secure against differential cryptanalysis but vulnerable to higher order attacks, such as the KN-CIPHER [NK95]. Instead of considering a difference between two values as done in the conventional differential setting, higher-order differential cryptanalysis studies the propagation of a set of differences between a collection of values. More precisely, a *d^{th}-order differential* is the difference between two $(d-1)^{st}$-order differentials and is a collection of 2^d texts. In general, high-order differential analysis is mainly successful against primitives with nonlinear components which have low algebraic degree and a small number of rounds, such as the KECCAK-f permutation [BCD11], used in the SHA-3 hash function.The *boomerang attack* can be seen as an application of second-order differentials (see Section 3.2.3).

3.2.2 Linear attacks

First introduced by Gilbert, Chassé and Tardy-Cordfir [GC91, TG92] then applied to the DES by Matsui [Mat94], linear cryptanalysis is a known plaintext attack that studies the linear relations between the plaintext, the ciphertext and the secret key in an attempt to find an effective approximation of the cipher. More precisely, this attack studies the probability of the event

$$\langle \alpha \cdot x \rangle \oplus \langle \beta \cdot E_K(x) \rangle = 0,$$

where $\langle \cdot \rangle$ denotes the scalar product on $\mathbb{F}_2^n$ *i.e* $\langle a \cdot b \rangle = \bigoplus_{i=0}^{n-1} a_i b_i$, and α and β are two n-tuples, also called *masks*. The pair $(\alpha \rightsquigarrow \beta)$ is the *linear approximation* of the cipher.

Definition 3.10 (Correlation of a linear approximation)**.** *Let $f\colon \mathbb{F}^{2^n} \to \mathbb{F}^{2^n}$ be a vectorial boolean function. Assume that the masks for input x and output $f(x)$ are α and β, respectively. The* correlation of the linear approximation *is defined as*

$$C(\alpha, \beta) = 2 \times Pr[\langle \alpha \cdot x \rangle + \langle \beta \cdot f(x) \rangle = 0] - 1$$

If f is a random permutation, the probability that $\langle \alpha \cdot x \rangle \oplus \langle \beta \cdot f(x) \rangle = 0$ is close to $\frac{1}{2}$ for any pair of nonzero elements (α, β). For a keyed permutation, however, the *bias* of the linear approximation, defined by

$$\epsilon(\alpha, \beta) = C(\alpha, \beta)/2$$

can be quite high. In such case, this results in a distinguisher. In other words, both masks should be chosen such that $\Pr[\langle \alpha \cdot x \rangle \oplus \langle \beta \cdot F(x) \rangle = 1]$ is as far as possible from $\frac{1}{2}$.

Linear round-approximations (or *linear trails*) with a high bias can be constructed by combining several one-round approximations under the assumption that the individual rounds are mutually independent, in an analogous fashion to differential trails.

Let $f = f_{r-1} \circ \cdots \circ f_1 \circ f_0$ be an iterated function. Linear approximations $(\gamma_i \rightsquigarrow \gamma_{i+1})$ of a single round f_i can be concatenated into a linear trail $(\gamma_0 \rightsquigarrow \gamma_1 \rightsquigarrow \cdots \rightsquigarrow \gamma_r)$ whose correlation is estimated using the *piling-up lemma*.

Lemma 3.11 (Piling-up lemma)**.** *Let $(\gamma_0 \rightsquigarrow \gamma_1 \rightsquigarrow \cdots \rightsquigarrow \gamma_r)$ be a linear trail of an iterated permutation. Then, the correlation of the linear trail can be computed as*

$$C(\gamma_0, \gamma_r) = \prod_{i=0}^{r-1} C(\gamma_i, \gamma_{i+1}).$$

Akin to the differential case, it is possible to combine several linear trails with masks α and β into a *linear hull* [Nyb95] $(\alpha \rightsquigarrow \beta)$. Extensions of linear cryptanalysis include the *zero-correlation attack* [BR11], which is based on linear approximations with a bias equal to zero.

3.2.3 Boomerang attacks

Subsequent to the discovery of differential and linear cryptanalysis, Vaudenay developed the *decorrelation theory* [Vau98], an approach for designing block ciphers that are provably secure against basic variants of these cryptanalytic attacks, among others. In the same paper, he proposed a concrete instantiation of this construction technique, COCONUT98, which was broken a year later by Wagner using a new type of attack that he called the *boomerang attack* [Wag99].

The boomerang attack is a chosen plaintext and ciphertext attack that combines sets of four messages that maximize the probability that

$$E^{-1}(E(M_0) \oplus \delta) \oplus E^{-1}(E(M_0 \oplus \alpha) \oplus \delta) = \alpha,$$

with M_0 a random message, α and β some input and output differences. The distinguisher works as follows:

1. the attacker first picks a pair of messages (M_0, M_1) with a difference α, and asks for the corresponding ciphertexts (C_0, C_1);
2. then, they apply a difference δ to obtain two additional ciphertexts $C_2 = C_0 \oplus \delta$ and $C_3 = C_1 \oplus \delta$;

3. (C_2, C_3) are decrypted to the plaintext pair $(M_2,\ M_3)$;

4. if $M_3 \oplus M_4 = \alpha$, the set (M_0, M_1, M_2, M_3) is a *right quartet* for the distinguisher.

This is summarized in Figure 3.4.

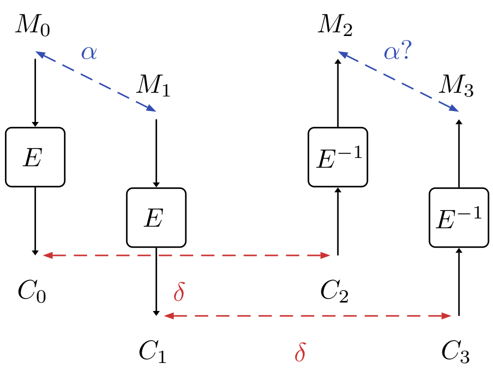


Figure 3.4: Basic boomerang distinguisher.

Probability of a right quartet. The boomerang technique is particularly effective against constructions for which finding good short differentials is easier than finding one full differential of a comparable probability. More precisely, the targeted cipher E is rewritten as the composition of two subciphers E_0 and E_1, such that there exist good differentials on each subcipher:

$$\Pr[\alpha \rightsquigarrow_{E_0} \beta] = p \text{ and } \Pr[\gamma \rightsquigarrow_{E_1} \delta] = q, \text{ where } E = E_1 \circ E_0.$$

The initial pair of plaintexts M_0 and M_1 with a difference α goes to difference β through the upper half of the cipher with probability p. The attacker obtains the corresponding ciphertexts C_0 and C_1, applies a difference δ to obtain ciphertexts $C_2 = C_0 + \delta$ and $C_3 = C_1 + \delta$, and decrypts them to plaintexts M_2 and M_3. The choice of δ is such that the difference propagates to the difference γ in the decryption direction through the lower half of the cipher with probability q. For the right quartet of texts, difference γ is created in the middle of the cipher between partial decryptions of C_2 and C_3 which propagates to the difference α in the plaintexts M_2 and M_3. The probability of the total structure is p^2q^2.

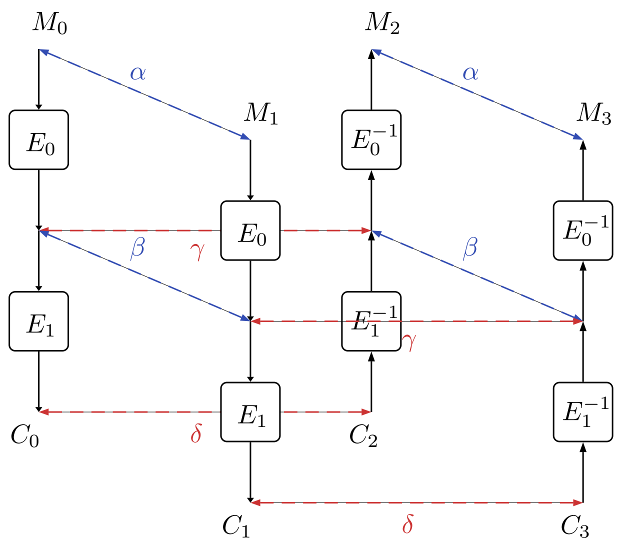


Figure 3.5: Basic boomerang distinguisher.

It was, however, observed that some dependencies can occur at the junction of the two subciphers, sometimes leading to some free rounds in the middle—these special cases are called *ladder switch* and *S-box switch* [BK09]—or incompatibilities leading to probability 0 [Mur11]. A more accurate estimation of the probability of the distinguisher was given by Dunkelman *et al.* in the so-called *Sandwich attack* [DKS10], which separates the cipher into three parts—an upper part E_0, a lower part E_1 and a middle part E_m, made of one S-box layer:

$$E = E_1 \circ E_m \circ E_0.$$

If E_m satisfies the requested differential propagation among four texts with probability r, then the probability of the boomerang distinguisher is p^2q^2r, using the same notation as previously introduced.

At Eurocrypt 2018, Cid *et al.* introduced the Boomerang Connectivity Table (BCT), a tool that applies to one S-Box to better estimate the probability of the middle round E_m of a boomerang distinguisher in a systematic way for SPNs [CHP$^+$18]. Their new table and the following works led to a refined understanding of boomerangs, and resulted in a series of improved attacks [BC18, LQSL19, ZDJ19, WP19, SQH19, LS19]. Chapter 7 presents a study that is complementary to Cid *et al.*'s work by introducing the FBCT, the Feistel counterpart of the BCT.

Variants of the boomerang attack have been proposed in several papers, including the *amplified boomerang attack* [KKS01] and the *rectangle attack* [BDK01]. Boomerang attacks can also be translated into the related-key setting [BDK05].

3.2.4 Interpolation attacks

The *interpolation attack* was introduced by Jakobsen and Knudsen [JK97] in 1997 and applied to variants of SHARK, a predecessor of RIJNDAEL which was optimized against linear and differential attacks. It is effective in the analysis of ciphers for which the round function can be written as a simple algebraic function.

Assuming that the ciphertexts c_i can be expressed as a (possibly multivariate) polynomial $p(m_i)$ of the messages m_i (or subwords of the messages) with a sufficiently small number of key-dependent coefficients, then such a polynomial can be efficiently reconstructed from a collection of plaintexts-ciphertexts pairs, using the *Lagrange interpolation formula.*

Definition 3.12 (Lagrange interpolation formula)**.** *Let $\mathcal{K}$ be a field. Given $2d$ elements $x_0, \ldots, x_{d-1}, y_0, \ldots, y_{d-1} \in \mathcal{K}$, where the x_i's are distinct, define:*

$$f(x) = \sum_{i=0}^{d-1} y_i \prod_{1 \leq j \leq f,\ j \neq i} \frac{x - x_j}{x_i - x_j},$$

then $f(x)$ is the only polynomial over $\mathcal{K}$ of degree at most $d-1$ such that $f(x_i) = y_i$ for $i = 0 \ldots d-1$.

As a result, in the simplest form of the attack, a representation equivalent to the cipher with the secret key is constructed, which allows an attacker to encrypt any plaintext message even though the exact value of the key has not been determined.

A variant of the attack that can be more efficient consists of combining it with a meet-in-the-middle approach to reduce the number of plaintexts needed. The attack can also be extended to a key recovery attack: the correctness of the guess on a round key is simply checked by applying the interpolation step on the remaining rounds. Finally, while the classical interpolation attack assumes that the polynomials exist with probability 1, Jakobsen presented a probabilistic variant in [Jak98].

3.3 Structural Attacks

As opposed to statistical attacks, which exploit the limitations of the nonlinear components of cryptosystem, structural attacks tend to focus more on its diffusion properties.

3.3.1 Integral attacks

Integral cryptanalysis, also known as the *Square attack* or the *saturation attack* [Luc02], was originally designed by Lars Knudsen and first presented together with the SQUARE cipher [DKR97], which served as basis for RIJNDAEL.

Integral cryptanalysis uses sets of chosen plaintexts with a fixed part and another that varies through all possibilities. Typically, for an n-bit block cipher, these sets contain 2^n messages that only differ in one block. The XOR of these 2^n values necessarily sums up to zero and the sum of the corresponding ciphertexts can leak information about the cipher's operation.

Inside a set, the blocks of the same index can reach four states, listed below.

- $\mathcal{P}$ (permutation): the block can take all possible values;
- $\mathcal{C}$ (constant): the block has a fixed value;
- $\mathcal{S}$ (sum): the sum of all blocks equals zero; if a block reaches a state $\mathcal{P}$ or $\mathcal{C}$, the sum also equals zero, thus $\mathcal{S}$ is only used in cases where $\mathcal{P}$ or $\mathcal{C}$ are unknown;
- '?': nothing can be said about the block.

A bijective function preserves the $\mathcal{P}$ and $\mathcal{C}$ states, however, $\mathcal{S}$ becomes '?'. $\mathcal{P} \oplus \mathcal{C}$ remains $\mathcal{P}$ since all elements of the block in state $\mathcal{P}$ are simply XORed with the same constants. On the other hand, $\mathcal{P} \oplus \mathcal{P}$ does not automatically yield $\mathcal{P}$, however, $\mathcal{S}$ is preserved because of the linearity of the sum. Figure 3.6 illustrates these behaviors.

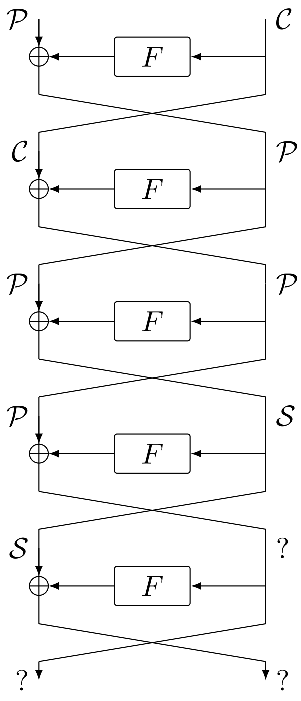


Figure 3.6: An integral characteristic on a Feistel network with a nonlinear component F.

Integral attacks can be extended to higher order integrals. An integral of order ℓ requires $2^{n\ell}$ plaintexts that take all possible 2^n values on ℓ blocks. Zhang *et al.* [ZSW+12] introduced a generic method to construct integral distinguishers for block ciphers.

Division property. Todo further investigated integral cryptanalysis and proposed the so-called *division property* [Tod15b] at Eurocrypt 2015, a more fine-grained technique to find integral characteristics than can be seen as a generalization of both integral and higher-order differential properties. The same author later applied this tool to mount the first theoretical attack on the full version of MISTY1 [Tod15a]. The division property exploits the fact that some components of a block cipher may be of a lower degree, depending on the inputs. More precisely, it can be used to predict wether the monomials of the *Algebraic Normal Form* (ANF) of a block cipher sum up to zero when restricted to a specific set of plaintexts. Let $u = (u_1, \dots, u_n)$ be a vector of $\mathbb{F}_2^n$ and let x^u denote the coordinate product

$$x = (x_1, \dots, x_n) \mapsto \prod_{i=0}^{n} x_i^{u_i},$$

then a set $X \subseteq \mathbb{F}_2^n$ has the division property $\mathcal{D}_k^n$, for some $1 \leq k \leq n$, if the sum over all vectors x in X of the product x^u equals 0, for all vectors u with Hamming weight strictly smaller than k, *i.e.*

$$\bigoplus_{x \in X} x^u = 0 \text{ for all } u \in \mathbb{F}_2^n \text{ such that } wt(u) < k.$$

In short, when encrypting a set of plaintexts with a defined structure—*e.g.*, some bits can be fixed to a constant value— some bits of the resulting ciphertexts might be *balanced*, *i.e.* they sum up to zero with probability 1 when going through the entire set of ciphertexts. The division property is thus a means to track which bits of the plaintext should be fixed in order to obtain such an output. This technique has sparked much interest from the community, leading to many follow-up results [ZW15, TM16, BC16, XZBL16, SWW17, EKKT19, DFL19].

3.3.2 Impossible differential attacks

Impossible differential attacks were first described by Knudsen in the AES competition proposal Deal [Knu98] but they were formalized by Biham, Biryukov and Shamir in [BBS99] at Crypto 1999. This type of attack is based on the fact that some differentials cannot be generated by a permutation and thus, have a zero probability: if for one guessed value of a key, the attacker is able to exhibit a pair of messages that satisfies this *impossible* differential, then the key can be discarded, therefore reducing the keyspace.

In order to find an *impossible differential characteristic*, the authors of [BBS99] introduced the *miss-in-the-middle* method, which consists of finding a differential with probability 1 on each half of the cipher but that cannot occur together *i.e* the output difference of the first one cannot transition to the input difference of the second one. This method was refined in subsequent papers [KHS+03, LWLG09].

3.4 Automatic tools

Studying the security of a cryptographic primitive can be a tedious task and several methods have been investigated in attempts to automate the process [BN10, BDF11, MWGP11, SHW+14, BV14, LWZ16, DF16, GLMS18, SGL+17, HW19]. These include Mixed-Integer Linear Programming (MILP), Boolean Satisfiability (SAT) and Constraint Programming (CP). Such generic solvers are an appealing alternative to dedicated approaches as a cryptanalytic problem simply needs be stated as a model—by means of linear inequalities for MILP, Boolean clauses for SAT, and constraints for CP—which is then automatically solved. The three aforementioned methods have been used to derive differential bounds and find new characteristics for Skinny; the results are discussed in Chapter 6.

3.4.1 Mixed-Integer Linear Programming (MILP)

Mixed-Integer Linear Programming was first adopted by Borghoff *et al.* to study Bivium, a simplified version of Trivium [BKS09], and has since been used to tackle many symmetric cryptanalysis problems [MWGP11, SHW+14, XZBL16, FWG+16, SWLW16, ZR17, ST17]. For instance, Sun *et al.*applied MILP to find optimal differential characteristics against bit-oriented block ciphers such as Simon, Present or LBlock [SHW+14]. In [ST17], this approach was used to search for impossible differentials, and the designers Skinny [BJK+16] relied on this tool as well to provide security bounds.

Milp models can only contain linear inequalities and describing Boolean operations can be complicated. The problems are first seen as general linear programming problems over the real numbers, then by Branch and Cut, illegal branches are ruled then the solutions are limited to 0-1 integers. Consequently, nonlinear operators need to be transformed into sets of linear inequalities. However, the resulting MILP model may always not scale well, as the performance depends heavily on the background and structure of the underlying problem. For instance, in [AST+17], authors introduce a MILP model of the nonlinear part of a block cipher to find differential trails. Some of their results on SKINNY-128 required about 15 days to find the characteristics.

3.4.2 Boolean Satisfiability problem (SAT)

The Boolean Satisfiability problem (SAT) considers whether there exists a valid assignment to Boolean variables satisfying a given set of Boolean clauses. SAT and Satisfiability Modulo Theories (SMT) can be effective in solving cryptanalysis problems. For example, Mouha and Preneel used a SAT solver to search for optimal differential characteristics for ARX ciphers [MP13]. In [SWW17], Sun *et al.* proposed a SAT/SMT model to search for division properties on ARX and word-based block ciphers, and in particular, they showed that this approached scaled better than MILP when applied to SHACAL-2. In [KLT15] the authors derived SAT/SMT models to study the differential and linear behaviors of SIMON-like round functions. Moreover, they computed optimal differential and linear characteristics for SIMON using the CryptoMiniSat [SNC09] solver—a solver that is well-suited for cryptanalysis problems since XOR operations can be easily modeled. Similarly to MILP, nonlinear operations cannot be described in a straightforward manner by means of clauses. In [Laf18], Lafitte proposed a method to encode a relation associated with a nonlinear operation into a set of clauses; Sun *et al.* [SWW18] were then able to reduce the number of clauses for this kind of encoding by using a similar approach as in [AST+17].

3.4.3 Constraint Programming (CP)

CP models have been used in [LCM+17] to model algebraic side-channel attacks on the AES and in [RSM+11] to design the nonlinear part of a block cipher with good properties. Regarding differential cryptanalysis, Gérault *et al.* proposed CP models to find related-key differential characteristics for the AES [GMS16, GLMS18], MIDORI [GL16], and SKINNY [SGL+17], showing that CP solvers can perform better than dedicated approaches on such problems. The models for the AES were refined in [GLMS20]: the authors added new constraints to detect inconsistencies sooner and developed a new two-step solving process that differs from the one previously used in [FJP13, BN10, GMS16]. Thanks to these improvements, optimal related-key differential characteristics were computed for all instance of the AES in a few hours, while it took more than two months of CPU time to solve the hardest instance with the models from [GLMS18].

Part II

Contribution to the NIST Lightweight Cryptography Competition

4

LILLIPUT-AE: a NIST Proposal

All hail Lilliput! All hail Lilliput!

GENERAL EDWARD EDWARDIAN

This chapter presents LILLIPUT-AE [ABC+18], a new Authenticated Encryption with Associated Data (AEAD) scheme that uses as primitive the LILLIPUT-TBC tweakable block cipher—which is itself based on the classical LILLIPUT block cipher presented in [BFMT16] (see Section 2.4.4), with a modified tweakey schedule. LILLIPUT-AE is the result of a collaborative effort with Alexandre Adomnicai, Thierry Berger, Christophe Clavier, Julien Francq, Virginie Lallemand, Kévin Le Gouguec, Marine Minier, Léo Reynaud and Gaël Thomas as part of the FUI 23 PACLIDO[1] project, following NIST's call for algorithms to be considered for lightweight cryptographic standards[2]. It was submitted to the standardization process in February 2019 and selected as Round 1 candidate in April 2019. My contributions to this proposal concerned some design aspects. To be more precise, I participated the design of the tweakey schedule and the choice of the modes. Even though our proposal is now based on the LILLIPUT block cipher, earlier in the design process, our intention was to design our own primitive, while taking inspiration from LILLIPUT. As such, I was tasked with studying other lightweight constructions of Extended Generalized Feistel Networks (EGFNs)—some of this work is presented in Chapter 8 of this thesis. For the sake of completeness, implementation results are still presented in this chapter.

[1]Protocoles et Algorithmes Cryptographiques Légers pour l'Internet Des Objets. http://paclido.fr/lilliput-ae/

[2]https://csrc.nist.gov/News/2018/requesting-nominations-for-lightweight-crypto-algs

4.1 Introduction

LILLIPUT-AE defines two authenticated encryption modes: LILLIPUT-I and LILLIPUT-II based respectively on ΘCB3 [KR11b] and SCT-2, two modes that are used in DEOXYS [JNPS16]. The ΘCB3 mode is a nonce-respecting mode whereas SCT-2 is a nonce-misuse resistant mode. From those two authenticated encryption modes—LILLIPUT-I and LILLIPUT-II—several sets of parameters are derived. These parameters conform with the NIST Submission Requirements and Evaluation Criteria for the Lightweight Cryptography Standardization Process. The primary member is LILLIPUT-II-128.

As shown in the next sections, LILLIPUT-AE is an authenticated encryption scheme that provides full 128-bit, 192-bit or 256-bit security level. It performs well in software and also in hardware. Moreover, the underlying block cipher LILLIPUT has been extensively studied by the cryptographic community [ST18, MNV18, ST16] and so far, no weakness has been exhibited for the full version of LILLIPUT.

We initially attempted to find lighter EGFN constructions [BMT14] inspired by LILLIPUT. Some of the results are presented in Chapter 8. While some schemes with a smaller XOR count could be instantiated, it turns out that the original structure of LILLIPUT still provided the best security bounds. Moreover, LILLIPUT is a well-studied block cipher and thus, extending it to an 8-bit oriented version and combining it with a mode with good performances and with reinforced security seemed like a good answer regarding both efficiency and security to the expectations of the NIST standardization process.

Main features of LILLIPUT-AE. From our point of view, LILLIPUT-AE brings many advantages:

- It is based on building schemes (authenticated encryption modes, encryption process) that have been significantly studied by the cryptographic community. Moreover, the security of these blocks has been strengthened by modifiying some parameters (*e.g.*, more secure S-box and tweakey schedule).
- Its primary member is a nonce-misuse resistant mode, which allows an easier management of cryptographic components deployed on the field.
- Its software implementations on 8-bit (*e.g.*, Atmel AVR ATmega128 microcontrollers) and 16-bit (*e.g.*, Texas Instruments MSP430F1611 microcontrollers) platforms are very competitive. In terms of execution time (which relates to power consumption), and for 128-bit keys, LILLIPUT-AE is comparable to lightweight winners of the CAESAR competition [CAE], ACORN and ASCON, on 8-bit platforms, and is significantly faster on 16-bit platforms.

- Its hardware implementations on FPGA platforms (*e.g.*, Xilinx Spartan-6) are more compact than Acorn and Ascon. Moreover, straightforward ASIC implementations of Lilliput-AE lead to at most around 5000 Gate Equivalents (GEs) for its maximum parameter sizes. Serial implementations will decrease this figure down to 4000 GEs or 3000 GEs depending on the parameter sizes, which is equivalent to serial implementations of plain Aes without authentication mode.

- Some degrees of freedom are given to the implementers of Lilliput-AE: for some operations (*e.g.*, in the tweakey schedule), they can trade code size for RAM usage and execution time. Some operations can also be tabulated to accelerate their computation.

- The design facilitates side-channel protection: in particular, the S-box of Lilliput-AE has been chosen to optimize its cost in threshold implementations.

- A first fault injection analysis of Lilliput-AE shows that faulting 7 rounds or more from the end of the algorithm requires injecting too many faults (say millions) to be practical. A cautious recommendation is then to protect the last 7 rounds of Lilliput-AE against fault injection, which leads to a 22% execution time overhead if a straightforward duplication countermeasure is implemented.

Organization of the chapter. In Section 4.2, we provide the complete specifications of Lilliput-AE including the two considered modes of authenticated encryption with associated data Lilliput-I and Lilliput-II (Section 4.2.2) and the tweakable block cipher Lilliput-TBC with its particular tweakey schedule (Section 4.2.3). In Section 4.3, we detail our design choices: first for the modes (Section 4.3.1) and second for the tweakable block cipher (Section 4.3.2). We also perform an extensive security analysis of these two parts in Section 4.3.3 and in Section 4.3.4. In Section 4.4, we give the implementation results we obtain for both software platforms and hardware platforms. Finally, in Section 4.6, we address the changes made between the version that was originally submitted to NIST and the one that is presented in this thesis, Lilliput-AE v1.1.

4.2 Specifications

This section presents the full specifications of our new Authenticated Encryption with Associated Data (AEAD) scheme Lilliput-AE.

After introducing notations and the sets of parameters, we introduce in Section 4.2.2 the two particular authenticated encryption modes: Lilliput-I based on the nonce-respecting mode ΘCB3 and Lilliput-II based on the nonce-misuse resistant mode SCT-2.

Then, in Section 4.2.3, we introduce our tweakable block cipher Lilliput-TBC used in both Lilliput-I and Lilliput-II.

Notations. Let us introduce the following notations: K will represent the key of length k bits, P the plaintext of length n bits, T the tweak of length t bits and we denote by $E_K(T, P)$ the ciphering process using the tweakable block cipher E_K^T.

The concatenation operation at binary level is represented by $||$ and $pad10^*$ is the function that applies the 10^* padding on n bits, *i.e.* $pad10^*(X) = X||1||0^{n-|X|-1}$ when $|X| < n$. For an empty string ϵ, the 10^* padding will not add any bit: $pad10^*(\epsilon) = \epsilon$. The truncation of the word X to the first i bits is given by $\lceil X \rceil_i$, and the truncation to the last i bits by $\lfloor X \rfloor_i$. To emphasize a string X is of length n, we may write it $X_{(n)}$. We denote by $\gg i$ and $\ll i$ respectively the right and left shifts of i bits, and by $\ggg i$ and $\lll i$ the right and left rotations of i bits.

We will also denote by $S^{\ggg i}$ and $S^{\lll i}$ the binary matrices of size 8×8 corresponding to a right shift by i bits positions or a left shift by i bits positions respectively. More precisely, and for example,

$$S^{\ggg 1} = \begin{pmatrix} 0&0&0&0&0&0&0&0\\ 1&0&0&0&0&0&0&0\\ 0&1&0&0&0&0&0&0\\ 0&0&1&0&0&0&0&0\\ 0&0&0&1&0&0&0&0\\ 0&0&0&0&1&0&0&0\\ 0&0&0&0&0&1&0&0\\ 0&0&0&0&0&0&1&0 \end{pmatrix} \text{ and } S^{\lll 1} = \begin{pmatrix} 0&1&0&0&0&0&0&0\\ 0&0&1&0&0&0&0&0\\ 0&0&0&1&0&0&0&0\\ 0&0&0&0&1&0&0&0\\ 0&0&0&0&0&1&0&0\\ 0&0&0&0&0&0&1&0\\ 0&0&0&0&0&0&0&1\\ 0&0&0&0&0&0&0&0 \end{pmatrix}$$

The encryption part $\mathcal{E}$ takes as input a variable-length plaintext M (with $m = |M|$ bits), a variable-length associated data A (with $a = |A|$ bits), a fixed-length public nonce N and a k-bit key K (we deliberately used the same letter K to represent the key in the authenticated encryption scheme and the one in the tweakable block cipher, since they always refer to the same object). It outputs an m-bit ciphertext C and a τ-bit tag, denoted `tag` (with $\tau \in [0, \cdots, n]$) *i.e.* $(C, \texttt{tag}) = \mathcal{E}_K(N, A, M)$.

The verification/decryption part $\mathcal{D}$ takes as input a variable-length ciphertext C (with $m = |C|$), a τ-bit tag, denoted `tag` (with $\tau \in [0, \cdots, n]$), a variable-length associated data A (with $a = |A|$), a fixed-length public nonce N and a k-bit key K. It outputs either an error string $\perp$ to signify that the verification has failed, or an m-bit string $M = \mathcal{D}_K(N, A, C, \texttt{tag})$ when the tag is valid.

The maximum message length (in n-bit blocks) is denoted max_l and the maximum number of messages that can be handled with the same key is denoted max_m (the same limitation applies to the associated data material).

4.2.1 Recommended Parameters

We derive our scheme LILLIPUT-AE into two authenticated encryption modes: LILLIPUT-I and LILLIPUT-II. LILLIPUT-I is a nonce-respecting mode corresponding with ΘCB3 and LILLIPUT-II is a nonce-misuse resistant mode corresponding with SCT-2.

The recommended parameter sets for all variants of these modes is given in Table 4.1. These parameters have been chosen according to the internal tweakable block cipher LILLIPUT-TBC.

Name	k	t	n	$\lvert N\rvert$	τ
LILLIPUT-I-128	128	192	128	120	128
LILLIPUT-I-192	192	192	128	120	128
LILLIPUT-I-256	256	192	128	120	128
LILLIPUT-II-128	**128**	**128**	**128**	**120**	**128**
LILLIPUT-II-192	192	128	128	120	128
LILLIPUT-II-256	256	128	128	120	128

Table 4.1: Recommended parameter sets for LILLIPUT-AE. Our primary member LILLIPUT-II-128 is in bold notation.

For both variants, $max_m = 2^{|N|} = 2^{120}$ bits. However, max_l is dependent on the tweak input and thus differs from one variant to the other:

- in the encryption part of Lilliput-I, the tweak is a concatenation of a 4-bit prefix, the nonce N and the index of the message block, thus $max_l = 2^{t-4-|N|}$ blocks.
- in the encryption part of Lilliput-II, the tweak is a concatenation of a 4-bit prefix and the index of the message block, thus $max_l = 2^{t-4}$ blocks.

As a result, the maximum message length in bytes is 2^{72} bytes for Lilliput-I and 2^{128} bytes for Lilliput-II.

4.2.2 The Lilliput-AE Authenticated Encryption scheme

In this section, we describe the authenticated encryption modes that are used in our proposal Lilliput-AE. These mode variants are similar to the two modes described in Deoxys [JNPS16]:

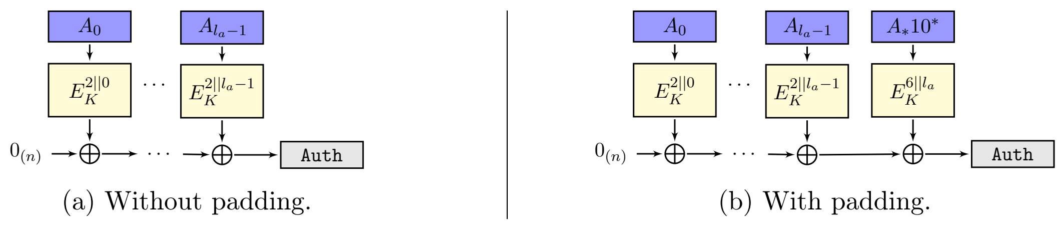
- Lilliput-I (Section 4.2.2): in this nonce-respecting variant, the same nonce N is expected to never be used twice with the same key for encryption. $\mathcal{E}^I$ denotes the encryption part and $\mathcal{D}^I$ the verification/decryption part.
- Lilliput-II (Section 4.2.2): in this variant, a nonce N may be reused with the same key for encryption. $\mathcal{E}^{II}$ denotes the encryption part and $\mathcal{D}^{II}$ the verification/decryption part.

As stated previously, 4-bit prefixes are used for the tweak input to separate the various types of encryption/authentication blocks, akin to what has been done in Deoxys [JNPS16].

Nonce-Respecting Mode: ΘCB3

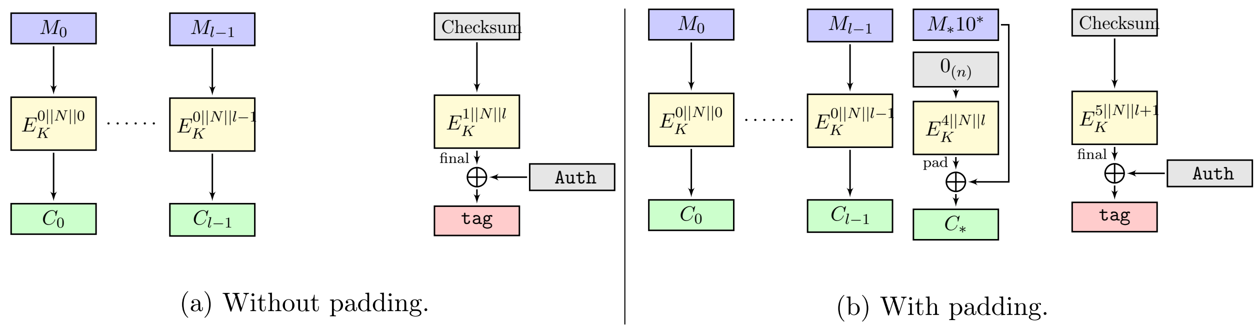
This scheme follows the ΘCB3 framework [KR11b] and therefore directly benefits from this framework's proof of security regarding authentication and privacy. In this mode, the tweak length is 192 bits. The encryption algorithm $\mathcal{E}^I$ is given in Algorithm 1 while the verification/decryption algorithm $\mathcal{D}^I$ is given in Algorithm 2.

If the length of the associated data is not a multiple of the block size, the final block is padded with the 10^* padding, as depicted in Figure 4.1. The same applies for the message and the ciphertext as shown in Figure 4.2 and Figure 4.3.



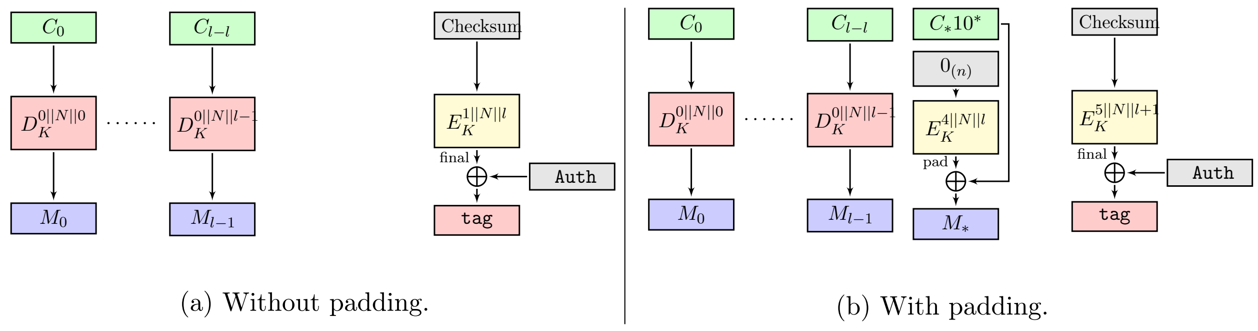
(a) Without padding. (b) With padding.

Figure 4.1: Handling of the associated data in the nonce-respecting mode.



(a) Without padding. (b) With padding.

Figure 4.2: Message processing for the nonce-respecting mode.



(a) Without padding. (b) With padding.

Figure 4.3: Ciphertext processing for the nonce-respecting mode.

Algorithm 1 The encryption algorithm $\mathcal{E}_K^{\mathrm{I}}(N, A, M)$.
In the tweak inputs, the value N is encoded on 120 bits, the integer values j and l are encoded on 68 bits, while the integer values i and l_a are encoded on 188 bits.

```
/* Associated data */  $A_0||\cdots||A_{l_a-1}||A_* \leftarrow A$ where each $|A_i| = n$ and $|A_*| < n$  Auth $\leftarrow 0_{(n)}$
for $i = 0$ to $l_a - 1$ do
  | Auth $\leftarrow$ Auth $\oplus E_K(0010||i, A_i)$
end
if $A_* \neq \epsilon$ then
  | Auth $\leftarrow$ Auth $\oplus E_K(0110||l_a, pad10^*(A_*))$
end
  /* Message */  $M_0||\cdots||M_{l-1}||M_* \leftarrow M$ where each $|M_j| = n$ and $|M_*| < n$  Checksum $\leftarrow 0_{(n)}$
for $j = 0$ to $l - 1$ do
  | Checksum $\leftarrow$ Checksum $\oplus M_j$  $C_j \leftarrow E_K(0000||N||j, M_j)$
end
if $M_* = \epsilon$ then
  | Final $\leftarrow E_K(0001||N||l$, Checksum)  $C_* \leftarrow \epsilon$
else
  | Checksum $\leftarrow$ Checksum $\oplus pad10^*(M_*)$  Pad $\leftarrow E_K(0100||N||l, 0_{(n)})$  $C_* \leftarrow M_* \oplus \lceil \mathrm{Pad} \rceil_{|M_*|}$
  | Final $\leftarrow E_K(0101||N||l + 1$, Checksum)
end
  /* Tag generation */  tag $\leftarrow$ Final $\oplus$ Auth  return $(C_0||\cdots||C_{l-1}||C_*, \texttt{tag})$
```

Nonce-Misuse Resistant Mode

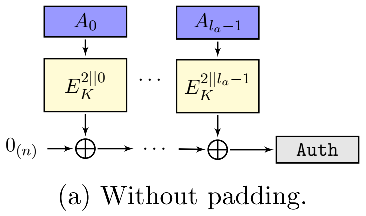
This scheme is the variant of SCT introduced in DEOXYS [JNPS16]: SCT-2. In this mode, the tweak length is 128 bits while the size of the nonce N remains unchanged and is 120 bits. The

Algorithm 2 The verification/decryption algorithm $\mathcal{D}^{\mathrm{I}}_K(N, A, C, \mathtt{tag})$.
In the tweak inputs, the value N is encoded on 120 bits, the integer values j and l are encoded on 68 bits, while the integer values i and l_a are encoded on 188 bits.

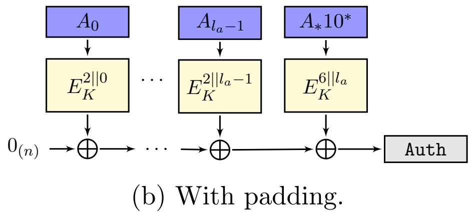
```
/* Associated data */  $A_0||\cdots||A_{l_a-1}||A_* \leftarrow A$ where each $|A_i| = n$ and $|A_*| < n$  Auth $\leftarrow 0_{(n)}$
for $i = 0$ to $l_a - 1$ do
    Auth $\leftarrow$ Auth $\oplus E_K(0010||i, A_i)$
end
if $A_* \neq \epsilon$ then
    Auth $\leftarrow$ Auth $\oplus E_K(0110||l_a, pad10^*(A_*))$
end
/* Ciphertext */  $C_0||\cdots||C_{l-1}||C_* \leftarrow C$ where each $|C_j| = n$ and $|C_*| < n$  Checksum $\leftarrow 0_{(n)}$
for $j = 0$ to $l - 1$ do
    $M_j \leftarrow D_K(0000||N||j, C_j)$  Checksum $\leftarrow$ Checksum $\oplus M_j$
end
if $C_* = \epsilon$ then
    Final $\leftarrow E_K(0001||N||l,$ Checksum$)$  $M_* \leftarrow \epsilon$
else
    Pad $\leftarrow E_K(0100||N||l, 0_{(n)})$  $M_* \leftarrow C_* \oplus \lceil \mathrm{Pad} \rceil_{|C_*|}$  Checksum $\leftarrow$ Checksum $\oplus pad10^*(M_*)$
    Final $\leftarrow E_K(0101||N||l + 1,$ Checksum$)$
end
/* Tag generation */  tag$'$ $\leftarrow$ Final $\oplus$ Auth  if tag$'$ = tag then
    return $(M_0||\cdots||M_{l-1}||M_*)$
else
    return $\perp$
end
```

encryption algorithm $\mathcal{E}^{II}$ is given in Algorithm 3 while the verification/decryption algorithm $\mathcal{D}^{II}$ is given in Algorithm 4.

The associated data is processed as in the previous variant, as depicted in Figure 4.4. The processing of the message is shown in Figure 4.5 and Figure 4.6 and decryption is shown in Figure 4.7.



(a) Without padding.



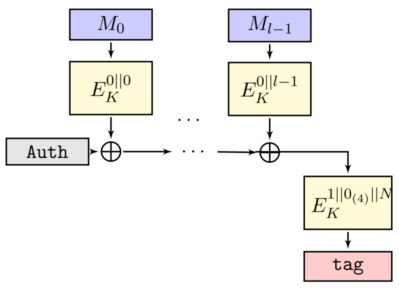
(b) With padding.

Figure 4.4: Handling of the associated data in the nonce-misuse resistant mode.

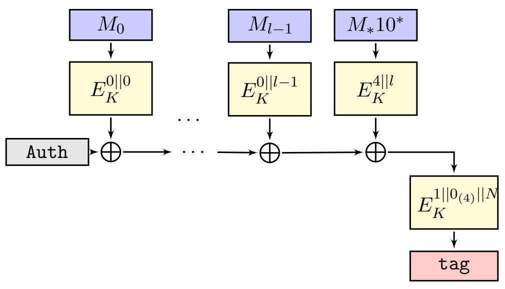
4.2.3 The Lilliput-TBC Tweakable Block Cipher

In this section we present our dedicated lightweight Tweakable Block Cipher Lilliput-TBC that is based on the EGFN [BMT14] described in Figure 4.8.

Lilliput-TBC is composed of 6 variants depending on the key lengths (possible key lengths are equal to 128, 192 and 256 bits) and on the tweak lengths (possible tweak lengths are equal to

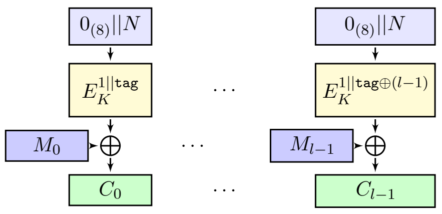


(a) Without padding.

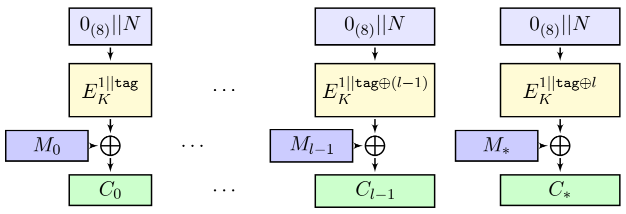


(b) With padding.

Figure 4.5: Message processing in the authentication part of the nonce-misuse resistant mode.

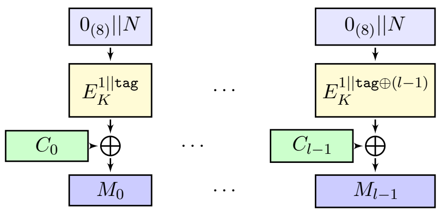


(a) Without padding.

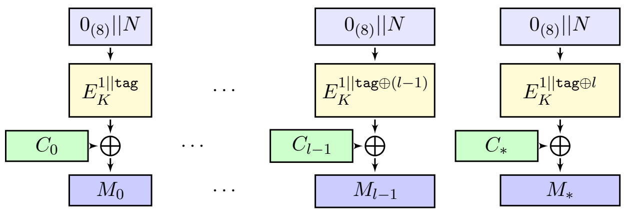


(b) With padding.

Figure 4.6: Message processing in the encryption part of the nonce-misuse resistant mode.



(a) Without padding.



(b) With padding.

Figure 4.7: Ciphertext processing in the decryption part of the nonce-misuse resistant mode.

128 or 192 bits). The different parameters for those variants are specified in Table 4.2. Lilliput-TBC-I for the three possible key lengths and a tweak length equal to 192 bits will be used in the mode Lilliput-I and Lilliput-TBC-II for the three possible key lengths and a tweak length equal to 128 bits will be used in the mode Lilliput-II.

Name	k	t	Nb of rounds r
Lilliput-TBC-I-128	128	192	32
Lilliput-TBC-I-192	192	192	36
Lilliput-TBC-I-256	256	192	42
Lilliput-TBC-II-128	128	128	32
Lilliput-TBC-II-192	192	128	36
Lilliput-TBC-II-256	256	128	42

Table 4.2: Recommended parameter sets for Lilliput-TBC.

Algorithm 3 The encryption algorithm $\mathcal{E}_K^{\mathrm{II}}(N, A, M)$.
In the tweak inputs, the integer values i, j, l and l_a are encoded on 124 bits. Moreover, $\texttt{tag} \oplus j$ values are encoded on 127 bits (the most significant bit is truncated since $|\texttt{tag}| = \tau$).

```
/* Associated data */  $A_0||\cdots||A_{l_a-1}||A_* \leftarrow A$ where each $|A_i| = n$ and $|A_*| < n$  $\text{Auth} \leftarrow 0_{(n)}$
for $i = 0$ to $l_a - 1$ do
    $\text{Auth} \leftarrow \text{Auth} \oplus E_K(0010||i, A_i)$
end
if $A_* \neq \epsilon$ then
    $\text{Auth} \leftarrow \text{Auth} \oplus E_K(0110||l_a, pad10^*(A_*))$
end
 /* Message authentication and tag generation */  $M_0||\cdots||M_{l-1}||M_* \leftarrow M$ where each $|M_j| = n$
and $|M_*| < n$  $\texttt{tag} \leftarrow \text{Auth}$
for $j = 0$ to $l - 1$ do
    $\texttt{tag} \leftarrow \texttt{tag} \oplus E_K(0000||j, M_j)$
end
if $M_* \neq \epsilon$ then
    $\texttt{tag} \leftarrow \texttt{tag} \oplus E_K(0100||l, pad10^*(M_*))$
end
$\texttt{tag} \leftarrow E_K(0001||0^4||N, \texttt{tag})$    /* Message encryption */   for $j = 0$ to $l - 1$ do
    $C_j \leftarrow M_j \oplus E_K(1||\texttt{tag} \oplus j, 0^8||N)$
end
if $M_* \neq \epsilon$ then
    $C_* \leftarrow M_* \oplus \lceil E_K(1||\texttt{tag} \oplus l, 0^8||N) \rceil_{|M_*|}$
else
    $C_* \leftarrow \epsilon$
end
 return $(C_0||\cdots||C_{l-1}||C_*, \texttt{tag})$
```

Encryption Process

Lilliput-TBC is a 128-bit tweakable block cipher with key sizes of 128, 192 or 256 bits and tweak sizes of 128 or 192 bits. The whole encryption process is depicted in Figure 4.9. As previously explained, Lilliput-TBC uses an Extended Generalized Feistel Network (EGFN) with a 128-bit state and a round function acting at byte level. The state X is seen as 16 bytes, denoted $X_{15}, \cdots, X_0$. In its 128-bit key version, the cipher is composed of $r = 32$ rounds, *i.e.* 32 repetitions of a single EGFN called `OneRoundEGFN`, depicted in Figure 4.8. Each F_j for j from 0 to 7 is defined as $F_j = S(X_j \oplus RTK_j^i)$ where S is an S-box that acts at byte level and RTK_j^i is the byte of position j of the 64-bit subtweakey RTK^i of round i. The 32 64-bit subtweakeys RTK^i are generated from the master key and the tweak using the tweakey schedule.

In more details, the round function denoted `OneRoundEGFN` in Figure 4.9 is composed of a layer of non linear components called `NonLinearLayer` for confusion; a new layer called `LinearLayer` in [BMT14] that represent a linear layer made of linear components applied in a Feistel way; and a block-wise permutation called `PermutationLayer` for diffusion. All three layers act at byte level on the EGFN state X and together constitute one iteration of the EGFN, as shown in Figure 4.8.

Note that with this new layer `LinearLayer`, it is possible to shuffle blocks better than what was possible using the block-wise permutation only of a classical Feistel scheme, while preserving the self-invertibility of the scheme.

Algorithm 4 The verification/decryption algorithm $\mathcal{D}_K^{\text{II}}(N, A, C, \texttt{tag})$.
In the tweak inputs, the integer values i, j, l and l_a are encoded on 124 bits. Moreover, $\texttt{tag} \oplus j$ values are encoded on 127 bits (the most significant bit is truncated since $|\texttt{tag}| = \tau$).

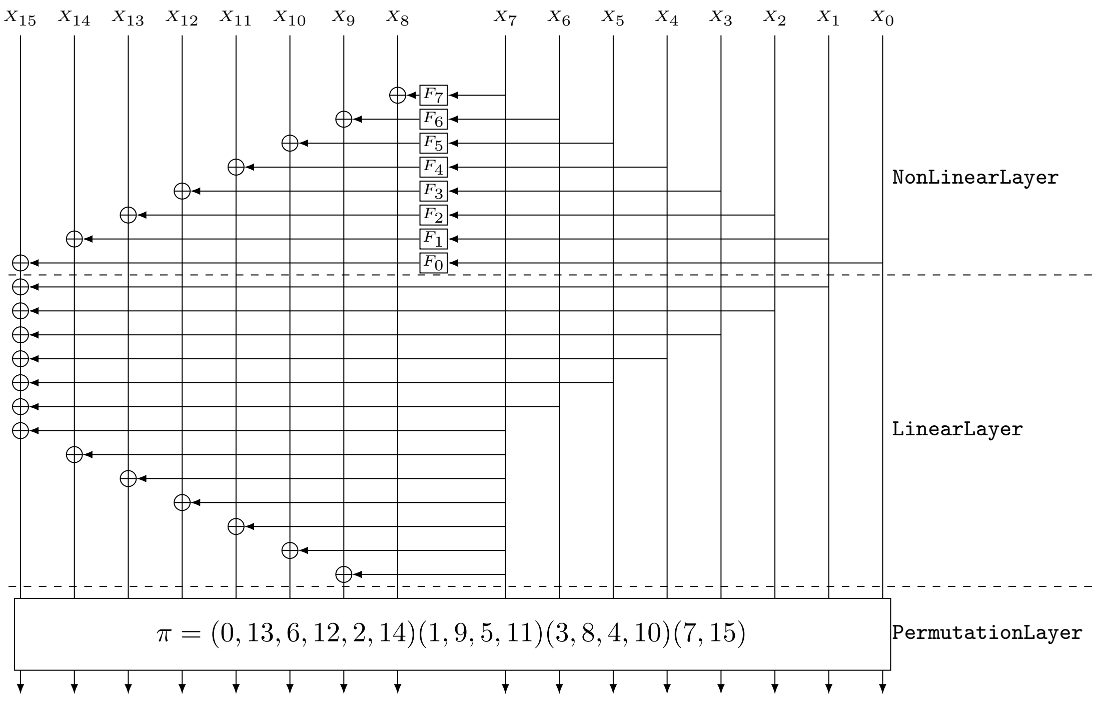
```
/* Message decryption */  $C_0||\cdots||C_{l-1}||C_* \leftarrow C$ where each $|C_j| = n$ and $|C_*| < n$  for $j = 0$ to $l-1$ do
    $M_j \leftarrow C_j \oplus E_K(1||\texttt{tag} \oplus j, 0^8||N)$
end
if $C_* \neq \epsilon$ then
    $M_* \leftarrow C_* \oplus \lceil E_K(1||\texttt{tag} \oplus l, 0^8||N)\rceil_{|C_*|}$
else
    $M_* \leftarrow \epsilon$
end
 /* Associated data */  $A_0||\cdots||A_{l_a-1}||A_* \leftarrow A$ where each $|A_i| = n$ and $|A_*| < n$  Auth $\leftarrow 0_{(n)}$
for $i = 0$ to $l_a - 1$ do
    Auth $\leftarrow$ Auth $\oplus E_K(0010||i, A_i)$
end
if $A_* \neq \epsilon$ then
    Auth $\leftarrow$ Auth $\oplus E_K(0110||l_a, pad10^*(A_*))$
end
 /* Message authentication and tag generation */  $M_0||\cdots||M_{l-1}||M_* \leftarrow M$ where each $|M_j| = n$ and $|M_*| < n$  $\texttt{tag}' \leftarrow$ Auth
for $j = 0$ to $l - 1$ do
    $\texttt{tag}' \leftarrow \texttt{tag}' \oplus E_K(0000||j, M_j)$
end
if $M_* \neq \epsilon$ then
    $\texttt{tag}' \leftarrow \texttt{tag}' \oplus E_K(0100||l, pad10^*(M_*))$
end
$\texttt{tag}' \leftarrow E_K(0001||0^4||N, \texttt{tag}')$   /* Tag verification */  if $\texttt{tag}' = \texttt{tag}$ then
    return $(M_0||\cdots||M_{l-1}||M_*)$
else
    return $\perp$
end
```

Note that the last round skips the `PermutationLayer` for involution reasons.
For the 192-bit and 256-bit key versions, the number of rounds r is 36 and 42 respectively.

Overview of the EGFN round function The particular EGFN we use in LILLIPUT-TBC with $k = 16$ blocks is depicted in Figure 4.8.

In more details, `OneRoundEGFN` is composed of:

- `NonLinearLayer`: It is the non-linear part of the EGFN and is made of 8 parallel updates of the EGFN state. Each F_j for j from 0 to 7 is defined as $F_j = S(X_j \oplus RTK_j^i)$ where S is an S-box that acts at byte level given in Table 4.4 and RTK_j^i is the byte of position j of the 64-bit subtweakey RTK^i of round i.
- `LinearLayer`: It aims at providing quick diffusion between bytes and consists in XORing some bytes to some other bytes. More precisely, as depicted in Figure 4.8, blocks X_1 to X_6 are xored to block X_{15}, and block X_7 is xored to blocks X_9 to X_{15}.



X_{15} X_{14} X_{13} X_{12} X_{11} X_{10} X_9 X_8 X_7 X_6 X_5 X_4 X_3 X_2 X_1 X_0
F_7 F_6 F_5 F_4 F_3 F_2 F_1 F_0
NonLinearLayer
LinearLayer
$\pi = (0,13,6,12,2,14)(1,9,5,11)(3,8,4,10)(7,15)$
PermutationLayer

Figure 4.8: The EGFN used in LILLIPUT-TBC that reaches full diffusion in $d = 4$ rounds. The permutation π is given as a product of cycles and can also be found in Table 4.3.

- `PermutationLayer`: It consists in applying the permutation π given in Table 4.3 to the bytes.

The permutation π used in `PermutationLayer` The permutation π is given in Table 4.3. It has been chosen to maximize the number of active S-boxes on 18, 19 and 20 rounds as it will be shown in Section 4.3.4. For each round $i \in \{0, \cdots, r-1\}$, let us denote Y^i the output at round i after the transformations `NonLinearLayer` and `LinearLayer` with $Y^i = (Y^i_{15}, \cdots, Y^i_0)$ its byte representation, *i.e.* $Y^i = (Y^i_{15}, \cdots, Y^i_0) = \texttt{LinearLayer}(\texttt{NonLinearLayer}(X^i))$. Then, the `PermutationLayer` is applied on Y^i in the following way:

$$\forall i \in \{1, \cdots, r-2\}, \forall j \{0, \cdots, 15\} \in \quad X^i_{\pi(j)} = Y^{i-1}_j.$$

Table 4.3: Block permutation π used in encryption mode and its inverse π^{-1} used in decryption mode.

i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
$\pi(i)$	13	9	14	8	10	11	12	15	4	5	3	1	2	6	0	7
$\pi^{-1}(i)$	14	11	12	10	8	9	13	15	3	1	4	5	6	0	2	7

The S-box S used in `NonLinearLayer` The S-box S used in `NonLinearLayer` is the 8-bit S-box given in Table 4.4. The properties of this S-box will be described in Section 4.3.2.

	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F
00	20	00	B2	85	3B	35	A6	A4	30	E4	6A	2C	FF	59	E2	0E
10	F8	1E	7A	80	15	BD	3E	B1	E8	F3	A2	C2	DA	51	2A	10
20	21	01	23	78	5C	24	27	B5	37	C7	2B	1F	AE	0A	77	5F
30	6F	09	9D	81	04	5A	29	DC	39	9C	05	57	97	74	79	17
40	44	C6	E6	E9	DD	41	F2	8A	54	CA	6E	4A	E1	AD	B6	88
50	1C	98	7E	CE	63	49	3A	5D	0C	EF	F6	34	56	25	2E	D6
60	67	75	55	76	B8	D2	61	D9	71	8B	CD	0B	72	6C	31	4B
70	69	FD	7B	6D	60	3C	2F	62	3F	22	73	13	C9	82	7F	53
80	32	12	A0	7C	02	87	84	86	93	4E	68	46	8D	C3	DB	EC
90	9B	B7	89	92	A7	BE	3D	D8	EA	50	91	F1	33	38	E0	A9
A0	A3	83	A1	1B	CF	06	95	07	9E	ED	B9	F5	4C	C0	F4	2D
B0	16	FA	B4	03	26	B3	90	4F	AB	65	FC	FE	14	F7	E3	94
C0	EE	AC	8C	1A	DE	CB	28	40	7D	C8	C4	48	6B	DF	A5	52
D0	E5	FB	D7	64	F9	F0	D3	5E	66	96	8F	1D	45	36	CC	C5
E0	4D	9F	BF	0F	D1	08	EB	43	42	19	E7	99	A8	8E	58	C1
F0	9A	D4	18	47	AA	AF	BC	5B	D5	11	D0	B0	70	BB	0D	BA

Table 4.4: The S-box in hexadecimal notation. The column indicates the least significant nibble and the row indicates the most significant nibble of the S-box input.

Overall encryption process Figure 4.9 gives an overview of the complete encryption process of Lilliput-TBC for all its variants.

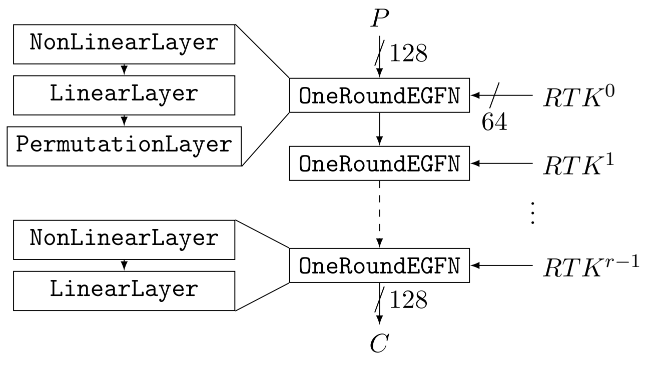


Figure 4.9: Lilliput-TBC Encryption process.

Decryption Process

As Lilliput-TBC is a Feistel network, decryption is quite analogous to encryption but uses the inverse block permutation π^{-1} given in Table 4.3 and the subkeys in the reverse order. Note that the tweakey process could be inverted at low cost.

Tweakey Schedule

An adapted version of the TWEAKEY framework [JNP14] was used as a building block for the scheduling of the key and the tweak. More specifically, we used a variant of the STK construction, where the key and the tweak inputs are handled almost the same way. The proposed version is depicted in Figure 4.14.

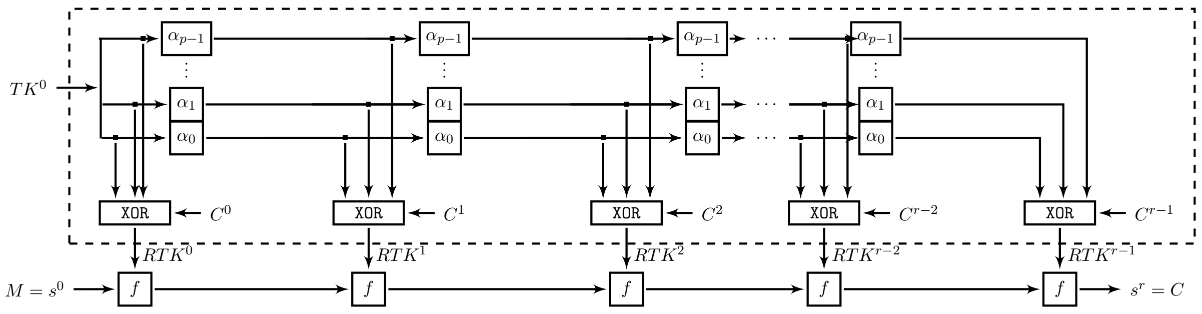


Figure 4.10: The tweakey schedule. f represents the round function OneRoundEGFN.

The tweakey schedule produces the $r = 32$ (36 or 42 respectively) 64-bit subtweakeys RTK^0 to RTK^{r-1} from the 128-bit (192 or 256 respectively) master key K and the tweak T that is 128 bits long when Lilliput-TBC-II is used and 192 bits long tweak when Lilliput-TBC-I is used.

As done in the STK construction, at each round $i \in \{0, \cdots, r-1\}$, the inner state TK^i is divided into $p = (t + k)/64$ lanes that we denote TK^i_j, $j \in \{0, \cdots, p-1\}$, where k is the key length and t is the tweak length. The values of p are shown in Table 4.5, depending on which version of Lilliput-TBC is used.

Name	k	t	p	r
Lilliput-TBC-I-128	128	192	5	32
Lilliput-TBC-I-192	192	192	6	36
Lilliput-TBC-I-256	256	192	7	42
Lilliput-TBC-II-128	128	128	4	32
Lilliput-TBC-II-192	192	128	5	36
Lilliput-TBC-II-256	256	128	6	42

Table 4.5: Recommended parameter sets for Lilliput-TBC and associated number of tweakey lanes.

TK^0 is initialized with the concatenation of the tweak T and the master key K. The first 2 (or 3) lanes are thus dedicated to the 128-bit (or 192-bit) tweak. The key is then stored in the following 2, 3 or 4 lanes, depending on its size.

For each round i, the 8-byte subtweakey word that is produced is denoted RTK^i:

$$\forall i \in \{0, \cdots, r-1\}, \quad RTK^i = RTK^i_7 || RTK^i_6 || RTK^i_5 || RTK^i_4 || RTK^i_3 || RTK^i_2 || RTK^i_1 || RTK^i_0,$$

where RTK^i_j is the byte that is xored to block X_j then used as an input of the nonlinear function F_j in the Lilliput-TBC round function of the encryption process.

The subtweakey word is obtained by XORing all p TK_j^i lanes and a round-dependent constant denoted C^i together. In our proposal, the round constant C^i is simply the round number i:

$$\forall i \in \{0, \cdots, r-1\}, \quad RTK^i = \bigoplus_{j=0}^{p-1} TK_j^i \oplus i.$$

To update the tweakey, at each round $i \in \{1, \cdots, r-1\}$, each 64-bit lane TK_j^i is multiplied by a nonzero coefficient denoted α_j, with $j \in \{0, \cdots, p-1\}$. These coefficients were carefully chosen such that in r consecutive rounds, at most $(p-1)$ cancellations occur as will be shown in Section 4.3.2. Next, we describe how to generate the sequences induced by coefficients α_j $(j = 0, \cdots, 6)$.

Sequences The α-multiplications are computed using word-ring-LFSRs [BMP09]. The sequences constructed as α-multiplications for the tweakey on $GF(2^{64})$ using word-ring-LFSRs are the following ones: consider first a 64-bit lane in byte notation as $x = (x_7, \cdots, x_0)$ where x_7 is the most significant byte and x_0 is the least significant one. In binary notations, we obtain the following vector of 64 bits: $x = (x_{63}^b, \cdots, x_0^b)$. Thus, we have $\alpha_0 = M$, $\alpha_1 = M^2$, $\alpha_2 = M^3$, $\alpha_3 = M^4$, $\alpha_4 = M_R$, $\alpha_5 = M_R^2$ and $\alpha_6 = M_R^3$.

Then the sequence generated by α_0 is produced using the ring-LFSR represented at byte level word by the following matrix:

$$M = \begin{pmatrix} 0 & Id & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & Id & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & S^{\ll 3} & Id & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & S^{\gg 3} & Id & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & Id & 0 & 0 \\ 0 & S^{\ll 2} & 0 & 0 & 0 & 0 & Id & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & Id \\ Id & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

where Id is the 8×8 identity matrix. The primitive polynomial associated with this matrix is computed as $Det(I - M \cdot X)$ at binary level, which gives: $x^{64} + x^{58} + x^{42} + x^{40} + x^{35} + x^{34} + x^{29} + x^{26} + x^{24} + x^{23} + x^{19} + x^{10} + 1$. The multiplication by α_0 is then generated as $(y_7, \cdots, y_0)^t = M \cdot (x_7, \cdots, x_0)^t$. Thus, we have: $(y_7, \cdots, y_0)^t = (x_6, x_5, x_4 \oplus x_5 \ll 3, x_3 \oplus x_4 \gg 3, x_2, x_1 \oplus x_6 \ll 2, x_0, x_7)^t$.

$$M^2 = \begin{pmatrix} 0 & 0 & Id & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & S^{\ll 3} & Id & 0 & 0 & 0 & 0 \\ 0 & 0 & S^{\ll 6} & M_1 & Id & 0 & 0 & 0 \\ 0 & 0 & 0 & S^{\gg 6} & S^{\gg 3} & Id & 0 & 0 \\ 0 & S^{\ll 2} & 0 & 0 & 0 & 0 & Id & 0 \\ 0 & 0 & S^{\ll 2} & 0 & 0 & 0 & 0 & Id \\ Id & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & Id & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

with M_1 equal to the binary 8×8 following matrix:

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

then

$$M^3 = \begin{pmatrix} 0&0&S^{\ll 3}&Id&0&0&0&0\\ 0&0&S^{\ll 6}&M_1&Id&0&0&0\\ 0&0&0&M_2&M_1&Id&0&0\\ 0&S^{\ll 2}&0&0&S^{\gg 6}&S^{\gg 3}&Id&0\\ 0&0&S^{\ll 2}&0&0&0&0&Id\\ Id&0&S^{\ll 5}&S^{\ll 2}&0&0&0&0\\ 0&Id&0&0&0&0&0&0\\ 0&0&Id&0&0&0&0&0 \end{pmatrix}$$

with M_2 a binary matrix of size 8×8 equal to

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

and finally

$$M^4 = \begin{pmatrix} 0&0&S^{\ll 6}&M_1&Id&0&0&0\\ 0&0&0&M_2&M_1&Id&0&0\\ 0&S^{\ll 2}&0&M_3&M_2&M_1&Id&0\\ 0&M_4&S^{\ll 2}&0&0&S^{\gg 6}&S^{\gg 3}&Id\\ Id&0&S^{\ll 5}&S^{\ll 2}&0&0&0&0\\ 0&Id&0&M_5&S^{\ll 2}&0&0&0\\ 0&0&Id&0&0&0&0&0\\ 0&0&S^{\ll 3}&Id&0&0&0&0 \end{pmatrix}$$

with M_3 a binary matrix of size 8×8 equal to

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

M_4 a binary matrix of size 8×8 equal to

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

and M_5 a binary matrix of size 8×8 equal to

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

To generate the three following sequences, we use the following matrices using the reciprocal primitive polynomial of $x^{64}+x^{58}+x^{42}+x^{40}+x^{35}+x^{34}+x^{29}+x^{26}+x^{24}+x^{23}+x^{19}+x^{10}+1$ equal to $x^{64}+x^{54}+x^{45}+x^{41}+x^{40}+x^{38}+x^{35}+x^{30}+x^{29}+x^{24}+x^{22}+x^{6}+1$.

The outputs are then computed in the reverse order using the relation $(y_0, \cdots, y_7)^t = M_R \cdot (x_0, \cdots, x_7)^t$. Note that the associated binary words are also written in the opposite way compared with the computations performed for M, M^2, M^3 and M^4. It means that, in this case, at binary level, we have $x = (x^b_0, \cdots, x^b_{63})$ and $x_i = (x^b_{8\cdot i+0}, \cdots, x^b_{8\cdot i+7})$.

Thus, we have: $(y_0, \cdots, y_7)^t = (x_1, x_2, x_3 \oplus x_4 \ll 3, x_4, x_5 \oplus x_6 \gg 3, x_6 \oplus x_3 \gg 2, x_7, x_0)^t$.

$$M_R = \begin{pmatrix} 0&Id&0&0&0&0&0&0\\ 0&0&Id&0&0&0&0&0\\ 0&0&0&Id&S^{\ll 3}&0&0&0\\ 0&0&0&0&Id&0&0&0\\ 0&0&0&0&0&Id&S^{\gg 3}&0\\ 0&0&0&S^{\gg 2}&0&0&Id&0\\ 0&0&0&0&0&0&0&Id\\ Id&0&0&0&0&0&0&0 \end{pmatrix}$$

$$M_R^2 = \begin{pmatrix} 0&0&Id&0&0&0&0&0\\ 0&0&0&Id&S^{\ll 3}&0&0&0\\ 0&0&0&0&Id&S^{\ll 3}&M_6&0\\ 0&0&0&0&0&Id&S^{\gg 3}&0\\ 0&0&0&S^{\gg 2}&0&0&Id&S^{\gg 3}\\ 0&0&0&0&S^{\gg 2}&0&0&Id\\ Id&0&0&0&0&0&0&0\\ 0&Id&0&0&0&0&0&0 \end{pmatrix}$$

with M_6 a binary matrix of size 8×8 equal to

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

and

$$M_R^3 = \begin{pmatrix} 0&0&0&Id&S^{\ll 3}&0&0&0\\ 0&0&0&0&Id&S^{\ll 3}&M_6&0\\ 0&0&0&M_7&0&Id&M_1&M_6\\ 0&0&0&S^{\gg 2}&0&0&Id&S^{\gg 3}\\ S^{\gg 3}&0&0&0&S^{\gg 2}&0&0&Id\\ Id&0&0&0&0&S^{\gg 2}&S^{\gg 5}&0\\ 0&Id&0&0&0&0&0&0\\ 0&0&Id&0&0&0&0&0 \end{pmatrix}$$

with M_7 is a binary matrix of size 8×8 equal to

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

The periods of each sequence given by the word-ring-LFSRs produced by the previous matrices are respectively: $2^{64} - 1$ for the sequences produced using M, M^2,M^4, M_R, M_R^2 and $\frac{2^{64}-1}{3}$ for the sequences produced using M^3 and M_R^3.

4.3 Design Rationale and Security Analysis

In this section, we will detail the design choices we made for Lilliput-TBC and we provide a complete security analysis regarding a wide variety of attacks.

4.3.1 Design Rationale of the Modes of Operation

ΘCB3

The OCB mode (Offset Codebook Mode) was designed by Phillip Rogaway, who took inspiration from Charanjit Jutla's IAPM (Integrity Aware Parallelizable Mode) [Jut01]. The original authenticated-encryption scheme has then been refined several times, leading to three named versions: OCB1 [RBBK01], OCB2 [Rog04] and OCB3 [KR11b]. The main change introduced with OCB2 is the possibility to handle Associated Data (AD), while the modifications made in OCB3 are rather minor (mostly a change in the way offsets are incremented). The OCB mode has many advantages, starting with the fact that it is parallelizable and only requires one block cipher invocation per message block, in contrary to schemes like GCM. In [KR11b], Krovetz and Rogaway also introduced a tweakable block cipher generalization of OCB3 denoted ΘCB3, which is at the source of the mode used for our candidate LILLIPUT-I.

OCB is covered by the United States Patent No. 7,949,129, United States Patent No. 8,321,675, United States Patent No. 7,046,802 and United States Patent No. 7,200,22. Still, it is unclear if ΘCB3 is also covered by patents. This lack of clarity is part of the reason why we selected LILLIPUT-II to be our primary member.

Under the assumption that the underlying (tweakable) block cipher is secure as a strong-PRP (PseudoRandom Permutation), OCB is provably secure and achieves confidentiality and authenticity. Confidentiality means that an adversary cannot make the distinction between OCB outputs and random bits, while authenticity (of ciphertexts) means that she cannot produce a valid nonce-ciphertext pair (different from the ones she previously obtained). Note that the various variants of OCB are not designed to resist to nonce reuse nor to enjoy beyond-birthday-bound security.

SCT-2

The Synthetic Counter in Tweak mode (SCT) was first devised at Crypto 2016 by Thomas Peyrin and Yannick Seurin [PS16]. Few months later, the mode was slightly modified by the same authors associated with Jérémy Jean and Ivica Nikolic to be used as a mode for one of the member of the family of authenticated ciphers DEOXYS v1. 43 [JNPS16], their submission to CAESAR (*Competition for Authenticated Encryption: Security, Applicability, and Robustness*). The rearranged mode was named SCT-2, and the corresponding authenticated cipher was coined DEOXYS-II.

The difference between SCT and SCT-2 only lies in the way the tag is produced (the encryption part is similar), a change that was done *"in order to provide graceful degradation of security for authentication with the maximal number of repetitions of nonce"* [JNPS16].

4.3.2 Design Rationale of LILLIPUT-TBC

When designing LILLIPUT-TBC from the block cipher LILLIPUT, our overall goal was to maximize diffusion between nibbles or bytes while keeping reasonable implementation performances. This diffusion could be measured using the notion of *full diffusion delay* of [BMT14]. It will be denoted by d and corresponds to the minimum number of rounds needed for all output bytes or nibbles to depend on all input bytes or nibbles. It is closely related to some structural attacks such as impossible differentials or integral attacks, as shown in [SM10, BMT14].

We decided to use the EGFN inferred in LILLIPUT [BFMT16] to reach this purpose because the full diffusion delay of the EGFN of LILLIPUT is equal to $d = 4$ which is the best diffusion delay obtained for a Feistel-like scheme.

We chose a 128-bit state as it is consistent with the NIST requirements. We split that state into 16 bytes so that the block size matches the S-box size, *i.e.* the non-linear layer is made only of 8 parallel calls to an 8-bit S-box and 8 subkey additions. As said before, the π permutation has been chosen to maximize the number of active S-boxes on 18, 19 and 20 rounds (see Section 4.3.4 and Table 4.7 for more details).

From those results and the security analysis performed in Section 4.3.4 and summed up in Table 4.10, we also deduced the number of rounds of each instance of LILLIPUT-I and of

Lilliput-II equal to 32, 36 or 42 rounds.

The EGFN Structure

As done in [SM10, BMT14], we analyze here the security of our underlying `EGFN` scheme regarding the pseudorandomness of the scheme. Note that the pseudorandomness bounds obtained are generic and essentially depend on the d value. We thus introduce the pseudo-random-permutation advantage (`prp`-advantage) and the strong-pseudo-random-permutation advantage (`sprp`-advantage) of an adversary. For this purpose, we introduce the two advantage notations as:

$$\mathrm{Adv}_{\mathsf{C}}^{\mathsf{prp}}(q) = \max_{\mathsf{A}:q\text{-}\mathsf{CPA}} \left|\Pr[\mathsf{A}^{\mathsf{C}} = 1] - \Pr[\mathsf{A}^{\mathsf{P}_n} = 1]\right| \tag{4.1}$$

$$\mathrm{Adv}_{\mathsf{C}}^{\mathsf{sprp}}(q) = \max_{\mathsf{A}:q\text{-}\mathsf{CCA}} |\Pr[\mathsf{A}^{\mathsf{C},\mathsf{C}^{-1}} = 1] - \Pr[\mathsf{A}^{\mathsf{P}_n,\mathsf{P}_n^{-1}} = 1]| \tag{4.2}$$

where C is the encryption function of an n-bit block cipher composed of Uniform Random Functions (URFs) as internal modules whereas C^{-1} is its inverse; P_n is an n-bit Uniform Random Permutation (URP) uniformly distributed among all the n-bit permutations; P_n^{-1} is its inverse. The adversary, A, tries to distinguish C from P_n using q queries in a CPA (Chosen Plaintext Attack) and tries to distinguish, always using q queries, $(\mathsf{C}, \mathsf{C}^{-1})$ from $(\mathsf{P}_n, \mathsf{P}_n^{-1})$ in a CCA (Chosen Ciphertext Attack). The notation means that the final guess of the adversary A is either 0 if A thinks that the computations are done using P_n, or 1 if A thinks that the computations are done using C. The maximums of Equations (4.1, 4.2) are taken over all possible adversaries A with q queries and an unbounded computational power.

To prove the bounds of our scheme in those models, we recall the result proved in [BFMT16]: let $\Phi_{kc,r}$ denote our k-block scheme acting on c-bit blocks with $n = kc$, using r rounds and with diffusion delay d. We then have (the proof can be found in [BFMT16]):

Theorem 4.1. *Given the r-round* EGFN $\Phi_{kc,r}$ *with k branches acting on c-bit blocks with a diffusion delay d where all c-bit round functions are independent URFs. Then we have:*

$$\mathrm{Adv}_{\Phi_{kc,d+2}}^{\mathsf{prp}}(q) \leq \frac{kd}{2^c} q^2 \tag{4.3}$$

$$\mathrm{Adv}_{\Phi_{kc,2d+2}}^{\mathsf{sprp}}(q) \leq \frac{kd}{2^{c-1}} q^2 \tag{4.4}$$

Thus, we have a classical security proof for the choice of the underlying `EGFN` used in Lilliput-TBC. Note that, in our case, c is equal to 8.

The π Permutation

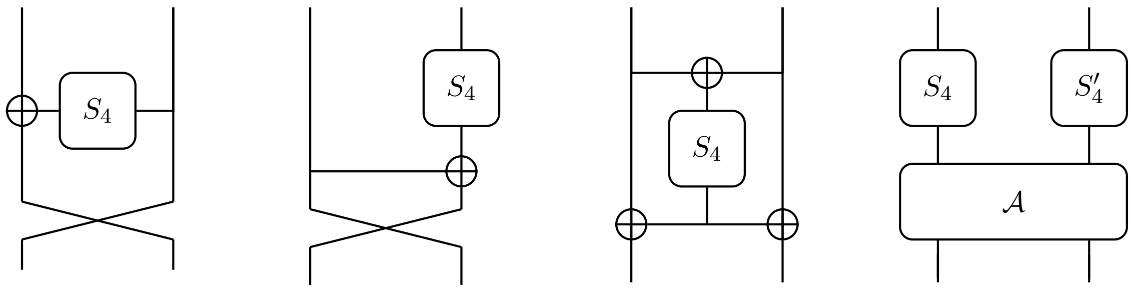
Full diffusion delay is closely related to some structural attacks such as impossible differentials or integral attacks, as shown in [SM10, BMT14]. As there are many EGFNs that achieve $d = 4$, we chose one by taking other considerations into account. Specifically, we chose the block-wise permutation to maximize resistance against differential and linear cryptanalysis.

We identified the criterion for a permutation π to achieve $d = 4$ to be as follows: first, π must swap the 8 right-most blocks with the 8 left-most, and second, π must specifically swap blocks Y_7 and Y_{15} (a complete proof could be found in [BMT14, BFMT16]).

Up to block reindexing equivalence, there are exactly 37108 such permutations. For each of them, we computed the minimal number of differentially and linearly active S-boxes up to 20 rounds (see Section 4.3.4 and Table 4.7 for more details) and picked the one that maximizes the number of active S-boxes on 18, 19 and 20 rounds.

The S-box

Overall structure We chose to build the 8-bit S-box from smaller ones so that it could be implemented with a fewer number of gates, which is a valuable property for hardware and bit-sliced software implementations. S-boxes built in this fashion usually rely on one of the four constructions depicted in Figure 4.11. We defined the following selection criteria for the candidate



(a) Feistel (b) MISTY (c) Lai-Massey (d) SPN

Figure 4.11: Some structures to build $2n$-bit S-boxes from n-bit ones.

S-box:

- Differential uniformity $\delta \leq 10$
- Linearity $\mathcal{L} \leq 64$
- Algebraic degree $\mathsf{deg} \geq 6$.

All constructions discussed above need to be iterated several times in order to achieve the desired cryptographic properties. Regarding Feistel and MISTY networks, [CDL16] gives lower bounds on the differential uniformity and linearity for 3-round balanced constructions. In this case, Feistel networks provide better cryptographic properties as it is possible to reach $\delta = 8$ and $\mathcal{L} = 64$ versus $\delta = 16$ and $\mathcal{L} = 64$ for MISTY networks. It comes from the fact that Feistel networks do not require inner 4-bit S-boxes to be permutations, allowing the use of Almost Perfect Nonlinear (APN) functions as inner components. Still, it has been shown that 3-round unbalanced MISTY networks (*e.g.*, dividing the 8-bit input into two inequal parts of 3 and 5 bits) can be used to build 8-bit S-boxes with $\delta = 8$ [CDL16]. However, because the unbalanced words induce components with strong unbalanced degrees for the ANF, we decided to rule out this option.

Regarding the Lai-Massey scheme, the family of block ciphers FOX [JV04] uses a 3-round iterated structure in order to build an 8-bit S-box with $\delta = 16$ and $\mathcal{L} = 64$. On the other side, some cryptographic primitives simply add a nonlinear layer (*i.e.* two 4-bit S-boxes in parallel) at the beginning and/or at the end of the original scheme depicted in Figure 4.11c instead of using an iterated structure and therefore save some `XOR` gates. For instance, the block cipher FLY [KG16] adds a nonlinear layer at the end of the scheme while the hash function Whirlpool [BRAN00] also adds one at the beginning resulting in a total of three and five inner 4-bit S-boxes, respectively. As for the MISTY ladder, the Lai-Massey structure requires inner 4-bit S-boxes to be permutations and therefore constructions that use three 4-bit S-boxes cannot reach a differential uniformity as good as Feistel networks. Although the cryptographic properties achieved by the variant using five 4-bit S-boxes are compliant with our selection criteria (*i.e.* $\delta = 8$, $\mathcal{L} = 56$ and $\mathsf{deg} = 7$ for the Whirlpool S-box), it is not worth the implementation cost.

The same reasoning can be applied to SPNs. For instance, the block cipher CLEFIA [SSA+07] uses two different 8-bit S-boxes and one of them relies on an SPN structure as defined in Figure 4.11d with an additional nonlinear layer after $\mathcal{A}$ which refers to a matrix multiplication over $\mathbb{F}_{16}$. It results in an S-box with $\delta = 10$, $\mathcal{L} = 56$ and $\mathsf{deg} = 6$ which is compliant with our selection criteria. However, because it uses four 4-bit S-boxes, it is more heavy than a 3-round Feistel network to implement.

For all these reasons, we opted for an 8-bit S-box based on a 3-round balanced Feistel network. In the rest of this section, S_4^i refers to the inner 4-bit S-box at the i-th round.

Inner 4-bit S-boxes According to [CDL16], in order to reach $\delta = 8$, S_4^1 and S_4^3 have to be APN functions while S_4^2 has to be a permutation with differential uniformity 4. The authenticated block cipher SCREAM [GLS+15] uses an 8-bit S-box built in this manner where the underlying APN functions are $S_4^1 =$ `020b300a1e06a452` and $S_4^3 =$ `20b003a0e1604a25` and the permutation is $S_4^2 =$ `02c75fd64e8931ba`.

S_4^1 can be implemented using 11 instructions in total (either `AND` or `OR` or `XOR` or `NOT` or `MOV`), including 4 non-linear ones, while S_4^3 is directly derived from it by adding a `NOT` instruction in order to avoid fixed points. Although it is possible to find APN functions over $\mathbb{F}_{16}$ that are built using 10 instructions from the same instruction set, they all require at least 6 non-linear ones [CDL16] which is not optimal regarding masked implementations. S_4^2 is built using 9 instructions from the same instruction set, including 4 non-linear ones, which is the smallest implementation cost for a 4-bit S-box with differential uniformity 4 [UDI+11]. Therefore, the SCREAM S-box allows very efficient implementations with and without masking as it only requires 44 instructions in total including 12 non-linear ones.

However, the number of non-linear operations is not the only criteria regarding Threshold Implementations (TI) where an S-box with algebraic degree d requires at least $n = d + 1$ shares. In order to limit the number of shares for a 4-bit S-box with $d > 2$, it has been proposed to use its decomposition into quadratic bijections [NRS11] (*i.e.* $S_4^i = F \circ G$) so that it is possible to achieve a TI with $n = 3$. In order to fulfill the uniformity criteria, it has been proposed to find affine functions A_1 and A_2 such that $F = A_1 \circ \mathcal{Q} \circ A_2$, so that when it is possible to achieve a uniform sharing of the quadratic function Q, applying A_1 and A_2 on all input and output shares respectively gives a uniform sharing of F [BNN+12].

In [BGG+16] the authors study first-order TIs for several 8-bit S-boxes, including the one used in SCREAM. It results that the two APN functions S_4^1 and S_4^3 can be directly decomposed into two quadratic functions while the permutation S_4^2 requires affine functions as described above. In order to achieve more efficient TIs by saving the implemention cost of the affine functions, we looked for (and found) alternatives to S_4^2 that could be directly decomposed into two quadratic functions.

We chose to investigate all possible circuits with a Breadth-First Search (BFS) approach, including only `AND`, `XOR` and `NOT` gates as they can be straightforwardly thresholded. This approach is very similar to [UDI+11]. We optimized the number of gates used without considering `MOV` instructions as we consider that wiring is free compared to the cost of the gates. We allowed 5 registers during the exploration. Keeping the affine equivalence notion of the previous paper, stopping the exploration to 8 gates allowed us to reach 62 affine equivalence classes, including 3 optimal classes according to [Saa11]. Following the same notation as [BNN+12] to refer to the equivalence classes, the three optimal classes we reached are $\mathcal{C}_{223}, \mathcal{C}_{296}$ and $\mathcal{C}_{297}$.

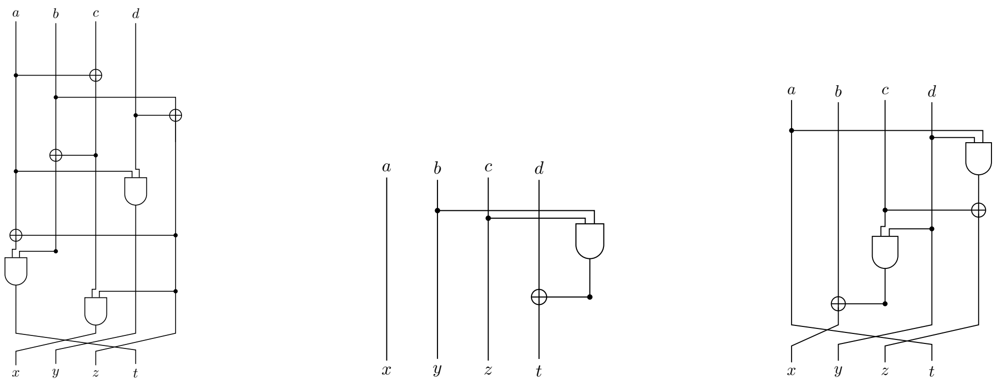
We focused on permutations of the optimal classes as they are the only ones with differential uniformity equal to 4. First, we eliminated candidates that did not allow to reach the selection criteria for the 8-bit S-box when used as S_4^2 in a 3-round Feistel network. After this step, there were still candidates in each optimal class. In order to go further into the optimization of TIs, we investigated the decomposition of the remaining candidates. Following [BNN+12], we decomposed those cubic permutations into two quadratic functions. There are six quadratic classes denoted by $\mathcal{Q}_4$, $\mathcal{Q}_{12}$, $\mathcal{Q}_{293}$, $\mathcal{Q}_{294}$, $\mathcal{Q}_{299}$ and $\mathcal{Q}_{300}$. It results from our BFS exploration that these classes can be implemented with a minimum of 2, 4, 6, 4, 6 and 6 gates, respectively. Among these classes, only $\mathcal{Q}_4$, $\mathcal{Q}_{294}$ and $\mathcal{Q}_{299}$ contain permutations that are uniform using direct sharing. However, neither $\mathcal{C}_{223}$ nor $\mathcal{C}_{296}$ nor $\mathcal{C}_{297}$ can be decomposed using $\mathcal{Q}_4$. On the other hand, because only $\mathcal{C}_{223}$ can be decomposed into two quadratic functions of the class $\mathcal{Q}_{294}$ that are uniform using direct sharing, this makes $\mathcal{C}_{223}$ the most interesting optimal class we reached regarding TIs.

As stated before, contrary to S_4^2, our aim was to avoid linear permutations between the quadratic functions. As we consider that wire permutations ω_i are free, for all the remaining candidates C in $\mathcal{C}_{223}$, we looked for 4-gate circuits Q_i and Q_j of $\mathcal{Q}_{294}$ that are uniform using direct sharing, such that $C = \omega_1 \circ Q_i \circ \omega_2 \circ Q_j \circ \omega_3$. As a final step to determine which composition to use, we calculated the cost (considered in Gate Equivalents −GEs) of a 3-share TI of all of them using this formula:

$$\text{GE} = 3\text{GE}_X \cdot X + (3\text{GE}_X + 3\text{GE}_A) \cdot A + \text{GE}_N \cdot N \tag{4.5}$$

with GE_X the area and X the number of `XOR` gates, GE_A the area and A the number of `AND` gates and GE_N the area and N the number of `NOT` gates. It comes from the fact that, when considering 3-share TIs, thresholding an `XOR` gate requires 3 `XOR` gates while thresholding an `AND` gate requires 6 `XOR` and 9 `AND` gates. Taking $\text{GE}_X = \frac{8}{3}$, $\text{GE}_A = \frac{4}{3}$ and $\text{GE}_N = \frac{2}{3}$, we found the minimum at $72 \cdot 2 = 143$GEs. Note that it does not include the cost of registers between the two permutations that are needed to ensure security against glitches.

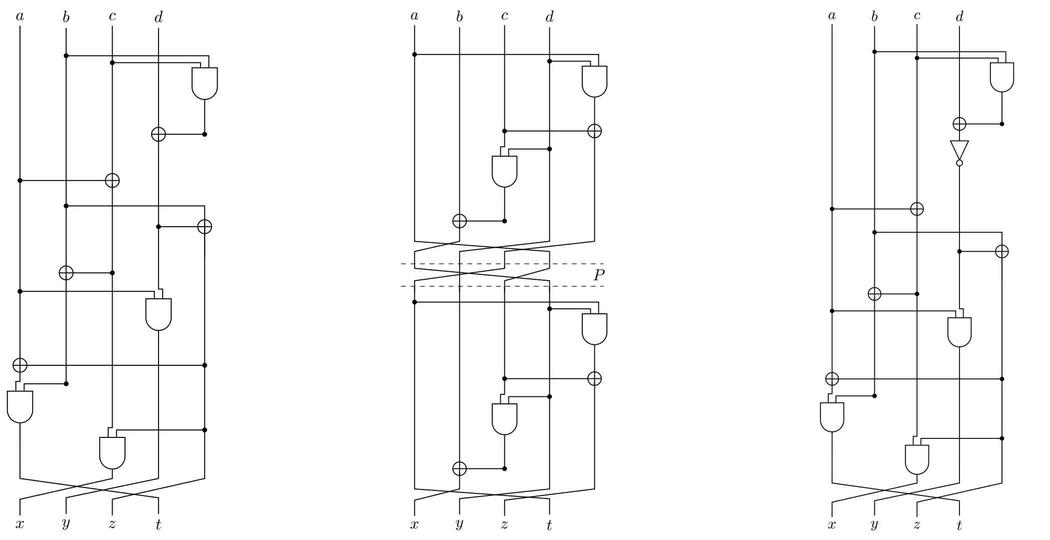
Among the several compositions that can be implemented using 143GEs, we chose the permutation illustrated in Figure 4.13b as it constitutes the only candidate that results from the composition of the same quadratic permutation $Q =$ `042e8ca6173d9fb5`, allowing to optimize the area cost in particular cases.



(a) $F =$ `020b30a01e06a425` (b) $G =$ `0123457689abcdfe` (c) $Q =$ `042e8ca6173d9fb5`

Figure 4.12: Quadratic functions used to build the cubic 4-bit S-boxes.

To put it in a nutshell, our 8-bit S-box is obtained by combining the two APN functions from the SCREAM S-box with the 4-bit permutation $\bar{S}_4^2 =$ `081f4c792b36e5da` in a 3-round Feistel network and achieves $\delta = 8$, $\mathcal{L} = 64$ and $\mathtt{deg} = 6$ without fixed points. Thanks to the BFS exploration, we ensure that our S-box requires a small number of gates and that its TI is efficient as it uses the smallest circuits of its possible decompositions and futhermore, it does not require the use of affine permutations when decomposed into two quadratic functions. The 4-bit S-boxes are depicted in Figure 4.13 while the underlying quadratic functions are depicted in Figure 4.12, where a and d refer to the most and the least significant bits, respectively.



(a) $S_4^1 = F \circ G$ `020b300a1e06a452` (b) $\bar{S}_4^2 = Q \circ P \circ Q$ `081f4c792b36e5da` (c) $S_4^3 = F \circ (\oplus 1) \circ G$ `20b003a0e1604a25`

Figure 4.13: The three inner 4-bit S-boxes.

The Tweakey Schedule

As done for some other Tweakable Block Ciphers, we first looked at the TWEAKEY construction of [JNP14] that fills p lanes of n bits divided into m words of c bits with the concatenation of the tweak T and of the key K. Then, to produce the subtweakey of each round, the TWEAKEY framework applies, on each lane, a permutation h acting on the m words and then multiply each of the m elements of c bits by a primitive root $\alpha_i, \forall i \in \{0, \cdots, p-1\}$ over $GF(2^c)$ different for each lane. Then, the subtweakey is the XOR of the p lanes and of a constant. From this construction that could be seen as the tensorial product of m Vandermonde matrices, the authors could deduce that the number of cancellations on $r+1$ subtweakeys is at most equal to $(p-1)$. Indeed, the updating function (excluding the h permutation) for the c bits words could be written as the following Vandermonde matrix

$$V = \begin{pmatrix} \alpha_0^0 & \alpha_0^1 & \alpha_0^2 & \cdots & \alpha_0^r \\ \alpha_1^0 & \alpha_1^1 & \alpha_1^2 & \cdots & \alpha_1^r \\ \alpha_2^0 & \alpha_2^1 & \alpha_2^2 & \cdots & \alpha_2^r \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \alpha_{p-1}^0 & \alpha_{p-1}^1 & \alpha_{p-1}^2 & \cdots & \alpha_{p-1}^r \end{pmatrix}$$

when all α_i with $0 \leq i \leq r$ are distinct considering that $r < ord(\alpha)$. In this case, the code defined by V is a Reed-Solomon code of length $r+1$ and dimension p over $GF(2^c)$ and it is known to be MDS (Maximum Distance Separable). It means that its minimum distance is equal to $r+1-(p+1)$.

For designing our own tweakey schedule, we adopted the same idea to keep the Vandermonde strategy in order to guarantee the maximal possible number of cancellations. However, as we wanted to reduce the latency of the tweakey schedule and thus the number of computations, we adopted the following strategy instead of considering a lane as a vector of m elements of $GF(2^c)$: we directly consider the field $GF(2^{cm})$ that will be equal in our case to $GF(2^{64})$. Indeed, in our case, the size of each lane is equal to $\frac{n}{2} = 64$ bits due to the use of a Feistel-like scheme requiring only $\frac{n}{2} = 64$ bits of tweakey injected in the round function at each iteration.

Thus, we consider the p 64-bit long lanes as p elements of $GF(2^{64})$ and we multiply each lane by p different α_i given in Section 4.2.3 in a byte oriented matrix representation. Thus, with our construction, we obtain the following Vandermonde matrix constructed on $GF(2^{64})$:

$$V' = \begin{pmatrix} \alpha_0^0 & \alpha_0^1 & \alpha_0^2 & \cdots & \alpha_0^{r-1} \\ \alpha_1^0 & \alpha_1^1 & \alpha_1^2 & \cdots & \alpha_1^{r-1} \\ \alpha_2^0 & \alpha_2^1 & \alpha_2^2 & \cdots & \alpha_2^{r-1} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \alpha_{p-1}^0 & \alpha_{p-1}^1 & \alpha_{p-1}^2 & \cdots & \alpha_{p-1}^{r-1} \end{pmatrix}$$

Thus, as $\forall 0 \leq i < r$, we have chosen the α_i such that $r < ord(\alpha_i)$, we preserve the MDS property induced by the underlying Reed-Solomon code and guarantee that the minimum distance is equal to $r-(p+1)$ leading to at most $(p-1)$ cancellations on r subtweakeys, seen always, as the XOR of the p lanes. This last choice is a logical one as done in many lightweight block ciphers, such as PRESENT [BKL+07], TWINE [SMMK13], LBLOCK [WZ11a] or SIMON [BSS+13] where the tweakey/key material is loaded in an initial register that is sequentially updated and where the subtweakeys/subkeys are extracted from that register.

We also chose to split the tweakey into p lanes of 64-bit instead of having a big state of $p \times 64$ bits and to update in parallel those p registers because small updating functions mix their content faster and increase performance. The downside is that each updating function could be attacked independently if their contents were not combined back during the subtweakey extraction which is not the case here.

Let us now explain how we have chosen the different α_i and the word-ring-LFSRs matrix multiplications at binary level that perform those operations.

We used LFSRs inspired by the results of [BMP09] and [ABMP11] on LFSRs. LFSRs are typically used either in Fibonacci or Galois mode. In the first case, many feedbacks are used to influence a single cell while in the second case a single feedback influences many cells. In [BMP09], the authors generalize LFSR beyond Fibonacci/Galois representation by allowing any cell to be used as feedback in any other cell. They call these new LFSRs "ring-LFSRs" because of the rotation occuring at each update. As the LFSRs in [ABMP11], the LFSRs chosen here have also a word-oriented structure: instead of performing bit-wise shift at each iteration and having binary feedbacks, they are shifted by one word at each update. As for the feedbacks, they are also word-oriented: one whole word is xored to another after possibly being transformed by a software-friendly operation such as shift or rotation. Those LFSRs are called word-LFSRs by their authors [ABMP11]. When a LFSR is both a word and ring LFSR, we call it a word-ring-LFSR. At the same time, they act at word level and they have feedbacks going from some word to some other. Word-ring-LFSRs have thus a smaller diffusion delay than classical Fibonacci or Galois LFSRs.

We have chosen our word-ring-LFSR defined by the matrix M for the α_0-multiplication with the minimal possible number of shift operations (3 at 8 bits words level) to minimize the number ofXOR gates, with a primitive polynomial of degree 64. We chose words of size 8 bits to fit well on software platforms. Then, α_1, α_2 and α_3 multiplications are deduced directly using M^2, M^3 and M^4.

Moreover, to construct the matrix M_R for the α_4 multiplication, we searched for a matrix with 3 shift operations implementing the reciprocal (primitive) polynomial that defines M to ensure that the matrix V' stays a Vandermonde matrix and that the sequences produced when multiplying by α_0 (α_1, α_2 and α_3 respectively) and α_4 (α_5 and α_6 respectively) have only a single common value.

We have also chosen the different α_i with a primitive retroaction polynomial to ensure that the induced periods are maximal: the period for $\alpha_0, \alpha_1, \alpha_3, \alpha_4$ and α_5 is maximal and equal to $2^{64} - 1$ whereas the period for α_2 and α_6 is equal to $\frac{2^{64}-1}{3}$.

Moreover, with this design strategy in mind, we are sure that the entire possible space is reached discarding the risk of an invariant attack as detailed in Section 4.3.4.

4.3.3 Security Analysis of the Modes of Operation

ΘCB3

The past year has seen several breakthroughs in the analysis of OCB, starting in October 2018 with the description by Inoue and Minematsu of a practical existential forgery attack [IM18]. Few weeks after, Poettering [Poe18] extended this result and broke the confidentiality of OCB2, result that was extended further by Iwata [Iwa18] who devised a plaintext recovery attack[3]. These attacks were clearly announced by their authors as not applicable to OCB1 and OCB3, so ΘCB3 is also safe. To the best of our knowledge, no attacks were devised on ΘCB3.

SCT-2

To the best of our knowledge, no flaws were found so far in SCT-2 and the results published on Deoxys [ZDW19, MMS18, CHP+17] only target the underlying cipher (that is, Deoxys-BC). In [CHP+17], the authors briefly discuss if their attacks on Deoxys-BC could apply once the cipher is used in the corresponding mode, and *"argue that [their] attacks are difficult to extend to* Deoxys-*II"*. This seems to indicate that the SCT-2 mode does not induce additional flaws to a cipher but on the contrary results in an extra protection coming from the fact that the attacker cannot access the decryption primitive.

To further support that the mode SCT-2 is trusted by the community, we recall here that Deoxys-II was selected after a 5-year process as the first choice for use case 3 ("Defense in depth") in the final Caesar portfolio [CAE].

Security Claims for the Modes

Our security claims for the different variants of Lilliput-AE are provided in Table 4.6.

The bounds are given in the case of a tag size $\tau \geq n$. Should a smaller tag size be used, the security claims will drop according to τ. We derived the security bounds from the security proofs of ΘCB3 [KR11b] and SCT [PS16] and we refer to them for more details.

[3]These three results have been recently merged together in [IIMP19]

	Security (bits)	
Goal (nonce-respecting case)	LILLIPUT-I	LILLIPUT-II
Key recovery	k	k
Confidentiality for the plaintext	n	$n-1$
Integrity for the plaintext	n	$n-1$
Integrity for the associated data	n	$n-1$

	Security (bits)	
Goal (nonce-misuse case)	LILLIPUT-I	LILLIPUT-II
Key recovery	k	k
Confidentiality for the plaintext	none	$n/2$
Integrity for the plaintext	none	$n/2$
Integrity for the associated data	none	$n/2$

Table 4.6: Security goals of LILLIPUT-AE in the nonce-respecting case and in the nonce-misuse case.

4.3.4 Security Analysis of LILLIPUT-TBC

We analyze the security of LILLIPUT-TBC regarding classical attacks in the unknown key model and also in the related-key model always considering the related tweak model. We will place ourselves for all the attacks in the so-called "paranoid"' case, where the worst case is always envisaged even if it could not be reached.

Thus, we divide this section in the following way: we first consider differential/linear cryptanalysis, thus, extending those first results to the case of related-key boomerang attacks and then, we give overall bounds for the so-called structural attacks that include impossible differential attacks, zero-correlation attacks, integral attacks and meet-in-the-middle attacks. Then, we take a particular attention on the following special attacks: division property, subspace cryptanalysis, algebraic attack.

Thus, we will first introduce the following bounds that will be used in the rest of this section:

- As the full diffusion is reached after $d = 4$ rounds for LILLIPUT-TBC, it means that no structural distinguisher can be constructed for more than $2d + 2$ rounds (see [BMT14] for a detailed analysis and the security proofs).
- We will also always consider that the number of rounds that can be added to the best distinguisher for the key recovery part at the beginning is equal to d and at the end is also equal to d. Indeed, if a property is found on a single byte at the beginning or at the end of the distinguisher then after d rounds, all the input/output bytes will be influenced, so a key recovery could not exceed those bounds.
- As the tweakey schedule is fully linear and based on the XOR of elements of a Vandermonde matrix, it means that by reversing the linear system, one is able to find in the related-tweak/related-key models $(p-1)$ cancellations (when p lanes are considered) placed at the best for the attacker.

First, let us precise that to prevent slide attacks [BW99] and as usually done in other tweakable block cipher proposals, different round constants are added to each subtweakey during the tweakey schedule process. So, we consider LILLIPUT-TBC immune to slide attacks.

Differential / Linear Cryptanalysis

To prove the resistance of LILLIPUT-TBC against differential and linear cryptanalysis, we give in Table 4.7 the lower bounds on the minimal number of active S-boxes in the single tweakey model considering no difference in the tweak. Those bounds partly fit with the ones given in [ST16] for the block cipher LILLIPUT. We have obtained those results using Constraint Programming up to 20 rounds in the single tweakey model. Due to the complexity of the tweakey schedule, we could not derive bounds for the related tweakey models (note that the related tweakey models are not considered for linear cryptanalysis). However, we could place ourselves in the worst case saying that authorizing a particular difference in a single lane i means that the results given in Table 4.7 on r rounds apply for $r+2$ rounds, in two lanes i and j means that the results given in Table 4.7 on r rounds apply for $r+3$ rounds, and so on up to p lanes are activated.

Moreover, we use here the fact that, as mentioned in Section 4.3.2, we have $\delta = 2^{-5}$ and $\mathcal{L} = 64$ for the chosen S-box.

Thus, with this reasoning, we could derive the following bounds for the different key lengths on the best differential/linear attacks:

- LILLIPUT-TBC-I-128 ($t = 192$, $k = 128$):
  - Best differential distinguisher on 13 rounds when no difference are introduced at all in the tweakey. Best possible differential attack on $13 + d + d = 13 + 8 = 21$ rounds in the same context. If a difference is introduced in b lanes, then the best attack is on $21 + b + 1$ rounds, leading to the best possible differential attack when the p lanes have differences equal to $21 + 5 + 1 = 27$ rounds.
  - With the same reasoning, the best linear distinguisher is on 16 rounds. Then, the best possible linear attack is on $16 + d + d = 16 + 8 = 24$ rounds.
- LILLIPUT-TBC-I-192 ($t = 192$, $k = 192$):
  - Best differential distinguisher on 17 rounds when no difference are introduced at all in the tweakey. Best possible differential attack on $13 + d + d = 17 + 8 = 25$ rounds in the same context. If a difference is introduced in b lanes, then the best attack is on $25 + b + 1$ rounds, leading to the best possible differential attack when the p lanes have differences equal to $25 + 6 + 1 = 32$ rounds.
  - With the same reasoning, the best linear distinguisher is on 23 rounds (extrapolating the results of Table 4.7 up to 48 active S-boxes). Then, the best possible linear attack is on $23 + d + d = 23 + 8 = 31$ rounds.
- LILLIPUT-TBC-I-256 ($t = 192$, $k = 256$):
  - Best differential distinguisher on 24 rounds when no difference are introduced at all in the tweakey (always extrapolating the results of Table 4.7 up to 51 active S-boxes). Best possible differential attack on $24+d+d = 24+8 = 32$ rounds in the same context. If, a difference is introduced in b lanes, then the best attack is on $32 + b + 1$ rounds, leading to the best possible differential attack when the p lanes have differences equal to $32 + 7 + 1 = 40$ rounds.

  – With the same reasoning, the best linear distinguisher is on 30 rounds (extrapolating the results of Table 4.7 up to 64 active S-boxes). Then, the best possible linear attack is on $30 + d + d = 30 + 8 = 38$ rounds.

- LILLIPUT-TBC-II-128 ($t = 128$, $k = 128$): The bound for the differential distinguisher is the same than the one given for LILLIPUT-TBC-I-192: the best differential attack works on 21 rounds for the single tweakey model and on 26 rounds when the p lanes have differences. The bound for the linear cryptanalysis is the same than the one given for LILLIPUT-TBC-I-192: 24 rounds.

- LILLIPUT-TBC-II-192 ($t = 128$, $k = 192$): The bound for the differential distinguisher is the same than the one given for LILLIPUT-TBC-I-192: the best differential attack works on 25 rounds for the single tweakey model and on 31 rounds when the p lanes have differences. The bound for the linear cryptanalysis is the same than the one given for LILLIPUT-TBC-I-192: 31 rounds.

- LILLIPUT-TBC-II-256 ($t = 128$, $k = 256$): The bound for the differential distinguisher is the same than the one given for LILLIPUT-TBC-I-256: the best differential attack works on 32 rounds for the single tweakey model and on 39 rounds when the p lanes have differences. The bound for the linear cryptanalysis is the same than the one given for LILLIPUT-TBC-I-256: 38 rounds.

Table 4.7: Minimal number of active S-boxes for every round. AS_D corresponds to the minimal number of S-boxes reached for a differential attack. AS_L corresponds to the minimal number of S-boxes reached for a linear attack.

Round	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
AS_D	0	1	2	3	5	9	12	15	17	19	22	24	25	28	29	31	34	40	41	43
AS_L	0	1	2	3	5	8	12	13	15	17	19	22	25	27	30	32	34	38	40	42

Related Tweakey Boomerang Attacks

As the attacker could introduce differences both in the tweak and in the key and as our tweakey schedule is linear and could be completely computed according the introduced differences, we could imagine that in a related tweakey boomerang attack, the attacker could find a forward differential trail with $(p - 1)$ rounds containing no difference and could also find a backward differential trail with $(p - 1)$ rounds without difference. Thus, always considering that the key recovery part at the top of the related tweakey boomerang distinguisher has $d = 4$ rounds and also $d = 4$ rounds at the bottom, we could construct a related tweakey boomerang attack containing $2 \cdot (p - 1) + 8$ rounds at the beginning and at the end, and b rounds in its middle part.

Let us determine how many rounds is b for the different key lengths and considering that between the first differential trail on E_0 and the second differential trail on E_1 with $e = E_1 \circ E_0$, we have 2 rounds for free, one because the best coefficient of the BCT is equal to 1 and one because LILLIPUT-TBC is a Feistel-like scheme. Thus, as in a related tweakey boomerang attack, we associated the probability p of the differential trail for E_0 and q for the differential trail for E_1. Thus, we expect that the overall probability for $b - 2$ rounds is $p^2 q^2$.

Thus, for a 256-bit key, we want $p^2q^2 \leq 2^{-256}$ considering that using several keys and several tweaks we could go beyond the full codebook limit. This leads to $p \leq 2^{-64}$ considering that $p = q$. Thus, referring to Table 4.7 with $\delta_S = 2^{-5}$, the differential trail for E_0 has at most $64/5 = 12.8$ active S-boxes leading to a differential propagating on at most 7 rounds. We apply the same reasoning for E_1. Thus, the maximal number of rounds a related tweakey boomerang attack could reach is equal to $7 + 7 + 8 + 2 \cdot (p - 1) + 2 = 36$ for Lilliput-TBC-I-256 and to $7 + 7 + 8 + 2 \cdot (p - 1) + 2 = 34$ for Lilliput-TBC-II-256.

Thus, for a 192-bit key, we want $p^2q^2 \leq 2^{-192}$. This leads to $p \leq 2^{-48}$ considering that $p = q$ and to $48/5 = 9.6$ active S-boxes for both E_0 and E_1 leading to a differential propagating on at most 6 rounds. Thus, the maximal number of rounds a related tweakey boomerang attack could reach is equal to $6 + 6 + 8 + 2 \cdot (p - 1) + 2 = 32$ for Lilliput-TBC-I-192 and to $6 + 6 + 8 + 2 \cdot (p - 1) + 2 = 30$ for Lilliput-TBC-II-192.

Thus, for a 128-bit key, we want $p^2q^2 \leq 2^{-128}$. This leads to $p \leq 2^{-32}$ considering that $p = q$ and to $32/5 = 6.4$ active S-boxes for both E_0 and E_1 leading to a differential propagating on at most 5 rounds. Thus, the maximal number of rounds a related tweakey boomerang attack could reach is equal to $5 + 5 + 8 + 2 \cdot (p - 1) + 2 = 28$ for Lilliput-TBC-I-128 and to $5 + 5 + 8 + 2 \cdot (p - 1) + 2 = 26$ for Lilliput-TBC-II-128.

Structural Attacks

In this Subsection, we consider all the so-called structural attacks which include impossible differential attacks [BBS99], zero-correlation attacks [BGW+14], integral attacks [DKR97] and meet-in-the-middle attacks [DS08]. The security analysis of those attacks mainly depend on the diffusion delay d of the scheme as shown in [BMT14, SM10]. Indeed, no distinguisher could be found for structural attacks beyond $2d + 2$ rounds as full diffusion is reached. Those notions are also mainly linked with the computation of the super-pseudo random permutation advantage of the underlying scheme as shown in [HR10, BMT14].

Thus, in the single tweakey model where no difference at all is injected through the tweakey schedule, the best distinguisher could be constructed on $2d + 2$ rounds. To complete the attack, we could add for the key recovery part d rounds at the top of the distinguisher and d rounds at the bottom leading to a structural attack with a maximum of $4d + 2$ rounds. For all instances of Lilliput-TBC, this leads to the possibility of covering at most 18 rounds for all the structural attacks considered here. Note that this bound is overestimated compared to the one provided in [ST17] concerning the particular case of an impossible differential attack on the block cipher Lilliput.

Moreover, in the related tweakey model where we consider that an adversary can control at most the content of p lanes, the adversary could directly in this context attack $4d + 2 + p$ rounds at most. This leads to the following upper bounds on the possible number of attacked rounds for the 6 instances of Lilliput-TBC: 22 rounds for Lilliput-TBC-II-128, 23 rounds for Lilliput-TBC-II-192 and Lilliput-TBC-I-128, 24 rounds for Lilliput-TBC-II-256 and Lilliput-TBC-I-192, 25 rounds for Lilliput-TBC-I-256.

Division Property

The division property was proposed by Todo [Tod15b] as a generalization of the integral property to correctly evaluate higher-order integral property. The best division distinguisher described in [ST18] on the block cipher Lilliput is on 13 rounds leading to a key recovery attack on 17 rounds in the single tweakey model. Note that the `Linear` procedure presented in Algorithm

1 of [ST18] is the same for LILLIPUT-TBC, only the `NonLinear` part diverges in the way to compute the sets. The distinguisher presented in [ST18] studies an integral property on 63 input bits and on 1 output bit that completely maximize the possible number of implied bits. Thus, we conjecture that there is no distinguisher that exploits division properties on more than 26 rounds of LILLIPUT-TBC as in this last case, the number of possible input bits implied in a division property is doubled, *i.e.* equal to 127 with the same procedure describing the linear part. Thus, we are still confident that our proposals offer a strong security margin regarding this class of attacks.

Subspace Cryptanalysis

Invariant subspace cryptanalysis uses affine subspaces that are invariant throughout the cipher. Those attacks work particularly well in the context of simple tweakey/key schedules where the invariant properties stay valid through the key addition. Thus, to avoid this kind of attacks, invariant subspaces must be destroyed by the key/tweakey schedule. As the tweakey schedule of LILLIPUT-TBC is composed of ring-LFSRs ranging all the possible spaces, we conjecture that our non-trivial tweakey schedule provide a good protection against those attacks.

However, we give here the 9 linear structures of our S-box, *i.e.* the list (b, a, c) such that $b \cdot (S(x) \oplus S(x \oplus a)) = c$ with c a constant: (1, 32, 1), (1, 64, 0), (1, 96, 1), (4, 64, 1), (4, 128, 0), (4, 192, 1), (5, 64, 1), (5, 160, 1), (5, 224, 0). Note that none of those structures is preserved through two applications of the S-box.

Thus, with our non-trivial tweakey schedule and the fact that the invariant subspaces deduced from the linear structures can not be chained for many rounds, we conjecture that LILLIPUT-TBC is immune against this kind of attacks.

Algebraic Attacks

Before deducing bounds for algebraic attacks, let us describe the algebraic properties of the S-box. The S-box has a maximal degree of 6 and a minimal degree of 4. Our S-box could also be described using $e = 14$ quadratic equations in the 16 input/output variables over $GF(2)$. Thus, from Table 4.7, we could see that for 13 rounds, we have 26 active S-boxes, it means that, from this bound the number of induced variables by the algebraic expression of the cipher LILLIPUT-TBC is greater than the block size.

Moreover, as our S-box S could be described with 14 quadratic equations in 16 variables, it means that the number of quadratic equations induced by a round is $14 \times 8 = 112$ quadratic equations in $16 \times 8 = 128$ variables and for 32 rounds of LILLIPUT-TBC, we thus obtain 3584 quadratic equations in 4096 variables. Thus, we obtain an under-determined system with more variables than for the AES.

Using those arguments, we conjecture that LILLIPUT-TBC is immune against algebraic attacks.

Differential Fault Analysis in Middle Rounds

We want to protect LILLIPUT-TBC against differential fault analysis. Such attacks consist in injecting faults in one of the last rounds of the encryption, and exploit pairs of faulty and correct ciphertexts. A common countermeasure consists in doubling the execution of a few last rounds in order to detect a fault. In the case of a fault injection – unless the attacker is able to inject twice the same fault in a very short period of time – the doubling results in two ciphertexts, one faulty and the other not. Such a result is detected and no output is given. In order to be detected, the fault must be injected during the rounds that are doubled. If a fault occurs before, the faulty state is copied and processed twice, resulting in two identical faulty ciphertexts which will be outputted, making the countermeasure ineffective. For this reason, it is important to protect enough rounds to prevent such attacks. It is also important to evaluate closely the number of rounds to protect as doubling increases time computation or surface area.

We analyze how much rounds must be protected in order to prevent the attack from Rivain [Riv09] adapted to Lilliput-TBC. This attack takes advantage of the Feistel scheme in order to inject fault in middle rounds and observe differential distributions in the last one.

As Lilliput-TBC is based on a Feistel scheme, we will denote the state at output of round i as (L^i, R^i), L^i and R^i being its 64-bit left and right parts respectively[4]. Hence, the plaintext is (L^0, R^0), and the ciphertext (L^r, R^r). The fact that the number of rounds r changes with the mode used does not change anything about the following results, as r is always greater than the number of rounds to protect. Notice that the subtweakey byte used in each non linear function F_j of Lilliput-TBC $(0 \leq j \leq 7)$ is the xor of a known constant and p byte values $(4 \leq p \leq 7)$ that come from some known tweak-dependent lanes and other unknown key-dependent ones. The following analysis focuses on retrieving this subtweakey byte value only, leaving uncertainty about key-lane bytes. A "successful" attack thus leaves the attacker with 2^8 pairs of key-dependent bytes (or 2^{16} triplets or 2^{32} quadruplets, depending on the key length).

As in [Riv09] we only consider faults in the left part of the state as it is the most efficient way to retrieve the subkey in a Feistel scheme. Injecting a fault ϵ in L^i induces changes in the next rounds and results in a faulty ciphertext $\widetilde{C}$. The correct ciphering of the same plaintext is denoted C. The main goal is to observe the distribution of $\Delta = L^{r-1} \oplus \widetilde{L}^{r-1}$, which is the xor of both left parts of the correct and faulty states before the last round. It is possible to compute each byte of $\Delta = (\delta_7, \dots, \delta_0)$ as a function of the correct/faulty ciphertexts and a guess g on the corresponding subtweakey byte in the last round. Denoting $L^i = (\ell^i_7, \dots, \ell^i_0)$ and $\widetilde{L}^i = (\widetilde{\ell}^i_7, \dots, \widetilde{\ell}^i_0)$ – and similarly with r^i_j and $\widetilde{r}^i_j$ for the right parts – we infer:

$$\delta_0 = \ell^r_0 \oplus \widetilde{\ell}^r_0 \oplus S(r^r_7 \oplus g) \oplus S(\widetilde{r}^r_7 \oplus g) \tag{4.6}$$

$$\delta_j = \ell^r_j \oplus \widetilde{\ell}^r_j \oplus S(r^r_{7-j} \oplus g) \oplus S(\widetilde{r}^r_{7-j} \oplus g) \oplus r^r_7 \oplus \widetilde{r}^r_7 \qquad \text{for } 1 \leq j \leq 6 \tag{4.7}$$

$$\begin{aligned}\delta_7 = {}& \ell^r_7 \oplus \widetilde{\ell}^r_7 \oplus S(r^r_0 \oplus g) \oplus S(\widetilde{r}^r_0 \oplus g) \oplus r^r_7 \oplus \widetilde{r}^r_7 \\ & \oplus r^r_6 \oplus \widetilde{r}^r_6 \oplus r^r_5 \oplus \widetilde{r}^r_5 \oplus r^r_4 \oplus \widetilde{r}^r_4 \oplus r^r_3 \oplus \widetilde{r}^r_3 \oplus r^r_2 \oplus \widetilde{r}^r_2 \oplus r^r_1 \oplus \widetilde{r}^r_1\end{aligned} \tag{4.8}$$

Depending on the round where the fault is injected, the attacker might be able to know the distribution of Δ. For example, if a fault ϵ is injected in L^{r-3}, then $\Delta = \epsilon$. If the attacker knows ϵ, he can check whether the g-dependent Δ value calculated from the previous equations equals ϵ or not, discarding subtweakey candidates that do not. Note that due to the byte oriented scheme of Lilliput-TBC, the attack can be done on each subtweakey byte independently, allowing to guess one byte and calculate one δ byte at a time. For this reason, the rest of the analysis focuses on a byte δ rather than on the whole Δ.

If the attacker is able to systematically fault with the same known ϵ in earliest rounds, he can build (in an offline phase) approximations of the theoretic distribution of any δ_j. Given N pairs of correct/faulty ciphertexts $(C, \widetilde{C})_N$, the attack then consists in calculating, for each candidate g, the corresponding empirical distribution of δ_j and select the one that is the most similar to the theoretic one. This can be done in a maximum-likelihood manner for instance.

In the case where the attacker is not able to predict the distribution of δ_j, he can still expect it to be biased for the correct key guess, and to be uniform for others (wrong-key

[4]With notations of Section 4.2.3 we have $L = (X_{15}, \dots, X_8)$ and $R = (X_7, \dots, X_0)$.

assumption). Similarly he can compute the empirical distributions of δ_j and select the one that is the farthest from the uniform distribution. This can be done with the Squared Euclidean Imbalance distinguisher for example.

In order to infer the number of rounds to protect, we have considered a strong attacker who is able to inject bit flips at the bit position of its choice. We then conducted attacks, injecting faults sooner and sooner until the attack becomes unfeasible.

Simulation Results and Recommendation We have simulated the differential fault analysis where a bit flip fault is injected at a precise round $r-s$ and at a chosen bit position b on LILLIPUT-TBC-I-128[5]. We have studied both the non profiled case where the distinguisher is the Squared Euclidean Imbalance of the observed distribution of δ_j, and the profiled case based on the maximum-likelihood of this distribution with respect to (an approximation of) the theoretic one. For sake of clarity, with our notations, when the attacker observes the distribution of byte δ_j, it helps him to recover the subtweakey byte RTK_{7-j}. Our objective is to determine the largest value of s for which we suspect that a fault attack can be realized. The fault bit position b is numbered from 0 to 63 which respectively denote the least significant bit of $X_8 = \ell_0$ and the most significant bit of $X_{15} = \ell_7$. For any set of parameters (round gap s, fault bit position b, attacked subtweakey byte position $(7-j)$, number of faults N, profiled/non profiled setting), our results are expressed as the average success rate on 1000 runs.

We first observed that for $s \leq 6$, the fault attack is somewhat easy. For $s = 6$, and for $N = 1000$ faults, there always exists a fault bit position for which the success rate is 1.0 for all positions j except for $j = 7$. Note that we have systematically observed that the subtweakkey byte number 0 ($j = 7$) is the most difficult to retrieve. For $j = 7$, the success rate may still be as large as 0.873 (depending on b) in the profiled case. An interesting observation is that the success of the attack greatly depends on the fault bit position. As we consider that the attacker can choose b, we think that the relevant criteria is the maximum success rate taken on all $b = 0 \dots 63$ values.

Table 4.8 and Table 4.9 present results for the non-profiled and the profiled settings respectively. We have considered a number of faults N belonging to the set $\{10^3, 3.10^3, 10^4, 3.10^4, 10^5, 3.10^5, 10^6, 3.10^6, 10^7\}$ for $s = 7$, and to the set $\{10^5, 3.10^5, 10^6, 3.10^6, 10^7\}$ for $s = 8$. Without surprise, one can observe that the profiled attack is more efficient than the non-profiled one. For $s = 7$ all subtweakey bytes except for $j = 7$ can be retrieved with about 10^5 faults. Even with only about 3000 faults two subtweakey bytes (for $j = 1$ and $j = 6$) can be recovered. We also observe that for $s = 8$ the attack does not work, even in the profiled case and even with ten millions faults, whatever the bit fault position and whatever the attacked byte.

Based on our simulation results, we recommend to protect LILLIPUT-TBC against differential fault analysis by doubling the execution of a minimum of seven last rounds.

Security Evaluation Summary

Table 4.10 gives a security evaluation summary for all the instances of LILLIPUT-TBC. From this table, we are able to say that each instance has a sufficient security margin (given in the last column).

Surprisingly, classical attacks such as differential and linear attacks reach more rounds than structural attacks for LILLIPUT-TBC. This is mainly linked with the choice of a Feistel-like scheme with a good diffusion.

[5] We guess that similar results would have been obtained for other versions.

round	faults	attacked subtweakey byte : RTK_{7-j}							
		j=0	j=1	j=2	j=3	j=4	j=5	j=6	j=7
$s = 6$	10^3	1.0	1.0	1.0	1.0	1.0	1.0	1.0	0.032
$s = 7$	10^3	0.009	0.014	0.011	0.008	0.010	0.008	0.008	0.011
	3.10^3	0.008	0.078	0.010	0.009	0.011	0.009	0.021	0.008
	10^4	0.009	0.523	0.027	0.012	0.010	0.010	0.160	0.008
	3.10^4	0.022	0.764	0.303	0.030	0.013	0.011	0.684	0.011
	10^5	0.381	1.0	0.998	0.352	0.017	0.020	0.994	0.009
	3.10^5	1.0	1.0	1.0	0.506	0.064	0.130	1.0	0.009
	10^6	1.0	1.0	1.0	0.891	0.666	0.878	1.0	0.010
	3.10^6	1.0	1.0	1.0	1.0	1.0	1.0	1.0	0.008
	10^7	1.0	1.0	1.0	1.0	1.0	1.0	1.0	0.008
$s = 8$	10^5	0.010	0.008	0.011	0.011	0.008	0.009	0.012	0.009
	3.10^5	0.008	0.011	0.008	0.008	0.009	0.008	0.008	0.009
	10^6	0.009	0.009	0.010	0.013	0.010	0.012	0.008	0.009
	3.10^6	0.008	0.008	0.011	0.011	0.010	0.008	0.010	0.009
	10^7	0.012	0.009	0.009	0.008	0.009	0.008	0.008	0.011

Table 4.8: Experimental success rate of non profiled (Squared Euclidean Imbalance) differential fault analysis

round	faults	attacked subtweakey byte : RTK_{7-j}							
		j=0	j=1	j=2	j=3	j=4	j=5	j=6	j=7
$s = 6$	10^3	1.0	1.0	1.0	1.0	1.0	1.0	1.0	0.837
$s = 7$	10^3	0.036	0.448	0.076	0.023	0.012	0.013	0.226	0.009
	3.10^3	0.083	0.730	0.275	0.076	0.019	0.019	0.668	0.010
	10^4	0.336	0.990	0.825	0.291	0.046	0.053	0.961	0.009
	3.10^4	0.875	1.0	1.0	0.555	0.113	0.180	1.0	0.009
	10^5	1.0	1.0	1.0	0.728	0.497	0.669	1.0	0.012
	3.10^5	1.0	1.0	1.0	0.992	0.965	0.992	1.0	0.010
	10^6	1.0	1.0	1.0	1.0	1.0	1.0	1.0	0.011
	3.10^6	1.0	1.0	1.0	1.0	1.0	1.0	1.0	0.009
	10^7	-	-	-	-	-	-	-	-
$s = 8$	10^5	0.010	0.007	0.009	0.007	0.009	0.011	0.010	0.011
	3.10^5	0.009	0.009	0.009	0.009	0.009	0.007	0.007	0.009
	10^6	0.009	0.012	0.008	0.011	0.010	0.010	0.009	0.011
	3.10^6	0.009	0.009	0.008	0.008	0.009	0.010	0.009	0.010
	10^7	0.009	0.008	0.011	0.008	0.010	0.009	0.008	0.012

Table 4.9: Experimental success rate of profiled (Maximum Likelihood) differential fault analysis

	STKM			RTKM				Nb rounds	Sec. Margin
	Diff.	Lin.	Struct.	Diff.	Lin.	RTKB	Struct.	(r)	(in rounds)
Lilliput-TBC-I-128	21	24	18	27	24	28	23	32	4
Lilliput-TBC-I-192	25	31	18	32	31	32	24	36	4
Lilliput-TBC-I-256	32	38	18	40	38	36	25	42	2
Lilliput-TBC-II-128	21	24	18	26	24	26	22	32	6
Lilliput-TBC-II-192	25	31	18	31	31	30	23	36	5
Lilliput-TBC-II-256	32	38	18	39	38	34	24	42	3

Table 4.10: Security Evaluation summary (“paranoid” case). STKM means “Single Tweakey Model”, RTKM means “Related Tweakey Model” and RTKB means “Related Tweakey Boomerang attack”.

4.4 Implementations

Lilliput-AE is suited to be implemented efficiently on a wide range of processors (especially those embedded in IoT platforms) and in hardware. This section will provide insights on efficient implementation methods, especially on 8-bit processors where Lilliput-AE is by design well adapted due to its byte-oriented nature. Many good properties on 8-bit platforms are also valid on 16-bit and 32-bit platforms.

4.4.1 Software Implementations

In this section, we give some possible variants for implementing Lilliput-AE. We take as reference IoT platforms those from the Felics (Fair Evaluation of LIghtweight Cryptographic Systems) framework [DCK+15]: 8-bit Atmel AVR ATmega128, 16-bit Texas Instruments MSP430F1611 and 32-bit Arduino Due ARM Cortex-M3. When useful, binary code size, extscram, and execution time optimizations will be discussed.

On an 8-bit processor, Lilliput-AE can be programmed by simply implementing the different component transformations.

Round Function `OneRoundEGFN`

If we look at the `OneRoundEGFN` the round function, `NonLinearLayer` function is only made of S-boxes computation, with previous subtweakey addition. `LinearLayer` function is just successive byte additions on x_{15} followed by byte additions on most significant bytes of `OneRoundEGFN` internal state. Finally, a byte-oriented permutation, `PermutationLayer`, is computed.

A straightforward implementation is then easy to implement on 8-bit processors. The implementation of all the F_i functions requires a table of 256 bytes. Since this table is fixed, it can be easily stored in EEPROM data. As mentioned in Section 4.3.2, S has been chosen to be easily masked in hardware and software.

Concerning code size, `OneRoundEGFN` can be easily computed with loops in order to save ROM program space. For example, Algorithm 5 shows that `NonLinearLayer` only needs one additional intermediate register in total to store the successive results of $SK_{7-i} \oplus x_{7-i}$ and the S computation on it: RAM stack usage is then minimal.

Similarly, Algorithm 6 shows that `LinearLayer` can be also implemented by `XOR` additions and accumulations.

Algorithm 5 x_{8+i} computation in `Non-Linear Layer`

```
for $i = 0$ to $7$ do
    $x_{8+i} \leftarrow x_{8+i} \oplus S(SK_{7-i} \oplus x_{7-i})$
return $(x_{15}, x_{14}, \cdots, x_8)$
```

Algorithm 6 x_{8+i} computation in `Linear Layer`

```
for $i = 0$ to $7$ do
    $x_{15} \leftarrow x_{15} \oplus x_i$
for $i = 0$ to $6$ do
    $x_{14-i} \leftarrow x_{14-i} \oplus x_7$
return $(x_{15}, x_{14}, \cdots, x_8)$
```

Finally, `PermutationLayer` can be simply implemented as a series of `MOV` operations. For example, in encryption mode: $x_{13} \leftarrow x_0$, $x_9 \leftarrow x_1$, $\cdots$, $x_7 \leftarrow x_{15}$.

Overall, a straightforward computation of `OneRoundEGFN` (which is the same for every TK size) needs 29 `XOR`s (21 in the datapath plus 8 before each S-box computation), 8 accesses in EEPROM memory for S-box computations, and 16 `MOV` operations for `PermutationLayer`, *i.e.* only 53 operations in total (29 arithmetic operations and 24 memory operations). The footprint on RAM stack, ROM program (as discussed earlier in this subsection) and on EEPROM data (256 bytes) of `OneRoundEGFN` is very lightweight for 8-bit platforms.

Tweakey Schedule

The tweakey schedule can be decomposed into two distinct functions:

- the *extraction* function, which is called r times to produce subtweakey RTK^i from the tweakey state TK^i, $\forall i \in \{0, \cdots, r-1\}$,
- the *update* function, which is called $r-1$ times to compute TK^{i+1} from TK^i, $\forall i \in \{0, \cdots, r-2\}$.

The update function consists in one multiplication α_i per lane. Each of these multiplications takes a different amount of operations to complete. Table 4.11 summarizes which multiplications are needed for each variant of LILLIPUT-TBC.

The extraction function consists in:

- XORing all p 64-bit lanes together bytewise: this requires $(p-1)$ 64-bit `XOR`s, hence $8 \times (p-1)$ 8-bit `XOR`s,

Name	k	t	p	Required α_i
LILLIPUT-TBC-I-128	128	192	5	α_0 to α_4
LILLIPUT-TBC-I-192	192	192	6	α_0 to α_5
LILLIPUT-TBC-I-256	256	192	7	α_0 to α_6
LILLIPUT-TBC-II-128	128	128	4	α_0 to α_3
LILLIPUT-TBC-II-192	192	128	5	α_0 to α_4
LILLIPUT-TBC-II-256	256	128	6	α_0 to α_5

Table 4.11: Multiplications needed for each variant of LILLIPUT-TBC

- XORing the resulting 64-bit word with the round constant C^i: this requires a single 8-bit `XOR`, since C^i fits on 8 bits.

This function thus requires $8 \times (p-1) + 1$ `XOR` operations.

4-lane case The following multiplications are needed to process four lanes: $\alpha_0 = M$, $\alpha_1 = M^2$, $\alpha_2 = M^3$, and $\alpha_3 = M^4$. As we will do for further number of lanes, we will develop the matrix relations to evaluate precisely the number of required operations.

Multiplication α_0 of vector $x = (x_7, x_6, \cdots, x_0)^t$ by matrix M can be expressed as:

$$\begin{pmatrix} y_7 \\ y_6 \\ y_5 \\ y_4 \\ y_3 \\ y_2 \\ y_1 \\ y_0 \end{pmatrix} = \begin{pmatrix} x_6 \\ x_5 \\ x_5 \ll 3 \oplus x_4 \\ x_4 \gg 3 \oplus x_3 \\ x_2 \\ x_6 \ll 2 \oplus x_1 \\ x_0 \\ x_7 \end{pmatrix}$$

Multiplication α_0 will thus require 14 operations in total:

- 3 shift operations,
- 3 `XOR`s,
- 8 assignments.

Multiplication α_1 is represented by matrix M^2, which corresponds to two successive applications of matrix M. Let us denote $M \cdot x$ as $a = (a_7, \cdots, a_0)^t$:

$$\begin{pmatrix} a_7 \\ a_6 \\ a_5 \\ a_4 \\ a_3 \\ a_2 \\ a_1 \\ a_0 \end{pmatrix} = \begin{pmatrix} x_6 \\ x_5 \\ x_5 \ll 3 \oplus x_4 \\ x_4 \gg 3 \oplus x_3 \\ x_2 \\ x_6 \ll 2 \oplus x_1 \\ x_0 \\ x_7 \end{pmatrix}$$

$y = M^2 \cdot x = M \cdot a$ can then be expressed as:

$$\begin{pmatrix} y_7 \\ y_6 \\ y_5 \\ y_4 \\ y_3 \\ y_2 \\ y_1 \\ y_0 \end{pmatrix} = \begin{pmatrix} a_6 \\ a_5 \\ a_5 \ll 3 \oplus a_4 \\ a_4 \gg 3 \oplus a_3 \\ a_2 \\ a_6 \ll 2 \oplus a_1 \\ a_0 \\ a_7 \end{pmatrix}$$

Some components of a are simply permuted components of x; others (namely, a_5, a_4 and a_2) result from a linear combination of components of x. Some of these combinations contribute to more than one components of y: specifically, $a_5 = x_5 \lll 3 \oplus x_4$ and $a_4 = x_4 \ggg 3 \oplus x_3$.

To minimize the number of operations, we can thus spend some registers to store a_5 and a_4. The final expression for $y = M^2 \cdot x$ then becomes:

$$\begin{pmatrix} y_7 \\ y_6 \\ y_5 \\ y_4 \\ y_3 \\ y_2 \\ y_1 \\ y_0 \end{pmatrix} = \begin{pmatrix} a_6 \\ a_5 \\ a_5 \lll 3 \oplus a_4 \\ a_4 \ggg 3 \oplus a_3 \\ a_2 \\ a_6 \lll 2 \oplus a_1 \\ a_0 \\ a_7 \end{pmatrix} = \begin{pmatrix} x_5 \\ a_5 \\ a_5 \lll 3 \oplus a_4 \\ a_4 \ggg 3 \oplus x_2 \\ x_6 \lll 2 \oplus x_1 \\ x_5 \lll 2 \oplus x_0 \\ x_7 \\ x_6 \end{pmatrix}$$

Multiplication α_1 will thus require 22 operations in total:

- 2 `XOR`s, 2 shifts and 2 assignments for a_5 and a_4,
- 4 `XOR`s, 4 shifts and 8 byte assignments.

Multiplication α_2 is represented by matrix M^3, which corresponds to three successive applications of matrix M. Let us denote $M^2 \cdot x$ as $b = (b_7, \cdots, b_0)^t$:

$$\begin{pmatrix} b_7 \\ b_6 \\ b_5 \\ b_4 \\ b_3 \\ b_2 \\ b_1 \\ b_0 \end{pmatrix} = \begin{pmatrix} a_6 \\ a_5 \\ a_5 \lll 3 \oplus a_4 \\ a_4 \ggg 3 \oplus a_3 \\ a_2 \\ a_6 \lll 2 \oplus a_1 \\ a_0 \\ a_7 \end{pmatrix}$$

$y = M^3 \cdot x = M \cdot b$ can then be expressed as:

$$\begin{pmatrix} y_7 \\ y_6 \\ y_5 \\ y_4 \\ y_3 \\ y_2 \\ y_1 \\ y_0 \end{pmatrix} = \begin{pmatrix} b_6 \\ b_5 \\ b_5 \lll 3 \oplus b_4 \\ b_4 \ggg 3 \oplus b_3 \\ b_2 \\ b_6 \lll 2 \oplus b_1 \\ b_0 \\ b_7 \end{pmatrix}$$

As with α_1, we can isolate components of b which satisfy the following constraints:

1. they result from a linear combination of more than one components of a,
2. they contribute to more than one components of y.

b_5, b_4 and b_2 satisfy constraint 1; among those, only $b_5 = a_5 \lll 3 \oplus a_4$ and $b_4 = a_4 \ggg 3 \oplus a_3 = a_4 \ggg 3 \oplus x_2$ satisfy constraint 2. To implement α_2 using as few operations as necessary, we thus need to:

1. pre-compute a_5 and a_4,
2. pre-compute b_5 and b_4,
3. compute y as follows:

$$\begin{pmatrix} y_7 \\ y_6 \\ y_5 \\ y_4 \\ y_3 \\ y_2 \\ y_1 \\ y_0 \end{pmatrix} = \begin{pmatrix} b_6 \\ b_5 \\ b_5 \ll 3 \oplus b_4 \\ b_4 \gg 3 \oplus b_3 \\ b_2 \\ b_6 \ll 2 \oplus b_1 \\ b_0 \\ b_7 \end{pmatrix} = \begin{pmatrix} a_5 \\ b_5 \\ b_5 \ll 3 \oplus b_4 \\ b_4 \gg 3 \oplus a_2 \\ a_6 \ll 2 \oplus a_1 \\ a_5 \ll 2 \oplus a_0 \\ a_7 \\ a_6 \end{pmatrix} = \begin{pmatrix} a_5 \\ b_5 \\ b_5 \ll 3 \oplus b_4 \\ b_4 \gg 3 \oplus x_6 \ll 2 \oplus x_1 \\ x_5 \ll 2 \oplus x_0 \\ a_5 \ll 2 \oplus x_7 \\ x_6 \\ x_5 \end{pmatrix}$$

Multiplication α_2 will thus require 30 operations in total:

- 2 `XOR`s, 2 shifts and 2 assignments for a_5 and a_4,
- 2 `XOR`s, 2 shifts and 2 assignments for b_5 and b_4,
- 5 `XOR`s, 5 shifts and 8 byte assignments.

Multiplication α_3 is represented by matrix M^4, which corresponds to four successive applications of matrix M. Let us denote $M^3 \cdot x$ as $c = (c_7, \cdots, c_0)^t$:

$$\begin{pmatrix} c_7 \\ c_6 \\ c_5 \\ c_4 \\ c_3 \\ c_2 \\ c_1 \\ c_0 \end{pmatrix} = \begin{pmatrix} b_6 \\ b_5 \\ b_5 \ll 3 \oplus b_4 \\ b_4 \gg 3 \oplus b_3 \\ b_2 \\ b_6 \ll 2 \oplus b_1 \\ b_0 \\ b_7 \end{pmatrix}$$

$y = M^4 \cdot x = M \cdot c$ can then be expressed as:

$$\begin{pmatrix} y_7 \\ y_6 \\ y_5 \\ y_4 \\ y_3 \\ y_2 \\ y_1 \\ y_0 \end{pmatrix} = \begin{pmatrix} c_6 \\ c_5 \\ c_5 \ll 3 \oplus c_4 \\ c_4 \gg 3 \oplus c_3 \\ c_2 \\ c_6 \ll 2 \oplus c_1 \\ c_0 \\ c_7 \end{pmatrix}$$

c_4 and c_5 are the only components of c which result from a linear combination of more than one components of b, while contributing to more than one components of y. Therefore, to implement α_3 using as few operations as necessary, we need to:

1. pre-compute a_5 and a_4,

2. pre-compute b_5 and b_4,
3. pre-compute c_5 and c_4,
4. compute y as follows:

$$\begin{pmatrix} y_7 \\ y_6 \\ y_5 \\ y_4 \\ y_3 \\ y_2 \\ y_1 \\ y_0 \end{pmatrix} = \begin{pmatrix} c_6 \\ c_5 \\ c_5 \ll 3 \oplus c_4 \\ c_4 \gg 3 \oplus c_3 \\ c_2 \\ c_6 \ll 2 \oplus c_1 \\ c_0 \\ c_7 \end{pmatrix} = \begin{pmatrix} b_5 \\ c_5 \\ c_5 \ll 3 \oplus c_4 \\ c_4 \gg 3 \oplus b_2 \\ b_6 \ll 2 \oplus b_1 \\ b_5 \ll 2 \oplus b_0 \\ b_7 \\ b_6 \end{pmatrix} = \begin{pmatrix} b_5 \\ c_5 \\ c_5 \ll 3 \oplus c_4 \\ c_4 \gg 3 \oplus x_5 \ll 2 \oplus x_0 \\ a_5 \ll 2 \oplus x_7 \\ b_5 \ll 2 \oplus x_6 \\ x_5 \\ a_5 \end{pmatrix}$$

Multiplication α_3 will thus require 39 operations in total:

- 2 `XOR`s, 2 shifts and 2 assignments for a_5 and a_4,
- 2 `XOR`s, 2 shifts and 2 assignments for b_5 and b_4,
- 2 `XOR`s, 3 shifts and 1 assignment for c_4,
- 1 `XOR`, 1 shift and 1 assignment for c_5,
- 5 `XOR`s, 5 shifts and 8 byte assignments.

To sum up, **for the 4-lane case**, the multiplications by α_0, α_1, α_2 and α_3 require $14 + 22 + 30 + 39 =$ **105 operations in total**.

Taking into account the $8 \times (p-1) + 1 = 8 \times 3 + 1 = 25$ operations needed for the extraction function, this leads to $105 + 25 =$ **130 operations for the whole subtweakey computation**.

5-lane case To process five lanes, multiplications $\alpha_0 = M$, $\alpha_1 = M^2$, $\alpha_2 = M^3$, $\alpha_3 = M^4$ (already described) and $\alpha_4 = M_R$ are needed.

Multiplication α_4 of vector $x = (x_0, x_1, \cdots, x_7)^t$ by matrix M_R (as explained in Section 4.2.3, we invert the direction of binary notations when dealing with M_R) can be expressed as:

$$\begin{pmatrix} y_0 \\ y_1 \\ y_2 \\ y_3 \\ y_4 \\ y_5 \\ y_6 \\ y_7 \end{pmatrix} = \begin{pmatrix} x_1 \\ x_2 \\ x_3 \oplus x_4 \ll 3 \\ x_4 \\ x_5 \oplus x_6 \gg 3 \\ x_3 \gg 2 \oplus x_6 \\ x_7 \\ x_0 \end{pmatrix}$$

Multiplication α_4 will thus require 14 operations in total:

- 3 `XOR`s and 3 shifts,
- 8 byte assignments.

To sum up, **for the 5-lane case**, the update function will require 105 operations (cf. 4-lane case) plus 14 operations for α_4, *i.e.* 119 operations. After adding $8 \times (5-1) + 1 = 33$ `XOR`s for the extraction function, we reach **138 operations for the whole subtweakey computation**.

6-lane case To process six lanes, multiplications $\alpha_0 = M$, $\alpha_1 = M^2$, $\alpha_2 = M^3$, $\alpha_3 = M^4$, $\alpha_4 = M_R$ (already described) and $\alpha_5 = M_R^2$ are needed.

Multiplication α_5 of vector $x = (x_0, x_1, \cdots, x_7)^t$ by matrix M_R^2 corresponds to two successive applications of matrix M_R. Let us denote $M_R \cdot x$ as $a = (a_0, \cdots, a_7)^t$:

$$\begin{pmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \\ a_4 \\ a_5 \\ a_6 \\ a_7 \end{pmatrix} = \begin{pmatrix} x_1 \\ x_2 \\ x_3 \oplus x_4 \ll 3 \\ x_4 \\ x_5 \oplus x_6 \gg 3 \\ x_3 \gg 2 \oplus x_6 \\ x_7 \\ x_0 \end{pmatrix}$$

$y = M_R^2 \cdot x = M_R \cdot a$ can then be expressed as:

$$\begin{pmatrix} y_0 \\ y_1 \\ y_2 \\ y_3 \\ y_4 \\ y_5 \\ y_6 \\ y_7 \end{pmatrix} = \begin{pmatrix} a_1 \\ a_2 \\ a_3 \oplus a_4 \ll 3 \\ a_4 \\ a_5 \oplus a_6 \gg 3 \\ a_3 \gg 2 \oplus a_6 \\ a_7 \\ a_0 \end{pmatrix}$$

As with α_1, α_2 and α_3, we will pre-compute components of a which depend on more than one components of x and contribute to more than one components of y. For α_5, this singles out $a_4 = x_5 \oplus x_6 \gg 3$. We end up with the following expression for y:

$$\begin{pmatrix} y_0 \\ y_1 \\ y_2 \\ y_3 \\ y_4 \\ y_5 \\ y_6 \\ y_7 \end{pmatrix} = \begin{pmatrix} a_1 \\ a_2 \\ a_3 \oplus a_4 \ll 3 \\ a_4 \\ a_5 \oplus a_6 \gg 3 \\ a_3 \gg 2 \oplus a_6 \\ a_7 \\ a_0 \end{pmatrix} = \begin{pmatrix} x_2 \\ x_3 \oplus x_4 \ll 3 \\ x_4 \oplus a_4 \ll 3 \\ a_4 \\ x_3 \gg 2 \oplus x_6 \oplus x_7 \gg 3 \\ x_4 \gg 2 \oplus x_7 \\ x_0 \\ x_1 \end{pmatrix}$$

Multiplication α_5 will thus require 21 operations in total:

- 1 `XOR`, 1 shift and 1 assignment for a_4,
- 5 `XOR`s, 5 shifts and 8 byte assignments.

To sum up, **for the 6-lane case**, the update function will require 119 operations (cf. 5-lane case) plus 21 operations for α_5, *i.e.* 140 operations. After adding $8 \times (6-1) + 1 = 41$ `XOR`s for the extraction function, we reach **181 operations for the whole subtweakey computation**.

7-lane case Finally, to process seven lanes, multiplications $\alpha_0 = M$, $\alpha_1 = M^2$, $\alpha_2 = M^3$, $\alpha_3 = M^4$, $\alpha_4 = M_R$, $\alpha_5 = M_R^2$ (already described) and $\alpha_6 = M_R^3$ are needed.

Multiplication α_6 of vector $x = (x_0, x_1, \cdots, x_7)^t$ by matrix M_R^3 corresponds to three successive applications of M_R. Let us denote $M_R^2 \cdot x$ as $b = (b_0, \cdots, b_7)$:

$$\begin{pmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \\ b_4 \\ b_5 \\ b_6 \\ b_7 \end{pmatrix} = \begin{pmatrix} a_1 \\ a_2 \\ a_3 \oplus a_4 \ll 3 \\ a_4 \\ a_5 \oplus a_6 \gg 3 \\ a_3 \gg 2 \oplus a_6 \\ a_7 \\ a_0 \end{pmatrix}$$

$y = M_R^3 \cdot x = M_R \cdot b$ can then be expressed as:

$$\begin{pmatrix} y_0 \\ y_1 \\ y_2 \\ y_3 \\ y_4 \\ y_5 \\ y_6 \\ y_7 \end{pmatrix} = \begin{pmatrix} b_1 \\ b_2 \\ b_3 \oplus b_4 \ll 3 \\ b_4 \\ b_5 \oplus b_6 \gg 3 \\ b_3 \gg 2 \oplus b_6 \\ b_7 \\ b_0 \end{pmatrix}$$

Only $b_4 = a_5 \oplus a_6 \gg 3 = x_3 \gg 2 \oplus x_6 \oplus x_7 \gg 3$ depends on more than one components of a while contributing to more than one components of y. To implement α_6 using as few operations as necessary, we thus need to:

1. pre-compute a_4
2. pre-compute b_4,
3. compute y as follows:

$$\begin{pmatrix} y_0 \\ y_1 \\ y_2 \\ y_3 \\ y_4 \\ y_5 \\ y_6 \\ y_7 \end{pmatrix} = \begin{pmatrix} b_1 \\ b_2 \\ b_3 \oplus b_4 \ll 3 \\ b_4 \\ b_5 \oplus b_6 \gg 3 \\ b_3 \gg 2 \oplus b_6 \\ b_7 \\ b_0 \end{pmatrix} = \begin{pmatrix} a_2 \\ a_3 \oplus a_4 \ll 3 \\ a_4 \oplus b_4 \ll 3 \\ b_4 \\ a_5 \oplus a_6 \gg 3 \oplus a_7 \gg 3 \\ a_4 \gg 2 \oplus a_7 \\ a_0 \\ a_1 \end{pmatrix} = \begin{pmatrix} x_3 \oplus x_4 \ll 3 \\ x_4 \oplus a_4 \ll 3 \\ a_4 \oplus b_4 \ll 3 \\ b_4 \\ x_3 \gg 2 \oplus x_6 \oplus x_7 \gg 3 \\ a_4 \gg 2 \oplus x_0 \\ x_1 \\ x_2 \end{pmatrix}$$

Overall, **multiplication $\boldsymbol{\alpha_6}$ will require 28 operations:**

- 1 `XOR`, 1 shift and 1 assignment for a_4,
- 2 `XOR`s, 2 shifts and 1 assignment for b_4,

- 6 `XOR`s, 6 shifts and 8 byte assignments.

To sum up, **for the 7-lane case**, the update function will require 140 operations (cf. 6-lane case) plus 28 operations for α_6, *i.e.* 168 operations. After adding $8 \times (7-1) + 1 = 49$ `XOR`s for the extraction function, we reach **217 operations for the whole subtweakey computation**.

Possible Trade-Offs

Another implementation of the tweakey schedule's update function can save some program space at the expense of extra RAM usage and latency. To multiply a lane x by matrix M (resp. M_R) raised to the power of $n > 1$, one can multiply x by M (resp. M_R) n times, instead of using the ad-hoc expressions given in Section 4.4.1. This allows the implementer to re-use the code for α_0 (resp. α_4) in order to compute α_1, α_2 and α_3 (resp. α_5 and α_6), although computing and storing each byte of every $M^i \cdot x, \forall i \in [1, n)$ requires more cycles and more working memory.

During LILLIPUT-TBC's encryption process, either the full tweakey schedule can be run to pre-compute all r subtweakeys before processing the plaintext, or each subtweakey can be computed on the fly to save extscram. Since the decryption process uses subtweakeys in the reverse order, the full tweakey schedule *must* be run before processing the plaintext, in order to compute the last subtweakey. The implementer can then either

- discard all intermediate subtweakeys, and re-compute RTK^{i-1} from RTK^i on the fly; this requires additional code to implemented the inverted tweakey multiplications, and adds some latency to the decryption process,
- keep all intermediate subtweakeys, which requires more extscram.

Note that there is no such issue with LILLIPUT-II, since Algorithm 4 does not need LILLIPUT-TBC's decryption process.

16-bit and 32-bit Platforms

Typical implementations of the LILLIPUT-TBC tweakey schedule and `OneRoundEGFN` function map each lane and (left and right) parts of Feistel network to a CPU word, resulting in the state of Lilliput-AE represented in 6 to 9 words of 64 bits each depending on the number of lanes.

Specifically, the implementation of LILLIPUT-AE on a 64-bit CPU can exploit 64-bit wide boolean operations and 64-bit rotations. Thus, the choice of LILLIPUT-AE favors 64-bit CPUs and yet remains efficient on 32-bit (and smaller) processors.

For implementing the tweakey schedule on a 32-bit CPU, the 64 bits of a lane should be distributed to two 32-bit words.

Implementing `OneRoundEGFN` computation leaves room for optimization on 16-bit and 32-bit processors. For 16-bit processors case, bytes should be considered and concatenated two-by-two ($x_1||x_0$, $x_3||x_2$, and so on) and then `XOR` operations can be extended to 16 bits. For 32-bit processors case, bytes should be considered and concatenated four-by-four ($x_3||x_2||x_1||x_0$, $x_7||x_6||x_5||x_4$, and so on) to get the same kind of benefits for 32-bit `XOR` operations. To ease the schedule of operations, the end of `LinearLayer` (Algorithm 7) should be computed before the beginning (Algorithm 8) to compute repetitive 16-bit or 32-bit `XOR` operations.

Algorithm 7 Linear layer - second loop

```
for $i = 0$ to $6$ do
    $x_{14-i} \leftarrow x_{14-i} \oplus x_7$
```

Algorithm 8 Linear layer - first loop

```
for $i = 0$ to $7$ do
    $x_{15} \leftarrow x_{15} \oplus x_i$
```

Performance Benchmarks Summary

In terms of memory footprint, `OneRoundEGFN` function of Lilliput-TBC can fit easily in the working memory (internal registers) of any considered processor, without requiring any additional RAM register. For example, 8-bit Atmel AVR ATmega128 processors implement 32×8-bit registers, and then, since only 16 internal registers are needed to process the entire internal state, it leaves room for 16 more available registers for intermediate computations. Concerning the tweakey schedule, since computations on each lane are executed separately, and at most 4 additional registers are needed to compute the most complex operation (α_3), RAM stack consumption is very low.

Table 4.12 compares the relative performance of a single round of each variant of the Lilliput-TBC family.

This subsection also showcases comparisons with other lightweight AEAD algorithms. We chose to compare Lilliput-AE with submissions to the Caesar [CAE] competition; in particular, we focused on the final portfolio for use-case 1, which includes Ascon [DEMS16] and Acorn [Wu16]. The features of this specific portfolio [Ber16] align with Lilliput-AE's own characteristics:

```
Use Case 1: Lightweight applications (resource constrained environments)
* critical: fits into small hardware area and/or small code for 8-bit CPUs
* desirable: natural ability to protect against side-channel attacks
* desirable: hardware performance, especially energy/bit
* desirable: speed on 8-bit CPUs
* message sizes: usually short (can be under 16 bytes), sometimes longer
```

A customized version of the Felics framework [FEL] has been developed to evaluate the code size, RAM consumption and execution time of these block ciphers on three microcontrollers:

- an 8-bit AVR ATmega128,
- a 16-bit TI MSP430,

p	Nb. of operations	**Cost wrt. 4-lane case**
4	158	1
5	191	1.21
6	234	1.48
7	270	1.71

Table 4.12: Relative cost of a single round of Lilliput-TBC $\forall p \in [4, 7]$. "Nb. of operations" is the sum of the number of operations for `OneRoundEGFN` (53, cf. Section 4.4.1) and the subtweakey computation (cf. Section 4.4.1).

- a 32-bit Arduino Due with ARM Cortex M3.

Our aim is to compare the performance of these block ciphers in a typical IoT situation: the test scenario thus consists in encrypting a single 16-byte message along with 16-byte associated data. The following compiler options were tested:

- `-O3`, to minimize computation time in order to decrease power consumption,
- `-Os`, to reduce code size and thus optimize for low memory footprint.

The FELICS framework was run on an Ubuntu 16.04 64-bit desktop with 4 3.5 GHz CPUs and 8 GB extscram. The software versions for platform-specific compilers, debuggers and other such utilities correspond to those distributed by Ubuntu, with the exception of software listed in Table 4.13.

Platform	Software	Version	Origin
AVR	simavr	`v1.6`	Developer release [Pol19]
	Avrora	`1.7.117-patched`	Cf. FELICS documentation [FEL19]
MSP	MSP430-GCC	`7.3.2.154`	Texas Instruments [MSP19]
	MSPDebug	`v0.25`	Developer release [Bee19]
ARM	J-Link Software	`V6.42f`	SEGGER [J-L19]

Table 4.13: Software versions for the FELICS framework.

The source code for the CAESAR algorithms was adapted from the SUPERCOP [SUP] toolkit in order to comply with FELICS's requirements. This implied, among other things:

- replacing platform-specific integer types with the exact-width types defined in `stdint.h`,
- isolating encryption code, decryption code, as well as constants held in read-only memory, into distinct files,
- specifying where buffers should be stored (program memory or extscram) and how they should be aligned, using FELICS-specific macro annotations.

In order to provide a fair assessment of each algorithm's performance, we looked for implementations of ASCON and ACORN distributed with SUPERCOP that performed well (*i.e.* better than the reference version) for each FELICS platform. Table 4.14 sums up which implementations were considered for each platform.

As for LILLIPUT-AE, we used the same implementation on all platforms. This implementation, called `felicsref`, differs from the reference implementation in the following ways:

- in the tweakey schedule, a loop over an array of function pointers has been unrolled manually; this was found to improve performance along every metric,
- instead of pre-computing all r round-tweakeys up-front, each round-tweakey is computed on-the-fly before applying `OneRoundEGFN`; this saves on RAM since instead of storing all r round-tweakeys in memory for the whole encryption process, only a single round-tweakey is stored.

Algorithm	**Platform**	**Implementations**
Ascon	AVR	`ref`
	MSP	`ref`
	ARM	`ref`, `opt32`
	PC	`ref`, `opt64`
Acorn	AVR	`8bitfast`
	MSP	`8bitfast`
	ARM	`opt1`
	PC	`opt1`

Table 4.14: Algorithm implementations for each platform

	Version	CFLAGS	**Code size (B)**	**RAM (B)**	**Execution time (cycles)**
Acorn-128	`8bitfast`	`-O3`	3700	263	287991
Ascon-128	`ref`	`-O3`	6140	268	191049
Ascon-128a	`ref`	`-O3`	6832	300	163315
Lilliput-I-128	`felicsref`	`-O3`	6100	266	129093
Lilliput-II-128	`felicsref`	`-O3`	6062	243	132650
Acorn-128	`8bitfast`	`-Os`	2850	240	335934
Ascon-128	`ref`	`-Os`	4322	323	254913
Ascon-128a	`ref`	`-Os`	4340	339	216080
Lilliput-I-128	`felicsref`	`-Os`	2780	261	250657
Lilliput-II-128	`felicsref`	`-Os`	2768	229	297992

Table 4.15: Performance results for 128-bit key algorithms on AVR ATmega128.

	Version	CFLAGS	**Code size (B)**	**RAM (B)**	**Execution time (cycles)**
Acorn-128	`8bitfast`	`-O3`	3276	274	391983
Ascon-128	`ref`	`-O3`	8358	290	544075
Ascon-128a	`ref`	`-O3`	8620	306	457998
Lilliput-I-128	`felicsref`	`-O3`	5760	300	121646
Lilliput-II-128	`felicsref`	`-O3`	4932	272	144399
Acorn-128	`8bitfast`	`-Os`	2326	218	381698
Ascon-128	`ref`	`-Os`	3686	372	567110
Ascon-128a	`ref`	`-Os`	3672	382	475176
Lilliput-I-128	`felicsref`	`-Os`	2304	260	201075
Lilliput-II-128	`felicsref`	`-Os`	2234	232	246193

Table 4.16: Performance results for 128-bit key algorithms on MSP430F1611.

Table 4.15, Table 4.16, Table 4.17 and Table 4.18 give our results for all 128-key algorithms on ATmega128, MSP430, ARM and desktop PC respectively. These results showcase performance for the full encryption process, including key (or tweakey) schedule.

Finally, Table 4.19, Table 4.20, Table 4.21 and Table 4.22 show the performance of the `felicsref` version of each member of the Lilliput-AE family.

	Version	CFLAGS	Code size (B)	RAM (B)	Execution time (cycles)
ACORN-128	`opt1`	`-O3`	7608	808	56288
ASCON-128	`opt32`	`-O3`	18912	268	12791
ASCON-128	`ref`	`-O3`	4080	600	32363
ASCON-128A	`opt32`	`-O3`	23764	272	11719
ASCON-128A	`ref`	`-O3`	4424	608	27688
LILLIPUT-I-128	`felicsref`	`-O3`	4656	444	86293
LILLIPUT-II-128	`felicsref`	`-O3`	4684	420	89390
ACORN-128	`opt1`	`-Os`	2364	344	44902
ASCON-128	`opt32`	`-Os`	16072	240	10221
ASCON-128	`ref`	`-Os`	1426	472	51036
ASCON-128A	`opt32`	`-Os`	18996	256	9298
ASCON-128A	`ref`	`-Os`	1408	480	42114
LILLIPUT-I-128	`felicsref`	`-Os`	1746	304	185796
LILLIPUT-II-128	`felicsref`	`-Os`	1768	272	266409

Table 4.17: Performance results for 128-bit key algorithms on ARM Cortex-M3.

	Version	CFLAGS	Code size (B)	RAM (B)	Execution time (cycles)
ACORN-128	`opt1`	`-O3`	6122	448	2966
ASCON-128	`opt64`	`-O3`	9616	192	1221
ASCON-128	`ref`	`-O3`	2236	1984	6641
ASCON-128A	`opt64`	`-O3`	11562	200	1111
ASCON-128A	`ref`	`-O3`	2102	1984	6308
LILLIPUT-I-128	`felicsref`	`-O3`	6880	528	10030
LILLIPUT-II-128	`felicsref`	`-O3`	6783	528	11816
ACORN-128	`opt1`	`-Os`	2564	392	3701
ASCON-128	`opt64`	`-Os`	9074	184	1257
ASCON-128	`ref`	`-Os`	1486	448	3718
ASCON-128A	`opt64`	`-Os`	10430	180	1164
ASCON-128A	`ref`	`-Os`	1466	448	3598
LILLIPUT-I-128	`felicsref`	`-Os`	2906	416	21345
LILLIPUT-II-128	`felicsref`	`-Os`	2867	400	24864

Table 4.18: Performance results for 128-bit key algorithms on PC.

4.4.2 Hardware Implementations

Theoretical Results on ASIC

In this section, we provide theoretical hardware implementation results on ASIC (Application-Specific Integrated Circuit) in terms of GEs. One GE is the area of a 2-input `NAND` gate in the considered CMOS technology. It allows to get normalized area and then ease comparisons between different implementations that use the same CMOS technology.

We provide here the global logic gates count for each lanes case, and translate it to the total number of GEs in a given CMOS technology. That respectively allows the reader to easily get estimations for other CMOS technologies and get real implementation numbers. The CMOS technology used here is UMCL18G212T3 (CMOS 180 nm technology). In this technology, area

	CFLAGS	**Code size (B)**	RAM **(B)**	**Execution time (cycles)**
Lilliput-I-128	-O3	6100	266	129093
Lilliput-I-192	-O3	6190	282	161775
Lilliput-I-256	-O3	6322	298	211347
Lilliput-II-128	-O3	6062	243	132650
Lilliput-II-192	-O3	6004	260	192982
Lilliput-II-256	-O3	6088	276	251050
Lilliput-I-128	-Os	2780	261	250657
Lilliput-I-192	-Os	2926	277	314893
Lilliput-I-256	-Os	3112	293	408907
Lilliput-II-128	-Os	2768	229	297992
Lilliput-II-192	-Os	2880	245	376124
Lilliput-II-256	-Os	3024	261	490172

Table 4.19: Performance of Lilliput-AE on AVR ATmega128.

	CFLAGS	**Code size (B)**	RAM **(B)**	**Execution time (cycles)**
Lilliput-I-128	-O3	5760	300	121646
Lilliput-I-192	-O3	5950	316	155267
Lilliput-I-256	-O3	6186	334	207617
Lilliput-II-128	-O3	4932	272	144399
Lilliput-II-192	-O3	5082	288	181715
Lilliput-II-256	-O3	5272	304	240655
Lilliput-I-128	-Os	2304	260	201075
Lilliput-I-192	-Os	2524	288	354297
Lilliput-I-256	-Os	2696	304	465474
Lilliput-II-128	-Os	2234	232	246193
Lilliput-II-192	-Os	2320	248	300677
Lilliput-II-256	-Os	2540	276	550781

Table 4.20: Performance of Lilliput-AE on MSP430F1611.

of respectively `XOR`, `NOT`, `AND` gates, and flip-flops are 2.67, 0.67, 1.33 and 5.33 GEs. We use non-scan flip-flops for registers in this estimation. Moreover, control logic (*e.g.*, multiplexers, finite state machine) is not taken into account, which can underestimate in the end the real practical results after Place-and-Route process. We also give a relative performance metric, which gives an estimation of the percentage of circuit area increase (considering the total number of GEs) for each lanes case, with the 4-lane case considered as a reference. We can estimate that one Lilliput-TBC S-box is equivalent to the total size of 12 `AND`, 26 `XOR` and 1 `NOT` gates, and so: $12 \times 1.33 + 26 \times 2.67 + 1 \times 0.67 = 15.96 + 69.42 + 0.67 \approx 86$GEs.

Nb. Lanes	**Registers**	**Round Function**	**Tweakey Schedule**	**Total**	**Relative Perf.**
4	384	8 S-boxes + 29×8 `XOR`s	440 `XOR`s	4530 GEs	1
5	448	8 S-boxes + 29×8 `XOR`s	507 `XOR`s	5050 GEs	1.15
6	512	8 S-boxes + 29×8 `XOR`s	577 `XOR`s	5626 GEs	1.24
7	576	8 S-boxes + 29×8 `XOR`s	650 `XOR`s	6115 GEs	1.35

	`CFLAGS`	**Code size (B)**	**RAM (B)**	**Execution time (cycles)**
Lilliput-I-128	`-O3`	4656	444	86293
Lilliput-I-192	`-O3`	4756	460	107526
Lilliput-I-256	`-O3`	4876	476	140480
Lilliput-II-128	`-O3`	4684	420	89390
Lilliput-II-192	`-O3`	4592	436	129354
Lilliput-II-256	`-O3`	4692	452	167175
Lilliput-I-128	`-Os`	1746	304	185796
Lilliput-I-192	`-Os`	1830	320	273022
Lilliput-I-256	`-Os`	1930	336	378989
Lilliput-II-128	`-Os`	1768	272	266409
Lilliput-II-192	`-Os`	1836	288	293416
Lilliput-II-256	`-Os`	1920	304	413374

Table 4.21: Performance of Lilliput-AE on arm Cortex-M3.

	`CFLAGS`	**Code size (B)**	**RAM (B)**	**Execution time (cycles)**
Lilliput-I-128	`-O3`	6880	528	10030
Lilliput-I-192	`-O3`	7073	552	12658
Lilliput-I-256	`-O3`	7295	560	16476
Lilliput-II-128	`-O3`	6783	528	11816
Lilliput-II-192	`-O3`	6946	536	14888
Lilliput-II-256	`-O3`	7139	560	19527
Lilliput-I-128	`-Os`	2906	416	21345
Lilliput-I-192	`-Os`	3049	440	31047
Lilliput-I-256	`-Os`	3210	464	37369
Lilliput-II-128	`-Os`	2867	400	24864
Lilliput-II-192	`-Os`	2979	424	30962
Lilliput-II-256	`-Os`	3113	448	44951

Table 4.22: Performance of Lilliput-AE on PC.

We can compare these results with other hardware implementations of cryptographic standards. One of the most compact implementations of Aes is the “Atomic v2” version [BBR16]: it is very lightweight and smaller than our Lilliput-TBC hardware implementations (only 2060 GEs) but processes data with a big latency (246 cycles) and then a low throughput (88.4 Mbps). One of the most compact implementation of Sha-3 (with 1088-bit block size) occupies 5522 GEs (which is bigger than the 128-bit key versions of Lilliput-TBC), and provides a very poor throughput (44.3 kbits) [KY10].

An argument against tunable parameters in a standard is that it makes implementations more expensive, as they usually have to support all parameter values to fully implement the standard. However, for Lilliput-AE, this can be mitigated by only implementing the hardware needed for computing the M and M_R functions, and iterate on them to compute the needed remaining multiplications. This version will allow to save some logic gates, but at the expense of a decreased throughput.

For the fpga implementation particular case, the S-box S can be put in dedicated block extscrams of the used fpga.

The high parallelization level of the nonce-respecting and the nonce-misuse resistant modes allows implementing in hardware many instances of E_K running in parallel and then getting high throughput, especially on dedicated ASICs.

VHDL Results

This subsection showcases performance results for iterated versions of all variants of the Lilliput-TBC tweakable block cipher. These results were produced using version 14.4 of the ISE Design Suite on a Virtex-6 XC6VLX75T device, with two optimization settings: "area reduction" and "timing performance".

Table 4.23 and Table 4.24 provide results of LilliputTBC with implementations optimized for circuit area and execution time, respectively.

Finally, Table 4.25 and Table 4.26 compare Lilliput-TBC to Ascon-128a, Ascon-128 and Aes when optimized for circuit area and execution time, respectively. We used the iterated implementation of Ascon-128a and Ascon-128 described in [Fiv16].

Table 4.23: Results for Lilliput-TBC, optimized for area reduction.

Lilliput-TBC	I-128	I-192	I-256	II-128	II-192	II-256
LUTs	1345	1605	1827	854	1055	1175
slices	384	428	534	249	302	344
registers	1076	1204	1336	943	1075	1203
flip-flop pairs	1345	1605	1827	886	1097	1223
unused flip-flops	464	538	634	233	258	264
unused LUTs	0	0	0	32	42	48
fully used	881	1067	1193	621	797	911
Freq (MHz)	282	286	288	338	326	339

Table 4.24: Results for Lilliput-TBC, optimized for timing performance.

Lilliput-TBC	I-128	I-192	I-256	II-128	II-192	II-256
LUTs	1625	1894	2140	1033	1218	1336
slices	568	564	649	430	433	415
registers	1109	1237	1369	1097	1108	1236
flip-flop pairs	1625	1894	2140	1136	1277	1405
unused flip-flops	700	777	896	315	370	375
unused LUTs	0	0	0	103	59	69
fully used	925	1117	1244	718	848	961
Freq (MHz)	357	352	367	402	388	408

4.4.3 Threshold Implementations

This section aims at giving the reader some insight into first order TIs of Lilliput-AE.

The S-box

The quadratic functions As stated in Section 4.3.2, the 8-bit S-box has been chosen with TIs in mind as it is built from three inner 4-bit S-boxes, each directly decomposable into quadratic

Table 4.25: Comparison of LILLIPUT-TBC, ASCON and AES implementations, optimized for area reduction.

	ASCON-128	ASCON-128A	TBC-I-128	TBC-II-128	AES
LUTs	1318	1422	1345	854	1615
slices	357	387	384	249	437
registers	933	997	1076	943	661
Freq (MHz)	372	357	288	338	170
Throughput(Mbit/sec)	3402	5084	1152	1352	2181

Table 4.26: Comparison of LILLIPUT-TBC, ASCON and AES implementations, optimized for timing performance.

	ASCON-128	ASCON-128A	TBC-I-128	TBC-II-128	AES
LUTs	1370	1754	1625	1033	3258
slices	392	497	568	430	897
registers	933	997	1109	1097	670
Freq (MHz)	432	460	357	402	175
Throughput(Mbit/sec)	3951	6544	1428	1608	2245

permutations. Therefore, a first order TI can be achieved using only three shares. The following algorithms describe, for each quadratic permutation F, G and Q, a function f that computes an output share $\langle x, y, z, t\rangle$ for two input shares $\langle a_0, b_0, c_0, d_0\rangle$ and $\langle a_1, b_1, c_1, d_1\rangle$.

Algorithm 9 $f_F(\langle a_0, b_0, c_0, d_0\rangle, \langle a_1, b_1, c_1, d_1\rangle) = \langle x, y, z, t\rangle$

```
$x \leftarrow (a_0 \oplus c_0)(b_0 \oplus d_0) \oplus (a_0 \oplus c_0)(b_1 \oplus d_1) \oplus (a_1 \oplus c_1)(b_0 \oplus d_0)$
$y \leftarrow a_0 d_0 \oplus a_0 d_1 \oplus a_1 d_0$
$z \leftarrow b_1 \oplus d_1$
$t \leftarrow (a_0 \oplus b_0 \oplus d_0)(a_0 \oplus b_0 \oplus c_0) \oplus (a_0 \oplus b_0 \oplus d_0)(a_1 \oplus b_1 \oplus c_1) \oplus (a_1 \oplus b_1 \oplus d_1)(a_0 \oplus b_0 \oplus c_0)$
```

Algorithm 10 $f_G(\langle a_0, b_0, c_0, d_0\rangle, \langle a_1, b_1, c_1, d_1\rangle) = \langle x, y, z, t\rangle$

```
$x \leftarrow a_1$
$y \leftarrow b_1$
$z \leftarrow c_1$
$t \leftarrow b_0 c_0 \oplus b_0 c_1 \oplus b_1 c_0 \oplus d_1$
```

Algorithm 11 $f_Q(\langle a_0, b_0, c_0, d_0\rangle, \langle a_1, b_1, c_1, d_1\rangle) = \langle x, y, z, t\rangle$

```
$x \leftarrow c_0 d_0 \oplus c_0 d_1 \oplus c_1 d_0 \oplus b_1$
$y \leftarrow d_1$
$z \leftarrow a_0 d_0 \oplus a_0 d_1 \oplus a_1 d_0 \oplus c_1$
$t \leftarrow a_1$
```

Therefore, for each quadratic function $A \in F, G, Q$, TI with three shares is achived by computing

$$A(\langle a_0, b_0, c_0, d_0\rangle, \langle a_1, b_1, c_1, d_1\rangle, \langle a_2, b_2, c_2, d_2\rangle) = \begin{aligned}&f_A(\langle a_1, b_1, c_1, d_1\rangle, \langle a_2, b_2, c_2, d_2\rangle),\\ &f_A(\langle a_2, b_2, c_2, d_2\rangle, \langle a_0, b_0, c_0, d_0\rangle),\\ &f_A(\langle a_0, b_0, c_0, d_0\rangle, \langle a_1, b_1, c_1, d_1\rangle)\,.\end{aligned} \tag{4.9}$$

Contrary to Q and G, the output sharing of F is not uniform but it does not matter as these functions are used in a Feistel network. Therefore, there is no need for re-masking and a threshold implementation of the 8-bit S-box can be built upon the algorithms described above. Note that the inner 4-bit S-box S_4^3 requires an additionnal `NOT` instruction: it only has to be applied to one of the three shares (*i.e.*, $\neg x = \neg x_0 \oplus x_1 \oplus x_2$).

Software implementation using Look-Up Tables In order to improve the performance of software implementations, it is possible to use look-up tables for the quadratic functions as done in [SBM18]. To do so, one can compute three 8-bit to 4-bit look-up tables from f_F, f_G and f_Q noted T_F, T_G and T_Q, respectively. Because $\bar{S}_4^2$ requires a bitwise permutation $P =$ `028a46ce139b57df` between the two quadratics, an additional 4-bit to 4-bit look-up table can be used.

However, as a_0 and d_0 do not interfere in the computation of $f_G(\langle a_0, b_0, c_0, d_0\rangle, \langle a_1, b_1, c_1, d_1\rangle)$, it is possible to divide the size of T_G by four (*i.e.*, from 256 to 64 bytes) at the cost of two bitwise operations at each table look-up. In the same way, b_0 does not interfere in the computation of $f_Q(\langle a_0, b_0, c_0, d_0\rangle, \langle a_1, b_1, c_1, d_1\rangle)$ and the size of T_Q can be reduced by half. In the rest of this section, we use these tricks in order to minimize the memory space required to store the look-up tables. The three resulting look-up tables are given below.

	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
0	0	2	0	2	2	0	2	0	0	2	0	2	2	0	2	0
1	0	2	9	b	3	1	a	8	d	f	4	6	e	c	7	5
2	0	b	0	b	b	0	b	0	1	a	1	a	a	1	a	1
3	9	2	0	b	3	8	a	1	5	e	c	7	f	4	6	d
4	1	2	8	b	3	0	a	9	9	a	0	3	b	8	2	1
5	0	3	0	3	3	0	3	0	5	6	5	6	6	5	6	5
6	8	2	1	b	3	9	a	0	1	b	8	2	a	0	3	9
7	0	a	0	a	a	0	a	0	4	e	4	e	e	4	e	4
8	1	e	0	f	b	4	a	5	1	e	0	f	b	4	a	5
9	c	3	4	b	7	8	f	0	1	e	9	6	a	5	2	d
a	0	6	1	7	3	5	2	4	1	7	0	6	2	4	3	5
b	4	2	c	a	6	0	e	8	8	e	0	6	a	c	2	4
c	8	6	0	e	2	c	a	4	0	e	8	6	a	4	2	c
d	4	a	5	b	f	1	e	0	1	f	0	e	a	4	b	5
e	0	7	8	f	3	4	b	c	9	e	1	6	a	d	2	5
f	5	2	4	3	7	0	6	1	1	6	0	7	3	4	2	5

Table 4.27: $T_F[x][y] = f_F(x, y)$

In this way, the memory space required to store all the look-up tables equals $|T_F| + |T_G| + |T_Q| + |P| = 256 + 64 + 128 + 16 = 464$ bytes. Finally, the output shares of the 8-bit S-box can be computed by running the Feistel network step by step as detailed by Section 4.4.3.

	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
0	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
1	0	1	2	3	5	4	7	6	8	9	a	b	d	c	f	e
2	0	1	3	2	4	5	7	6	8	9	b	a	c	d	f	e
3	1	0	2	3	4	5	7	6	9	8	a	b	c	d	f	e

Table 4.28: $T_G[x][y] = f_G(x \ll 1, y)$

	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
0	0	4	2	6	8	c	a	e	1	5	3	7	9	d	b	f
1	0	4	a	e	8	c	2	6	3	7	9	d	b	f	1	5
2	0	c	2	e	8	4	a	6	1	d	3	f	9	5	b	7
3	8	4	2	e	0	c	a	6	b	7	1	d	3	f	9	5
4	0	6	2	4	8	e	a	c	1	7	3	5	9	f	b	d
5	2	4	8	e	a	c	0	6	1	7	b	d	9	f	3	5
6	0	e	2	c	8	6	a	4	1	f	3	d	9	7	b	5
7	a	4	0	e	2	c	8	6	9	7	3	d	1	f	b	5

Table 4.29: $T_Q[x][y] = f_Q(x + 4, y)$

Application to the Entire Algorithm

The tweakey schedule Because the key is manipulated along with the tweak during the tweakey schedule, this step must be protected to prevent a side-channel attack. To do so, one can share the tweak and the key into two shares. There is no difficulty to apply TI to the tweakey schedule as it operates in a linear fashion. As a result, the tweakey schedule produces subtweakeys splitted in two shares RTK_0^i and RTK_1^i. In order to limit the amount of randomness to generate, it is possible to share the key only. However, note that non-sharing the tweak implies that a profiling attack against the tweakey schedule would allow to deduce some information on the power consumption model of the device.

The EGFN round function A way of applying TI to the round function is to share the input block into three shares which are processed during the entire round function. More precisely, if $X_{i,j}$ refers to the i^{th} byte of the j^{th} share of X, then a TI of F_i at round r consists in $F_i' = S'(X_{i,0} \oplus RTK_{i,0}^r,\ X_{i,1} \oplus RTK_{i,1}^r,\ X_{i,2})$ where S' refers to the Section 4.4.3. Because the remaining steps of the round function are linear, it is sufficient to apply it on each share independently.

Performance Impact

We implemented the thresholding scheme described in this section, using lookup tables for the S-box, and compared its performance with our `felicsref` implementation, in the conditions described in Section 4.4.1 with the compiler option `-O3`. Table 4.30 shows the impact for each metric on each platform.

Note that the `threshold` implementation used in this benchmark does not include a random number generator; these results therefore do not account for the overhead induced by share initialization.

Algorithm 12 $S'(s_0, s_1, s_2) = s'_0, s'_1, s'_2$ with look-up tables T_F, T_G, T_Q and P

```
/* Decompose 8-bit shares into 4-bit shares */ for $i = 0$ to $2$ do
    $\bar{s}_i \leftarrow s_i \gg 4$
    $\underline{s}_i \leftarrow \texttt{AND}(s_i, 15)$
end

/* First 4-bit S-box */ $t_0 \leftarrow T_G[\texttt{AND}(\underline{s}_1, 7) \gg 1][\underline{s}_2]$
$t_1 \leftarrow T_G[\texttt{AND}(\underline{s}_2, 7) \gg 1][\underline{s}_0]$
$t_2 \leftarrow T_G[\texttt{AND}(\underline{s}_0, 7) \gg 1][\underline{s}_1]$
$\bar{s}_0 \leftarrow \bar{s}_0 \oplus T_F[t_1][t_2]$
$\bar{s}_1 \leftarrow \bar{s}_1 \oplus T_F[t_2][t_0]$
$\bar{s}_2 \leftarrow \bar{s}_2 \oplus T_F[t_0][t_1]$

/* Second 4-bit S-box */ $t_0 \leftarrow P\,[T_Q[\texttt{AND}(\bar{s}_1, 3) \oplus (\texttt{AND}(\bar{s}_1, 8) \gg 1)][\bar{s}_2]]$
$t_1 \leftarrow P\,[T_Q[\texttt{AND}(\bar{s}_2, 3) \oplus (\texttt{AND}(\bar{s}_2, 8) \gg 1)][\bar{s}_0]]$
$t_2 \leftarrow P\,[T_Q[\texttt{AND}(\bar{s}_0, 3) \oplus (\texttt{AND}(\bar{s}_9, 8) \gg 1)][\bar{s}_1]]$
$\underline{s}_0 \leftarrow \underline{s}_0 \oplus T_Q[\texttt{AND}(t_1, 3) \oplus (\texttt{AND}(t_1, 8) \gg 1)][t_2]$
$\underline{s}_1 \leftarrow \underline{s}_1 \oplus T_Q[\texttt{AND}(t_2, 3) \oplus (\texttt{AND}(t_2, 8) \gg 1)][t_0]$
$\underline{s}_2 \leftarrow \underline{s}_2 \oplus T_Q[\texttt{AND}(t_0, 3) \oplus (\texttt{AND}(t_0, 8) \gg 1)][t_1]$

/* Third 4-bit S-box */ $t_0 \leftarrow T_G[\texttt{AND}(\underline{s}_1, 7) \gg 1][\underline{s}_2] \oplus 1$
$t_1 \leftarrow T_G[\texttt{AND}(\underline{s}_2, 7) \gg 1][\underline{s}_0]$
$t_2 \leftarrow T_G[\texttt{AND}(\underline{s}_0, 7) \gg 1][\underline{s}_1]$
$\bar{s}_0 \leftarrow \bar{s}_0 \oplus T_F[t_1][t_2]$
$\bar{s}_1 \leftarrow \bar{s}_1 \oplus T_F[t_2][t_0]$
$\bar{s}_2 \leftarrow \bar{s}_2 \oplus T_F[t_0][t_1]$

/* Build 8-bit output shares from 4-bit shares */ for $i = 0$ to $2$ do
    $s'_i \leftarrow (\bar{s}_i \ll 4) \oplus \underline{s}_i$
end
```

4.5 External Cryptanalysis of Lilliput-AE

4.5.1 Lilliput-AE v1

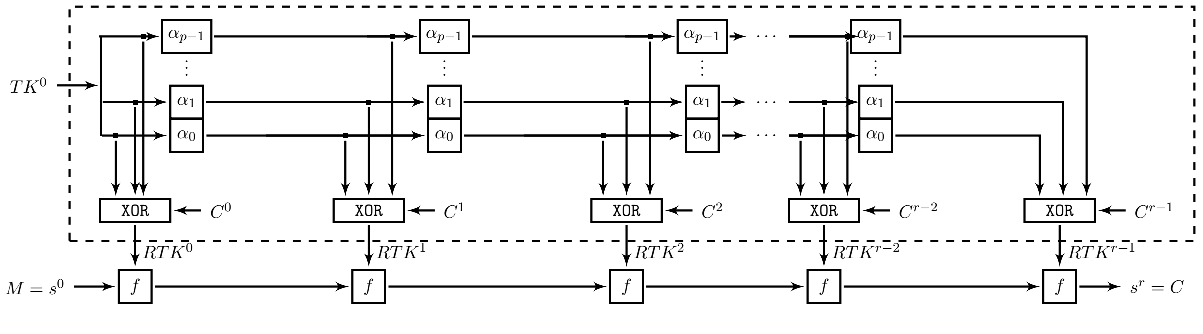


Figure 4.14: The tweakey schedule of Lilliput-AE.

Platform	**Member**	$\frac{\text{ROM}_{threshold}}{\text{ROM}_{felicsref}}$	$\frac{extscram_{threshold}}{extscram_{felicsref}}$	$\frac{cycles_{threshold}}{cycles_{felicsref}}$
AVR	LILLIPUT-I-128	2.37	1.60	5.17
	LILLIPUT-I-192	2.39	1.59	4.87
	LILLIPUT-I-256	2.43	1.59	4.57
	LILLIPUT-II-128	2.39	1.63	6.41
	LILLIPUT-II-192	2.42	1.61	5.34
	LILLIPUT-II-256	2.46	1.61	5.02
MSP	LILLIPUT-I-128	1.85	1.51	4.39
	LILLIPUT-I-192	1.85	1.51	4.12
	LILLIPUT-I-256	1.87	1.50	3.87
	LILLIPUT-II-128	2.01	1.54	4.85
	LILLIPUT-II-192	2.00	1.53	4.57
	LILLIPUT-II-256	2.02	1.53	4.29
ARM	LILLIPUT-I-128	1.99	1.51	4.36
	LILLIPUT-I-192	1.98	1.51	4.19
	LILLIPUT-I-256	1.99	1.51	3.95
	LILLIPUT-II-128	1.99	1.52	5.53
	LILLIPUT-II-192	2.01	1.52	4.55
	LILLIPUT-II-256	2.02	1.52	4.34
PC	LILLIPUT-I-128	1.49	1.29	4.54
	LILLIPUT-I-192	1.49	1.28	4.38
	LILLIPUT-I-256	1.50	1.30	4.13
	LILLIPUT-II-128	1.50	1.26	5.17
	LILLIPUT-II-192	1.50	1.28	4.84
	LILLIPUT-II-256	1.51	1.27	4.51

Table 4.30: Performance impact of the thresholding scheme.

In the first version of LILLIPUT-AE that was submitted to NIST [ABC+18], the tweakey schedule was slightly different. More precisely, while the overall structure and type of operations were identical, other α coefficients were originally used. In particular, we chose $\alpha_0 = I$, $\alpha_1 = M$, $\alpha_2 = M^2$, $\alpha_3 = M^3$, $\alpha_4 = M_R$, $\alpha_5 = M_R^2$ and $\alpha_6 = M_R^3$, where I is the 64×64 identity matrix, M and M_R are the same matrices as described in Section 4.2.3. These choices ensured that in r consecutive rounds, at most $(p-1)$ cancellations occurred in the tweakey schedule while allowing for a very efficient processing of the tweak at the same time[6].

Unfortunately, this simple design introduced a fundamental flaw that was overlooked by our team: the same part of the tweakey material was reused at each round. In 2019, Dunkelman, Keller, Lambooij and Sasaki exploited this weakness and found a related tweakey differential characteristic with probability 1 that was then used to mount practical forgery attacks on LILLIPUT-AE [DKLS19]. In the following, we briefly describe their attack and discuss its impact on our design.

[6]We recall that M defines a word-ring-LFSR with minimum number of XOR gates and a primitive polynomial of degree 64. M_R is defined by the reciprocal polynomial of M.

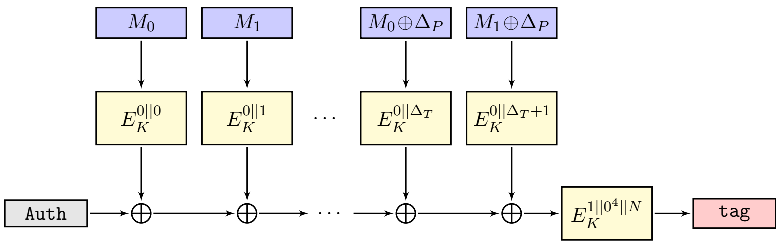


Figure 4.15: Tag collision using a 1-round iterative related tweakey differential.

4.5.2 External analysis of LILLIPUT-AE

Related-tweak differential characteristic for LILLIPUT-TBC Dunkelman *et al.* made the observation that the first lane of the tweakey schedule—the least significant 64-bit word of the tweak—was never updated between each round. Consequently, when considering two tweaks that differed by $\Delta_T = (\Delta^7, \ldots, \Delta^0)$ in the least significant word, all corresponding round subkeys had the same difference Δ. By setting the difference in the right-hand side of the state to Δ_T, they were able to cancel out the input difference to all S-boxes, and thus, to pass the non-linear layer with probability 1. Then, by ensuring that $\Delta_T^7 = 0$ and $\Delta_T^1 \oplus \cdots \oplus \Delta_T^6 = 0$ and by exploiting the cycle decomposition of the permutation π, the authors were able to identify several configurations allowing for a 1-round iterative related-tweak differential characteristic with probability 1.

Forgery attacks on the LILLIPUT-AE scheme The authors then found that the characteristic

$$\Delta_P = (0,0,0,0,0,01_{(8)},0,01_{(8)}||0,0,0,01_{(8)},01_{(8)},0,0,0), \Delta_T = (0,0,0,01_{(8)},01_{(8)},0,0,0),$$

where Δ_P and Δ_T denote the difference between the plaintexts and the tweaks, respectively, could be used to produce a pair of messages with the same tag, as shown in Section 4.5.2. Indeed, let ℓ be equal to $\Delta_T + 2$, where Δ_T is seen as an integer[7], then choosing a message M of length at least ℓ blocks such that $M_{\ell-2} = M_0 \oplus \Delta_P$ and $M_{\ell-1} = M_1 \oplus \Delta_P$ yields:

$$E_K^{0||0}(M_0) \oplus E_K^{0||\Delta_T}(M_{\Delta_T}) = E_K^{0||0}(M_1) \oplus E_K^{0||\Delta_T}(M_{\Delta_T+1}) = \Delta_P.$$

As a consequence, the difference before the $E_K^{1||0^4||N}$ operation equals zero, which leads to a collision on the tag if the same nonce is reused. In their paper, the authors also present a known message variant, requiring a message of length at least $2^{32} + 2^{24} + 1$ blocks of message and that can also be applied in the nonce respecting mode.

4.5.3 Impact

As stated by the authors of [DKLS19], the weakness exploited in the analysis comes from the fact that part of the tweak does not get updated[8]. This resulted in an unfortunate interaction between the mode and a probability-1 differentials, which highlights the potential security risk in using a tweakey schedule design that reuses the same part in every round. Authors suggested replacing the identity transformation α_0 used in the tweakey schedule with some mixing linear transformation to avoid having the same difference in all round keys, therefore thwarting their

[7]If $\Delta_T = (0,0,0,01_x,01_x,0,0,0)$, then $\ell = 2^{32} + 2^{24} + 2$.

[8]Therefore, the security of the LILLIPUT block cipher is not compromised.

attack. This led to an updated design of our proposal—called Lilliput v1.1—to ensure that no lane goes through the tweakey schedule unmodified:

- α_0 was changed from I to M,
- α_1, from M to M^2,
- α_2, from M^2 to M^3,
- α_3, from M^3 to M^4.

4.6 Conclusion

Lilliput-AE is a lightweight aead scheme that is based on a egfn construction to achieve high diffusion. Its nonlinear components have been chosen to optimize their cost in threshold implementations, thus facilitating side-channel protection while ensuring attractive cryptographic properties. Lilliput-AE is suited to be implemented efficiently on a wide range of processors and in hardware. Its performances are comparable to those of other lightweight algorithms, such as Acorn and Ascon, which are part of the final portfolio of the Caesar competition [CAE]. Future works could include optimized software implementations of Lilliput-AE on IoT platforms, side-channel protected implementations with performance benchmark, or optimized hardware implementations (*e.g.*, serial implementations).

From the cryptanalysis aspect, Lilliput-AE will most likely receive less attention from the community, since it was not selected for Round 2 of the nist standardization process. Still, it would be interesting to see what cryptanalysts come up with to attack the tweaked version that has been presented in this chapter.

5

Cryptanalysis Results on Spook

But it was my integrity that was important. Is that so selfish? It sells for so little, but it's all we have left in this place.

Valerie Page

Spook [BBB+19] is one of the 32 candidates that has made it to the second round of the NIST Lightweight Cryptography Standardization process, and is particularly interesting since it proposes differential side channel resistance. This chapter introduces a joint work with Patrick Derbez, Virginie Lallemand, María Naya-Plasencia, Léo Perrin and André Schrottenloher [DHL+20] in which practical distinguishers of the full 6-step version of the underlying permutations of Spook, namely Shadow-512 and Shadow-384, are exhibited, thus solving challenges proposed by the designers on the permutation. We also discuss practical forgeries for the S1P mode of operation in the nonce misuse scenario—which is allowed by the CIML2 security game considered by the authors—using a 4-step Shadow. All the results presented in the following have been implemented.

5.1 Introduction

The number of applications running on interconnected resource-constrained devices has increased exponentially in the last decade, bringing new challenges to both the community and the industry. Sensor networks, Internet-of-Things, smart cards and healthcare are a few examples which handle sensitive data that should be protected (see Chapter 2).

These new platforms have their own specific sets of requirements, in particular in terms of implementation efficiency. As common cryptographic primitives were not designed to satisfy these specific use cases, they can be ill-suited in these contexts. A staggering number of algorithms has been proposed to fulfill such requirements, such as Present [BKL+07] (low gate count in hardware), Prince [BCG+12] (low latency in hardware), Midori [BBI+15] (low power consumption), or Lea [HLK+14] (low ROM and cycle count on micro-controllers). Such primitives have been nicknamed *lightweight*. Because the corresponding devices can often be expected to be physically interacted with by an attacker, an algorithm easing side channel resistance has a significant advantage. Hence many recent proposals were designed to be *naturally* resistant against side-channel attacks or, at least, protectable at low cost. For instance, the authenticated encryption (AE) scheme Pyjamask [GJK+19] was designed with a minimal number of non-linear gates to allow efficient masked implementations while the AE scheme ISAP [DEM+17] is resistant to differential power analysis, a powerful type of attack where the adversary try to deduce information about the secret key from power consumption.

This need for lightweight cryptographic primitives led the American National Institute of Standards and Technology (NIST) to initiate the Lightweight Cryptography Project, aiming at the standardization of hash functions and authenticated encryption algorithms suitable for constrained devices. It received 57 algorithm proposals in February 2019 and accepted 56 of them. In August 2019, 32 primitives were announced as the 2nd round candidates.

In this chapter we study `Spook`, an Authenticated Encryption scheme with Associated Data (AEAD) which is among those 2nd round candidates. It was designed to achieve both resistance against side-channel analysis and low-energy implementations and is particularly interesting as it aims at providing strong integrity guarantees even in the presence of nonce misuse and leakage. AEAD is provided using three sub-components: the *Sponge One-Pass* mode of operation (S1P), the tweakable block cipher Clyde-128 and the permutation `Shadow`. Both Clyde and `Shadow` are based on simple extensions of the LS-design framework first introduced by the designers of the lightweight block ciphers Robin and Fantomas [GLSV15]. This strategy leads to efficient bitslicing and side-channel resistant implementations on a wide range of platforms. To further simplify the implementation, the permutation uses the round function of the tweakable block cipher as a sub-routine, effectively combining 3 or 4 parallel instances of a round-reduced cipher using a simple linear layer to construct a 384- or 512-bit permutation.

Motivation and contributions. In Section 4.3 of the specification document of `Spook` [BBB+19], the designers explicitly point out that an important requirement for the permutation in the S1P mode of operation is that it provides collision resistance with respect to the 255 bits that generate the tag and they say:

> *"Hence, a more specific requirement is to prevent truncated differentials with probability larger than 2^{-128} for those 255 bits. A conservative heuristic for this purpose is to require that no differential characteristic has probability better than 2^{-385}, which happens after twelve rounds (six steps)."*

In this chapter we show that this heuristic is not conservative, providing practical truncated distinguishers on `Shadow`, the inner permutation of `Spook`. We exhibit non-random behavior for up to the full version of `Shadow-512`. Moreover, the same technique would also distinguish `Shadow-512` extended by 2 more rounds at the end. More precisely, we exhibit two particular subspaces E and F of co-dimension 128 and an efficient algorithm which returns pairs of messages (m, m') such that $m \oplus m' \in E$ and $\texttt{Shadow-512}(m) \oplus \texttt{Shadow-512}(m') \in F$. This implies in particular a practical collision on 128 bits of the output. This problem is a particular instance of the so-called *limited birthday* problem, which was first introduced by Gilbert and Peyrin when looking for known key distingushers against the Aes [GP10]. As a permutation can be seen as a block cipher with a known key, it is natural to borrow distinguishers from this field. While the complexity of a generic algorithm performing this task is around 2^{64} because of the birthday bound (see [IPS13] for more details), our un-optimized implementations of our distinguishers run in at most a few minutes on a regular desktop computer.

We also provide similar distinguishers targeting up to 10 (out of 12) rounds of `Shadow-384`, the small version of Shadow. Note that, as for `Shadow-512`, adding 2 more rounds at the end of the permutation would not increase its security as there would exist a similar distinguisher on the last 12 rounds (a 2-round shifted version of the proposed permutation).

As other several sponge-based lightweight algorithms [1], the authors purposefully relied on a permutation for which distinguishers could exist as this allows to use fewer permutation rounds (`Spook` designers pointed out for instance that 12 rounds were not enough to have 512 bits of security with respect to linear distinguishers) and thus an increase in the speed of data processing. Nevertheless, our distinguishers seem to prove that the behavior of `Shadow` is not compatible with the requirements given by the authors on the permutation for the S1P mode of operation.

The next important question is whether these distinguishers are a threat to `Spook` itself, as the impact is *a priori* not clear. For `Spook`, we are able to leverage the results we obtained to produce practical existential forgeries for the S1P mode of operation when `Shadow-512` is reduced to 8 rounds out of 12 in the nonce misuse scenario, which is allowed by the CIML2 security game considered by the authors [BPPS17].

Distinguishers on both `Shadow-512` and `Shadow-384` along with the forgeries on 4-step `Spook` have been implemented and verified against the reference implementation provided by the designers.

Chapter Organization. In Section 5.2 we describe `Shadow` and introduce some cryptanalysis techniques. Then in Section 5.3 we make some observations on the structure of the permutation that will play a crucial role in our cryptanalysis. Finally, in Sections 5.4 and 5.5 we present the results of our analysis of both versions of `Shadow`, including a distinguisher on the full `Shadow-512`, as well as forgeries against `Spook` when `Shadow-512` is reduced to 8 rounds.

All the analyses presented in this chapter are practical and have been implemented and tested. Their source code is available at:

https://who.paris.inria.fr/Leo.Perrin/code/spook/index.html

Our results have been acknowledged and discussed by the designers of `Spook` in [BBB+20].

[1]See for instance Ascon [DEMS16], Ketje [BDP+16], or Sparkle [BBdS+19]

5.2 Preliminaries

The specific mode of operation we target will be described in the relevant section. Here, we present the `Shadow` family of permutations and recall the definition of differential distinguishers.

5.2.1 Specification of `Shadow-384` and `Shadow-512`

The `Spook` algorithm is based on a permutation named `Shadow` that exists in two flavors: `Shadow-384` and `Shadow-512`, where `Shadow-512` is the one used in the primary candidate to the NIST Lightweight competition. In both cases, the internal state is seen as a collection of m two-dimensional arrays (or bundles) each of dimensions 32×4: as depicted in Figure 5.1, $m = 4$ for `Shadow-512` and $m = 3$ for `Shadow-384`. The permutations have a Substitution Permutation Network (SPN) structure based on a 4-bit S-box layer and two distinct linear layers, each being used every second round.

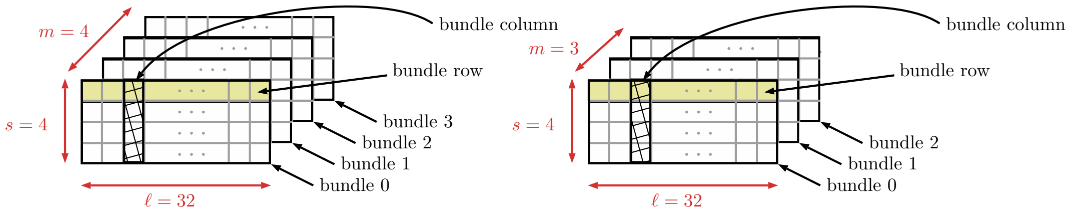


Figure 5.1: State Organization of `Shadow-512` (left) and of `Shadow-384` (right).

The full versions of the permutations iterate 6 *steps*. As represented in Figure 5.2, one step is made of two rounds, denoted round A and round B, interleaved with round constant additions. `Shadow-384` and `Shadow-512` only differ in the definition of the D layer.

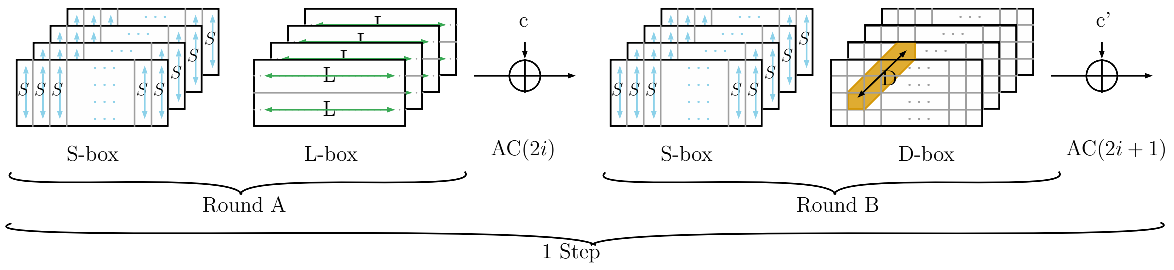


Figure 5.2: Description of one step of `Shadow-512`.

Round A first applies a non-linear layer made by the application on each bundle column of the 4-bit S-box recalled in Table 5.1. It then applies the so-called L-box which calls the L' transformation to the first two and last two rows of each bundle. If we denote by (x, y) the input and by (a, b) the output the definition of L' is given by:

$$(a, b) = L'(x, y) = \begin{pmatrix} \mathrm{circ}(\texttt{0xec045008}) \cdot x^T \oplus \mathrm{circ}(\texttt{0x36000f60}) \cdot y^T \\ \mathrm{circ}(\texttt{0x1b0007b0}) \cdot x^T \oplus \mathrm{circ}(\texttt{0xec045008}) \cdot y^T \end{pmatrix}$$

where $\mathrm{circ}(A)$ stands for a circulant matrix whose first line is a row vector given by the binary decomposition of A.

Table 5.1: 4-bit S-box used in `Shadow`.

x	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
S(x)	0	8	1	f	2	a	7	9	4	d	5	6	e	3	b	c

Round B starts with the same S-layer as round A but uses a different linear layer, denoted D. The purpose of D is to provide diffusion between the m bundles of the state: as depicted in Figure 5.2, it takes as input one bit of each bundle. It modifies them with the application of a near-MDS matrix (which previously appeared in the design of the ciphers Midori [BBI$^+$15] and Mantis [BJK$^+$16] for instance), respectively:

$$D(a,b,c,d) = \begin{pmatrix} 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 0 \end{pmatrix} \times \begin{pmatrix} a \\ b \\ c \\ d \end{pmatrix}$$

for `Shadow-512` while for `Shadow-384` we use:

$$D(a,b,c) = \begin{pmatrix} 1 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 1 & 0 \end{pmatrix} \times \begin{pmatrix} a \\ b \\ c \end{pmatrix} .$$

The **round constants** used in the permutation correspond to the internal state of a 4-bit LFSR. They are recalled in Table 5.2. At the end of every round (for rounds from 0 to 11), the 4-bit constant is XORed at 4 different positions, one time in each bundle: in bundle b (for $b = 0, 1, 2, 3$), the constant is XORed to the column number b. Without loss of generality, we hereafter position bit number 0 on the right of the state in our figures.

Table 5.2: Round constants used in `Shadow`. Note that the LSB is on the left.

Round	Constant	Round	Constant	Round	Constant	Round	Constant
0	(1,0,0,0)	1	(0,1,0,0)	2	(0,0,1,0)	3	(0,0,0,1)
4	(1,1,0,0)	5	(0,1,1,0)	6	(0,0,1,1)	7	(1,1,0,1)
8	(1,0,1,0)	9	(0,1,0,1)	10	(1,1,1,0)	11	(0,1,1,1)

5.2.2 Differential Distinguishers

As indicated in the `Spook` specification, the black box security analysis of the mode of operation that is used in `Spook` (S1P) relies on the assumption that the permutations are random. In this chapter we challenge this assumption by exhibiting distinguishers for the permutations – that is, algorithms that unveil a non-random behavior.

Our distinguishers use the notion of differential, a technique that was introduced by Biham and Shamir in [BS91a]. The idea is to find a couple of XOR differences (δ, Δ) such that if two messages differ from δ then with high probability their output difference after encryption is equal to Δ.

This idea was later extended by Knudsen in 1994 to define *truncated differentials* [Knu95], a variant in which only a portion of the difference is fixed (while the remaining part is undetermined). This technique is illustrated in Figure 5.5 for instance, where we introduce a distinguisher that ends with a difference of the form $(*, *, *, 0)$ before the last D operation: the '*' symbol indicates that the difference between the messages is not determined over the first three bundles, while the '0' symbol indicates that the two messages are identical on the last bundle (128 bits).

5.3 Structural Observations

In this section we present the general properties we found that we will later exploit in our analysis. While our distinguisher is a truncated differential one, our method for finding right pairs does not rely on a high probability differential trail (whose very existence is disproved by the authors' wide trail argument). Instead, we exploit the similarity between the functions applied in parallel on each bundle. To better describe them, we introduce the notion of Super S-box (as it applies to `Shadow`) and we study the propagation of the following type of properties through the step function. Note that we next provide the details for `Shadow-512` but that similar results apply to `Shadow-384`.

Definition 5.1 (i-identical state)**.** *We call i-identical an internal state of* `Shadow` *in which i bundles are equal.*

5.3.1 Super S-box

Given the fact that in every step only the D layer is mixing the bundles together, it is possible to rewrite `Shadow` as an SPN using four 128-bit *Super S-boxes* (each operating on one bundle) interleaved with a linear permutation D operating on the full state. If a, b, c and d are the 128-bit bundles, this linear permutation is represented as follows:

$$D(a,b,c,d) = \begin{pmatrix} 0 & I & I & I \\ I & 0 & I & I \\ I & I & 0 & I \\ I & I & I & 0 \end{pmatrix} \times \begin{pmatrix} a \\ b \\ c \\ d \end{pmatrix}.$$

D is an involution with branching number 4 (over 128-bit words) and it verifies that $\forall a \in \mathbb{F}_2^{128}, D(a,a,a,0) = (0,0,0,a)$.

We denote by σ_j for $j \in \{0,1,2,3\}$ the four parallel *Super S-boxes* of the cipher. They correspond to the first four operations of the step, namely: the S-layer and the linear operation L of round A, the constant addition (that is done on a different position for each Super S-box), and the S-layer of round B.

In the following, we show that even though the four bundles of a state go through different Super S-boxes it might be possible to have a `Shadow` state with four equal bundles that is transformed into a `Shadow` state of the same form at the output of a full step.

5.3.2 4-Identical States

In the discussion below we follow the evolution of the 4-identical property through a step and show the required conditions for it to remain in the end. This evolution is also summarized in Figure 5.3.

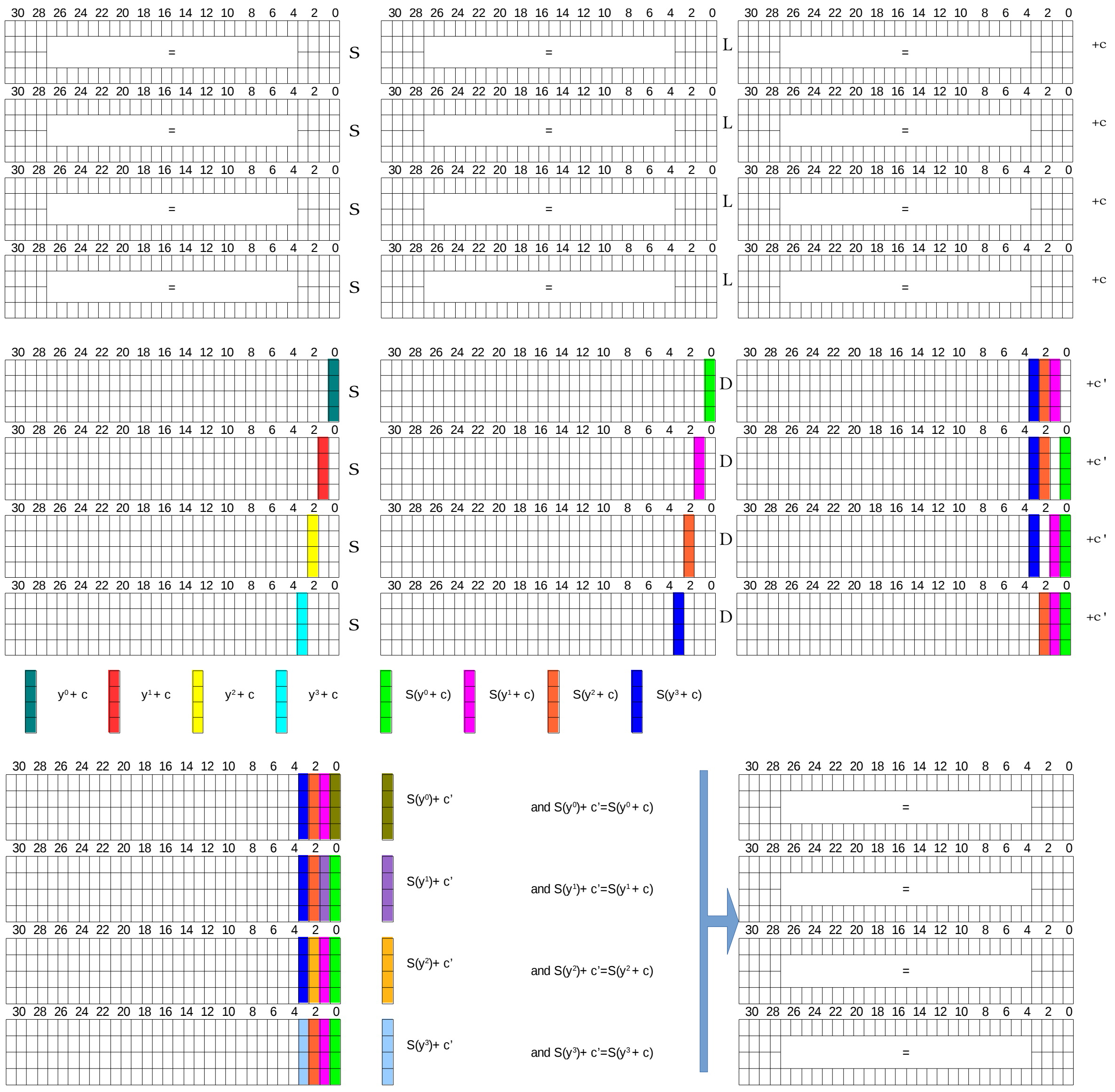


Figure 5.3: Evolution of two rounds with a starting 4-identical state, where the four bundles are equal in the beginning.

Probability of Maintaining the 4-Identical Property through a Step. We start from a 4-identical state X that we write $X = (x, x, x, x)$. Each bundle x is made of 32 columns: $x = (x^{31}, x^{30}, \cdots, x^1, x^0)$.

- **Application of the Super S-boxes.** The step starts with one non-linear layer followed by the L layer, applied in parallel (that is, independently) on each of the 4 bundles. Since these transformations are identical for each bundle the 4-identical property is followed with probability one up to this point and so we have $L \circ S(X) = (y, y, y, y)$ with $y = L \circ S(x)$. We next have the addition of the first round constant on column j of bundle j for $j \in \{0, 1, 2, 3\}$, that we will call AC, and finally we apply another S-box layer. By denoting the round constant by[2] c, we obtain the following values for $S \circ AC(2i) \circ L \circ S(X)$:

$$\begin{array}{lcccccc}
B_0: & S(y^{31}) & \cdots & S(y^4) & S(y^3) & S(y^2) & S(y^1) & S(y^0 \oplus c) \\
B_1: & S(y^{31}) & \cdots & S(y^4) & S(y^3) & S(y^2) & S(y^1 \oplus c) & S(y^0) \\
B_2: & S(y^{31}) & \cdots & S(y^4) & S(y^3) & S(y^2 \oplus c) & S(y^1) & S(y^0) \\
B_3: & S(y^{31}) & \cdots & S(y^4) & S(y^3 \oplus c) & S(y^2) & S(y^1) & S(y^0)
\end{array}$$

where B_i is bundle i. At this stage, the 4 bundles stop being 4-identical but differ on the value of their 4 first columns.

- **D-box and second round constant addition.** The D layer mixes together the 4 bundles by XORing 3 of them together to form one output bundle, as described in Section 5.2.1. In the above representation of the state, it operates columnwise by replacing each column element with the XOR of the 3 others. The last operation is the addition of the second round constant, that we denote c', at the same positions as before (column j of bundle j for $j \in \{0, 1, 2, 3\}$). Formally, the expression of the bundles of $AC(2i + 1) \circ D \circ S \circ AC(2i) \circ L \circ S(X)$ is the following:

$$\begin{array}{lcccccc}
B_0: & S(y^{31}) & \cdots & S(y^4) & S(y^3 \oplus c) & S(y^2 \oplus c) & S(y^1 \oplus c) & S(y^0) \oplus c' \\
B_1: & S(y^{31}) & \cdots & S(y^4) & S(y^3 \oplus c) & S(y^2 \oplus c) & S(y^1) \oplus c' & S(y^0 \oplus c) \\
B_2: & S(y^{31}) & \cdots & S(y^4) & S(y^3 \oplus c) & S(y^2) \oplus c' & S(y^1 \oplus c) & S(y^0 \oplus c) \\
B_3: & S(y^{31}) & \cdots & S(y^4) & S(y^3) \oplus c' & S(y^2 \oplus c) & S(y^1 \oplus c) & S(y^0 \oplus c)
\end{array}$$

To ensure a 4-identical state at this point, the following 4 equations need to be satisfied:

$$\begin{cases}
S(y^3 \oplus c) = & S(y^3) \oplus c' \\
S(y^2 \oplus c) = & S(y^2) \oplus c' \\
S(y^1 \oplus c) = & S(y^1) \oplus c' \\
S(y^0 \oplus c) = & S(y^0) \oplus c' \ .
\end{cases}$$

Depending on the values of c and c' – that vary with the index of the step – these 4 equations are either never verified or can be verified with a rather high probability. In fact, their number of solutions corresponds to the probability of the transition from a difference of c to a difference of c' through the S-box S. We computed the corresponding probabilities for all the steps and report the results in Table 5.3. Note that we experimentally verified these values.

Table 5.3: Probability that an output of step s of `Shadow` is 4-identical knowing that the input is.

s	0	1	2	3	4	5
Probability	0	0	0	2^{-12}	2^{-8}	0

5.3.3 3-Identical States

A similar reasoning applies to states for which only 3 (out of 4) bundles are equal. Without loss of generality, let us consider a 3-identical state $X = (x, x, x, z)$ for which the first 3 bundles are

[2] Recall here that the value of the round constant depends on the round index.

identical. The step starts with the application of the same operations to each bundle, namely S and L, we denote the modified state by $L \circ S(X) = (y, y, y, w)$. Once the other operations are applied the output becomes:

$$\begin{array}{lllll}
S(w^{31}), \cdots, & S(w^3 \oplus c), & S(y^2) \oplus S(y^2 \oplus c) \oplus S(w^2), & S(y^1) \oplus S(y^1 \oplus c) \oplus S(w^1), & S(w^0) \oplus c', \\
S(w^{31}), \cdots, & S(w^3 \oplus c), & S(y^2) \oplus S(y^2 \oplus c) \oplus S(w^2), & S(w^1) \oplus c', & S(y^0) \oplus S(y^0 \oplus c) \oplus S(w^0), \\
S(w^{31}), \cdots, & S(w^3 \oplus c), & S(w^2) \oplus c', & S(y^1) \oplus S(y^1 \oplus c) \oplus S(w^1), & S(y^0) \oplus S(y^0 \oplus c) \oplus S(w^0), \\
S(y^{31}), \cdots, & S(y^3) \oplus c', & S(y^2 \oplus c), & S(y^1 \oplus c), & S(y^0 \oplus c)
\end{array}$$

To ensure a 3-identical state the following equations thus have to be satisfied:

$$\begin{aligned}
S(y^2 \oplus c) &= S(y^2) \oplus c' \\
S(y^1 \oplus c) &= S(y^1) \oplus c' \\
S(y^0 \oplus c) &= S(y^0) \oplus c' .
\end{aligned}$$

In this case, since we have one fewer S-box transition to constrain, the probabilities of Table 5.3 increase to the ones provided in Table 5.4.

Table 5.4: Probability that an output of step s of `Shadow` is 3-identical knowing that the input is.

s	0	1	2	3	4	5
Probability	0	0	0	2^{-9}	2^{-6}	0

These probabilities do not depend on the choice of the positions of the 3 input bundles that are identical. Instead, they are valid as soon as the 3 positions are the same in the input and in the output.

5.3.4 2-Identical States

We can follow a similar reasoning to obtain the probability of keeping a 2-identical state. We obtain two equations to solve, and the probabilities become the ones given in Table 5.3. Again, the position of the 2 identical bundles does not impact these probabilities but it has to be the same in the input and in the output.

Table 5.5: Probability that an output of step s of `Shadow` is 2-identical knowing that the input is.

s	0	1	2	3	4	5
Probability	0	0	0	2^{-6}	2^{-4}	0

5.4 A Distinguisher Against Full `Shadow-512` (and More)

In this section, we present a practical distinguisher which allows us to exhibit pairs (x, x') of 512-bit inputs of the `Shadow-512` permutation such that

$$x \oplus x' = (*, *, *, 0) \text{ and } \pi(x) \oplus \pi(x') = D(0, 0, 0, *) , \tag{5.1}$$

for the full version and such that

$$x \oplus x' = (*, *, *, 0) \text{ and } \pi(x) \oplus \pi(x') = D(*, *, *, 0) , \tag{5.2}$$

for a "round-extended" version of `Shadow-512` using 7 steps rather than 6. In other words, we efficiently solve a limited-birthday problem.

As proved by Iwamoto et al. [IPS13], generating such pairs for a random permutation would require roughly 2^{64} queries. However, we can produce pairs satisfying Property (5.1) for full `Shadow-512` using about 2^{15} calls to said permutation. The exact same technique finds pairs satisfying Property (5.2) for 7-step `Shadow-512`. The corresponding procedures are described in Section 5.4.2.

These distinguishers hinge on two properties: the propagation of 3-identical states which we described in Section 5.3.3, and a probability 1 truncated differential explained in Section 5.4.1. The latter can be used directly as a distinguisher for 10-round `Shadow-512`.

5.4.1 A 5-Step Truncated Differential Property

We start by devising a distinguisher of `Shadow-512` reduced to 5 steps out of 6. The truncated trail we use is summarized in Figure 5.4. Starting from the middle, we can easily construct pairs of states such that their difference propagates with probability 1 over 2 forward and 2 backward steps.

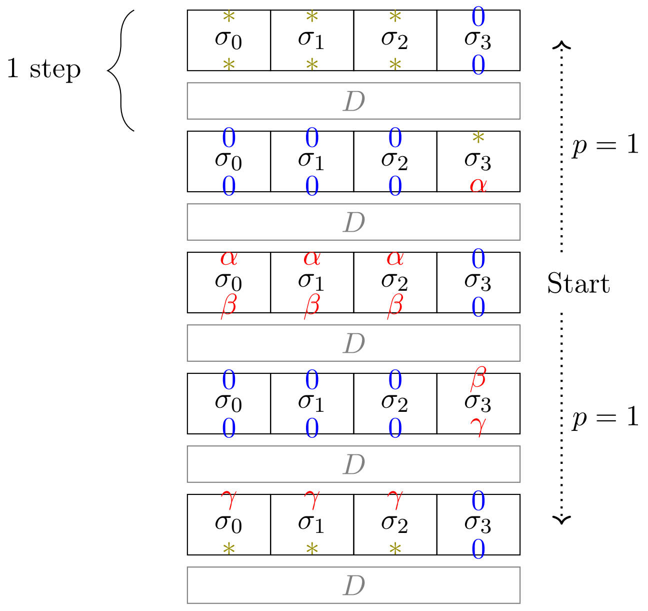


Figure 5.4: A 5-step distinguisher against the 512-bit permutation `Shadow`.

All propagations are of probability 1, the only place where we would *a priori* have to pay for the cost of a transition is for the three Super S-box level transitions $\alpha \rightsquigarrow \beta$ in step 2. However, the high similarity between the Super S-boxes provides us with a simple way to obtain three such pairs of 128-bit blocks.

Building pairs of bundles that follow the same differential for different Super S-boxes. Recall that the only difference between two Super S-boxes lies in the constant addition operation that is done right after the L linear layer. The 4-bit constant c is added to the input of only one S-box of the second non-linear layer, and the index of this S-box depends on the Super S-box index.

Thanks to this limited difference between the Super S-boxes, we can easily build an input difference α so that the output difference of the S-box does not depend on its index. More precisely, this difference α should be chosen so that it does not diffuse to the last 4 columns of the bundle. This simple fact is formalized in the following lemma:

Lemma 5.2. *If $x \in \mathbb{F}_2^{128}$ and $\alpha \in \mathbb{F}_2^{128}$ are such that $(L \circ S)(x) \oplus (L \circ S)(x \oplus \alpha) = \beta$ and if β is set to 0 on the 4 S-boxes that can receive the round constant c, then the value of $\sigma_b(x) \oplus \sigma_b(x \oplus \alpha)$ does not depend on the bundle index b.*

Proof. We denote by y and $y \oplus \beta$ the respective values of $(L \circ S)(x)$ and $(L \circ S)(x \oplus \alpha)$. By expanding these into the column notation we get:

$$\begin{array}{rcccccccc} y = & y^{31} & \cdots & y^4 & y^3 & y^2 & y^1 & y^0 \\ y \oplus \beta = & y^{31} \oplus \beta^{31} & \cdots & y^4 \oplus \beta^4 & y^3 & y^2 & y^1 & y^0 \end{array}$$

Let us first look at σ_0. We have that

$$\begin{array}{rcccccccc} \sigma_0(x) = & S(y^{31}) & \cdots & S(y^4) & S(y^3) & S(y^2) & S(y^1) & S(y^0 \oplus c) \\ \sigma_0(x \oplus \alpha) = & S(y^{31} \oplus \beta^{31}) & \cdots & S(y^4 \oplus \beta^4) & S(y^3) & S(y^2) & S(y^1) & S(y^0 \oplus c) \end{array}$$

so summing these equations yields

$$\begin{array}{rcccccccc} \sigma_0(x) \oplus \sigma_0(x \oplus \alpha) = & \gamma^{31} & \cdots & \gamma^4 & 0 & 0 & 0 & 0 \end{array}$$

Without loss of generality, let us now consider σ_1. We have

$$\begin{array}{rcccccccc} \sigma_1(x) = & S(y^{31}) & \cdots & S(y^4) & S(y^3) & S(y^2) & S(y^1 \oplus c) & S(y^0) \\ \sigma_1(x \oplus \alpha) = & S(y^{31} \oplus \beta^{31}) & \cdots & S(y^4 \oplus \beta^4) & S(y^3) & S(y^2) & S(y^1 \oplus c) & S(y^0) \end{array}$$

As we can see we have the exact same pairs of values, and thus the same output differences, unless we look at one of the first 4-bit nibbles. However, in this case, the values in $\sigma_1(x)$ and $\sigma_1(x \oplus \alpha)$ are identical to one another, meaning that their difference is equal to 0 as well. This concludes the proof since the two differences are equal. □

To put it differently, this lemma allows us to build pairs of messages that follow the same differential trail over one step whatever the index of the Super S-box.

Our distinguisher for 5 steps of `Shadow-512` thus works by following the process described in Algorithm 13. As depicted in Figure 5.4, the choice of the difference β ensures that the same transition is followed for the 3 first Super S-boxes of step 2. The differential pattern then propagates as expected with probability 1 through steps 1 then 0 (backward), and 3 then 4 (forward).

We have verified experimentally that this distinguisher works as predicted.

The differences in the output of the Super S-box layer of step 5 (denoted by $*$ in Figure 5.4) are *a priori* different from one another, meaning that this approach cannot cover more rounds. Fortunately, we can use the property studied in Section 5.3.3 to our advantage, as explained below.

Algorithm 13 A distinguisher for 5-step `Shadow`.

1. Choose $\beta \in \mathbb{F}_2^{128}$ such that it is set to 0 on the 4 S-boxes of lowest weight
2. Choose a random $y \in \mathbb{F}_2^{128}$ and a random $z \in \mathbb{F}_2^{128}$
3. Compute $x = \sigma_0^{-1}(y)$ and $x + \alpha = \sigma_0^{-1}(y + \beta)$,
4. Set the two states at step 2 to be
$$X_2 = (x,\ x,\ x,\ z) \ \text{ and } \ X_2' = (x + \alpha,\ x + \alpha,\ x + \alpha,\ z)\ .$$
5. Invert step 1 and step 0 on X_2 and X_2' to obtain a pair of states (X_0, X_0') such that $\pi(X_0) \oplus \pi(X_0') = D(*, *, *, 0)$.

5.4.2 A Distinguisher for 6- and 7-Step `Shadow`

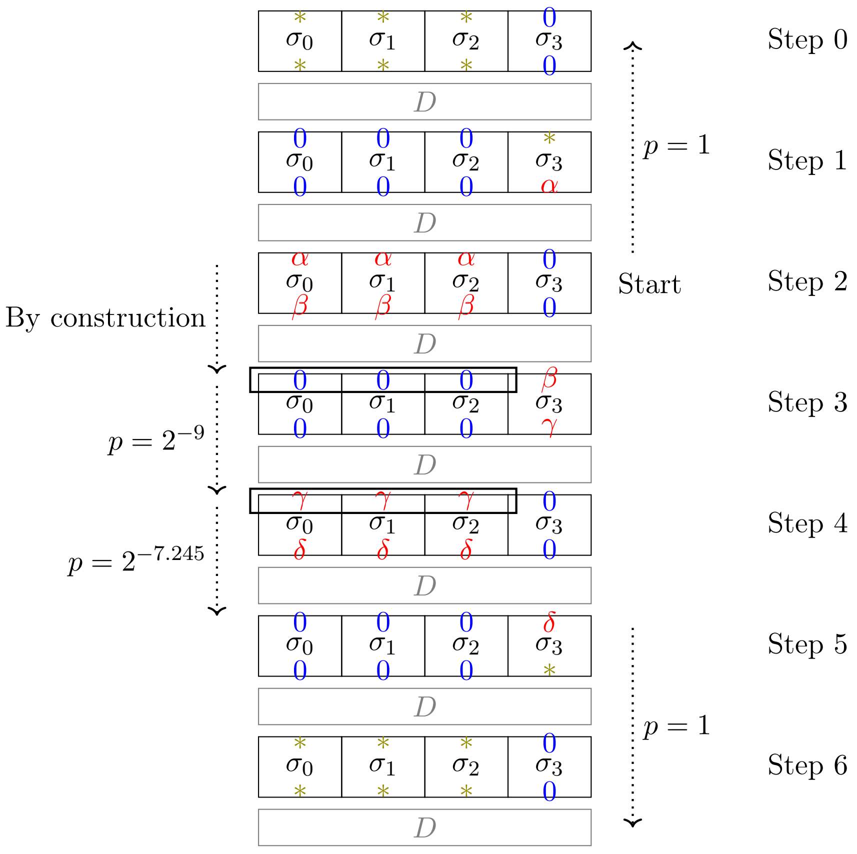


Figure 5.5: A 7-step distinguisher against the 512-bit permutation `Shadow`.

Using the truncated trail discussed in Section 5.4.1 with the observation in Section 5.3.3, we can build a distinguisher on 6 steps of `Shadow`, *i.e.* on the full permutation. It would naturally extend to a distinguisher on 7 steps if we defined such a "round-extended" variant of `Shadow`. This distinguisher is summarized in Figure 5.5.

Structure of the distinguisher. Our distinguisher works as follows.

- We first focus on the input of step 2 and build a pair of messages that differ by $(\alpha, \alpha, \alpha, 0)$. This difference automatically sets the input difference of step 0 to be equal to 0 on the third bundle. Our choice of the two messages must also ensure that their difference at the end of step 2 is equal to $(0, 0, 0, \beta)$ and that the two output messages are 3-identical (the 3-identical property is depicted by the thick rectangle in Figure 5.5.).
- In step 3, we want to keep the 3-identical property in order to ease the following step. As we established before (see Table 5.4), this event has a probability equal to 2^{-9}. The input difference at the end of step 3 is then equal to $(\gamma, \gamma, \gamma, 0)$.
- We next aim for a difference equal to $(\delta, \delta, \delta, 0)$ at the output of the Super S-boxes of step 4, an event whose probability we later prove to be $2^{-7.245}$. When this condition is fulfilled we obtain a difference equal to $(0, 0, 0, \delta)$ at the output of step 4, which automatically leads to the required difference of the form $(*, *, *, 0)$ at the end of step 5.

Let us now show how we can efficiently find two states that verify the conditions at the input and the output of Step 3.

Suppose that there is an $x \in \mathbb{F}_2^{128}$ such that the following holds during step 2 for some 128-bit values α, β, ϵ and ϵ':

$$\begin{cases} \sigma_0(x) + \sigma_0(x + \alpha) & = \beta \\ \sigma_1(x + \epsilon) + \sigma_1(x + \epsilon + \alpha) & = \beta \\ \sigma_2(x + \epsilon') + \sigma_2(x + \epsilon' + \alpha) & = \beta \ , \end{cases} \tag{5.3}$$

these constraints corresponding to the differential trail at step 2 that is used in Figure 5.5. Such an x would allow us to run the 5-round distinguisher described in Section 5.4.1. However, as we explained, the property would not extend beyond the fifth step. To achieve this, we add another set of constraints:

$$\begin{cases} (AC \circ D)\big(\sigma_0(x),\ \sigma_1(x + \epsilon),\ \sigma_2(x + \epsilon'),\ z\big) & = (y,\ y,\ y,\ z') \\ (AC \circ D)\big(\sigma_0(x + \alpha),\ \sigma_1(x + \alpha + \epsilon),\ \sigma_2(x + \alpha + \epsilon'),\ z\big) & = (y,\ y,\ y,\ z' + \beta) \ . \end{cases} \tag{5.4}$$

In other words, we impose that each state we consider is 3-identical.

In this case, the difference between the states has to be equal to $(0, 0, 0, \beta)$ at the input of step 3. Furthermore, each state is 3-identical. This property is carried over to the next step with some probability. Should this happen, we would have at the input of step 4 that the difference between the states is equal to $(0, 0, 0, \gamma)$ for some $\gamma \in \mathbb{F}_2^{128}$, and that each state is 3-identical.

Finding Solutions for Properties (5.3) and (5.4). It turns out that a specific probability 1 truncated differential pattern allows us to trivially find solutions satisfying both Property (5.3) and Property (5.4).

Indeed, we remark that:

1. the impact of the constant additions both within the Super S-box layers and outside it (after the D layer) is limited to the S-boxes with indices in $\{0, 1, 2, 3\}$ (i.e. the 4 of lowest weight) within each Super S-box, and
2. the bits with indices 22 and 23 in each of the 4 input words of a Super S-box do not influence the output bits with indices in $\{0, 1, 2, 3\}$.

Using the reference implementation, we can indeed see that

$$\begin{aligned} L(0, e_{22}) &= (\texttt{1b880510},\ \texttt{6c06f000}) \\ L(e_{22}, 0) &= (\texttt{36037800},\ \texttt{1b880510}) \\ L(0, e_{23}) &= (\texttt{37100a20},\ \texttt{d80de000}) \\ L(e_{23}, 0) &= (\texttt{6c06f000},\ \texttt{37100a20})\ , \end{aligned}$$

where the 4 bits of lowest weight in each output are always equal to 0. We then define the vector space $\nabla \subset (\mathbb{F}_2^4)^{32}$ as

$$\nabla = \{a \times e_{22} + b \times e_{23}, a \in \mathbb{F}_2^4, b \in \mathbb{F}_2^4\}\ ,$$

where the multiplications are done in the finite field $\mathbb{F}_2^4$. As a consequence of our observations, we have the following lemma.

Lemma 5.3. *Let $x \in (\mathbb{F}_2^4)^{32}$ be a 128-bit vector and let $\alpha \in \nabla$ be a difference. Then for all steps and all bundle index i, we have that*

$$\sigma_i(x) + \sigma_i(x + \alpha) = (*, *, ..., *, 0, 0, 0, 0)\ .$$

As evidenced by our experimental results (see below), this approach is efficient, and with the cost of computing 1 Super S-box we can obtain about 2^{16} internal states that verify the condition of step 2.

Description of the Full Distinguisher. Algorithm 14 details our distinguisher. Using the techniques described so far, we are able to find input differences that satisfy the truncated trail and are 3-identical where needed (see Figure 5.5) from the beginning of step 0 to the end of step 2. Let us now see what happens in the remaining steps.

Algorithm 14 Our 7-step distinguisher against `Shadow`.

Output: A pair (x, y, z, t), (x', y', z', t) such that $\pi(x, y, z, t) \oplus \pi(x', y', z', t) = (*, *, *, 0)$ with probability at least $2^{-16.245}$ after 7-step `Shadow-512`.

1. Select a difference $\epsilon \in \nabla$.
2. Select a state (y_2, y_2, y_2, z_2) that will be a state after step 2.
3. Invert step 2 on (y_2, y_2, y_2, z_2), obtaining (x_1, y_1, z_1, t_1).
4. Invert step 1 on (x_1, y_1, z_1, t_1) and $(x_1 \oplus \epsilon, y_1 \oplus \epsilon, z_1 \oplus \epsilon, t_1)$, obtaining (x_0, y_0, z_0, t_0) and (x_0, y_0, z_0, t_0').
5. Invert step 0, obtaining a pair of `Shadow-512` states with a zero-difference in the last bundle.
6. Return this pair of state. With high probability ($\geq 2^{-16.24}$), it satisfies the truncated trail in Figure 5.5.

Step 3. We start from two messages that are built such that at the end of step 2 they are 3-identical, and we want that the two messages are again 3-identical at the end of step 3. With a reasoning similar to the one given in Section 5.3.3, we obtain 6 equations to solve, while in fact only 3 are independent (the 3 equations obtained for the second message are the same as the 3 obtained for the first message since they only differ on the last bundle), and as detailed in Table 5.4 the probability is equal to 2^{-9} since this is step number 3.

Step 4. Our objective is to obtain a difference of the form $(0, 0, 0, \delta)$ for any non-zero δ in $\mathbb{F}_2^{128}$ at the beginning of step 5 (see Figure 5.5). In order for this to happen, we need to have a difference equal to $(\delta, \delta, \delta, 0)$ at the end of step 4. To estimate the probability of this event, let us write the corresponding equations. We denote the two messages after the application of S and L of step 4 by (y, y, y, w) and (y', y', y', w) respectively. Since the input of step 4 is 3-identical, $y^i = y'^i$ for all $i > 3$. The expression of the last 4 column values at the end of step 4 (i.e. after applying D and AC) is then as follows for (y, y, y, w)

$$\begin{array}{llll}
S(w^3 \oplus c) & S(y^2) \oplus S(y^2 \oplus c) \oplus S(w^2) & S(y^1) \oplus S(y^1 \oplus c) \oplus S(w^1) & S(w^0) \oplus c' \\
S(w^3 \oplus c) & S(y^2) \oplus S(y^2 \oplus c) \oplus S(w^2) & S(w^1) \oplus c' & S(y^0) \oplus S(y^0 \oplus c) \oplus S(w^0) \\
S(w^3 \oplus c) & S(w^2) \oplus c' & S(y^1) \oplus S(y^1 \oplus c) \oplus S(w^1) & S(y^0) \oplus S(y^0 \oplus c) \oplus S(w^0) \\
S(y^3) \oplus c' & S(y^2 \oplus c) & S(y^1 \oplus c) & S(y^0 \oplus c)
\end{array}$$

and as follows for (y', y', y', w):

$$\begin{array}{llll}
S(w^3 \oplus c) & S(y'^2) \oplus S(y'^2 \oplus c) \oplus S(w^2) & S(y'^1) \oplus S(y'^1 \oplus c) \oplus S(w^1) & S(w^0) \oplus c' \\
S(w^3 \oplus c) & S(y'^2) \oplus S(y'^2 \oplus c) \oplus S(w^2) & S(w^1) \oplus c' & S(y'^0) \oplus S(y'^0 \oplus c) \oplus S(w^0) \\
S(w^3 \oplus c) & S(w^2) \oplus c' & S(y'^1) \oplus S(y'^1 \oplus c) \oplus S(w^1) & S(y'^0) \oplus S(y'^0 \oplus c) \oplus S(w^0) \\
S(y'^3) \oplus c' & S(y'^2 \oplus c) & S(y'^1 \oplus c) & S(y'^0 \oplus c)
\end{array}$$

In order for the sum of these two states to be equal to $(0, 0, 0, \delta)$ (for any non-zero δ), the following relations have to be satisfied:

$$\begin{aligned}
S(y'^2) \oplus S(y'^2 \oplus c) &= S(y^2) \oplus S(y^2 \oplus c) \\
S(y'^1) \oplus S(y'^1 \oplus c) &= S(y^1) \oplus S(y^1 \oplus c) \\
S(y'^0) \oplus S(y'^0 \oplus c) &= S(y^0) \oplus S(y^0 \oplus c) \ .
\end{aligned}$$

Since we are looking at step 4, the constant c is equal to $0x5$ and then each equality has a probability equal to $2^{-2.415}$ to be verified (assuming that the value of y and y' are independent).

Step 5. This last step is passed with probability one, so in the end we observe an output difference equal to $(*, *, *, 0)$ with a probability at least equal to $(2^{-2.415})^3 \times 2^{-9} = 2^{-16.245}$.

Step 6. One additional round can be added with probability one, since by inverting D we would find a difference equal to 0 in the last bundle with the same probability of $2^{-16.245}$.

Experimental Results. Experiments showed that the probability of the distinguisher is slightly higher than what we expected, since in fact the previously detailed trail is not the only one that leads to the required output difference (see the next subsection for a description of another valid trail). By running Algorithm 14 for 2^{22} times, we obtained 124 successful pairs, a probability close to 2^{-15}. Our unoptimized C++ implementation found all these pairs in less than 30 seconds on a desktop computer. Below is an example for 7 steps.

x_1				x_2			
9c7fbdf0	4a9a3523	90bd4f15	33e12e8f	b4764864	aaabc55e	2b65df83	33e12e8f
5554509d	5ea7c50d	db9fd14e	8cd31faf	30d8625c	6d513db3	9024c477	8cd31faf
5f0785c3	14ce1b1f	b9a7f521	336e44ba	89fb6758	5d19b594	e69ccd64	336e44ba
fcf630fb	82cafa8e	abf5b881	e5534b79	4f3d62a5	3e530b8b	f7ccf2b7	e5534b79

$x_1 \oplus x_2$				$\pi(x_1) \oplus \pi(x_2)$			
2809f594	e031f07d	bbd89096	00000000	39e368a5	03e51caf	f2d7ae55	00000000
658c32c1	33f6f8be	4bbb1539	00000000	2668956a	b1720999	00c93f81	00000000
d6fce29b	49d7ae8b	5f3b3845	00000000	4aed9270	2b317fb5	6f1a183b	00000000
b3cb525e	bc99f105	5c394a36	00000000	d902b8fd	5c7db7c2	2ef09921	00000000

Another High Probability Characteristic over 7 Steps

As previously stated, the trail that is represented in Figure 5.5 is not the only one contributing to the probability of our 7-step distinguisher. Indeed, while we expected a probability of $2^{-16.245}$, our experiments returned a probability close to 2^{-15}. In this section we detail a second trail of high probability that benefits from the definition of L, namely from the fact that the bits in column 2 do not diffuse to column 0, 1 and 2.

Structure of the trail. The trail is represented in Figure 5.6 and works as follows:

- As before, our construction at step 2 gives a pair of messages that leads to the desired difference $(*, *, *, 0)$ at the input of the permutation with probability 1, while the states at the output of step 2 are 3-identical and differ by $(0, 0, 0, \beta)$.
- With probability 2^{-9}, the two states keep their 3-identical property at the end of step 3, and their difference is $(\gamma, \gamma, \gamma, 0)$.
- We then require that at the end of step 4 the two states are 2-identical while they share the same third bundle value. As we detail next, this event is of probability $2^{-8.3}$.
- Step 5 ends with a null difference in the last bundle with probability 1 thanks to the definition of L. The distinguisher can be extended to a 7-step one for free.

The total probability of this trail is thus equal to $2^{-17.3}$. Adding it to the probability of the other trail previously discussed in this section, we obtain something closer to what is observed experimentally: $2^{-17.3} + 2^{-16.245} = 2^{-15.678}$. Note that other trails add up to this probability, for instance the ones with a difference of the form $(0, \tau, \tau, \kappa)$ or $(\tau, 0, \tau, \kappa)$ at the end of step 4.

Detail of the probabilities.

Step 3. As for the first trail described in this section, the probability that the states remain 3-identical is equal to 2^{-9}.

Step 4. At the end of step 4, we aim for a pair of messages that are 2-identical in their first 2 bundles and that have no difference in their third bundle. Formally, let us denote by (y, y, y, w) and (y', y', y', w) the two states after S and L. After applying the first constant addition, the second S-layer, the D operation and the second constant addition, we obtain:

$$\begin{array}{lrrrr}
S(w^{31}), \cdots, & S(w^3 \oplus c), & S(y^2) \oplus S(y^2 \oplus c) \oplus S(w^2), & S(y^1) \oplus S(y^1 \oplus c) \oplus S(w^1), & S(w^0) \oplus c', \\
S(w^{31}), \cdots, & S(w^3 \oplus c), & S(y^2) \oplus S(y^2 \oplus c) \oplus S(w^2), & S(w^1) \oplus c', & S(y^0) \oplus S(y^0 \oplus c) \oplus S(w^0), \\
S(w^{31}), \cdots, & S(w^3 \oplus c), & S(w^2) \oplus c', & S(y^1) \oplus S(y^1 \oplus c) \oplus S(w^1), & S(y^0) \oplus S(y^0 \oplus c) \oplus S(w^0), \\
S(y^{31}), \cdots, & S(y^3) \oplus c', & S(y^2 \oplus c), & S(y^1 \oplus c), & S(y^0 \oplus c)
\end{array}$$

and

$$\begin{array}{lrrrr}
S(w^{31}), \cdots, & S(w^3 \oplus c), & S(y'^2) \oplus S(y'^2 \oplus c) \oplus S(w^2), & S(y'^1) \oplus S(y'^1 \oplus c) \oplus S(w^1), & S(w^0) \oplus c', \\
S(w^{31}), \cdots, & S(w^3 \oplus c), & S(y'^2) \oplus S(y'^2 \oplus c) \oplus S(w^2), & S(w^1) \oplus c', & S(y'^0) \oplus S(y'^0 \oplus c) \oplus S(w^0), \\
S(w^{31}), \cdots, & S(w^3 \oplus c), & S(w^2) \oplus c', & S(y'^1) \oplus S(y'^1 \oplus c) \oplus S(w^1), & S(y'^0) \oplus S(y'^0 \oplus c) \oplus S(w^0), \\
S(y'^{31}), \cdots, & S(y'^3) \oplus c', & S(y'^2 \oplus c), & S(y'^1 \oplus c), & S(y'^0 \oplus c)
\end{array}$$

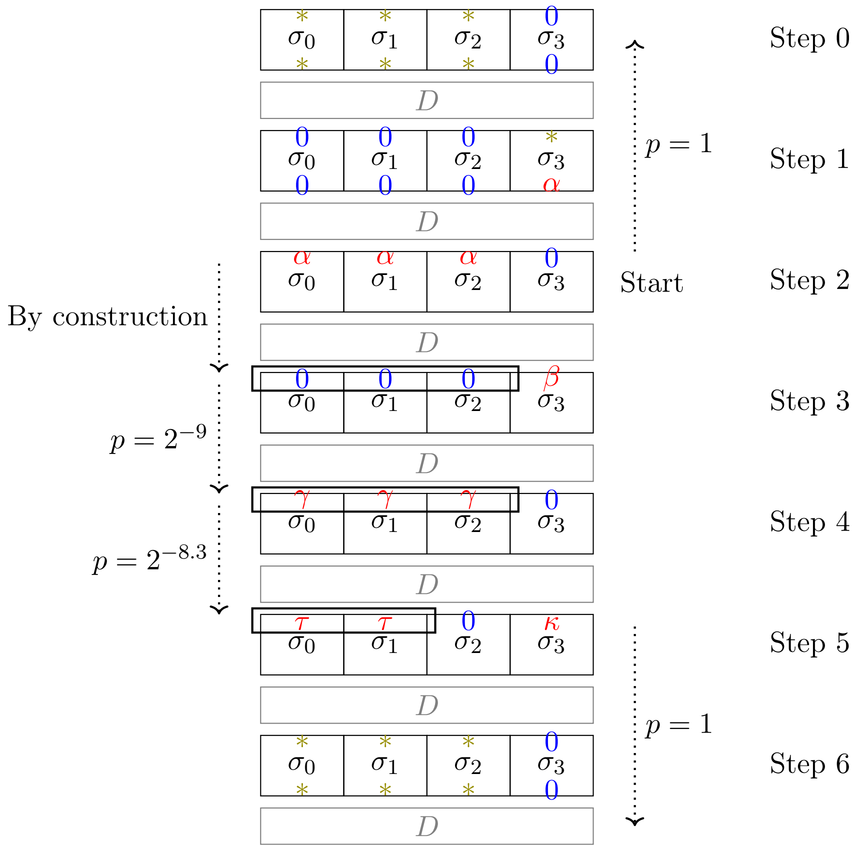


Figure 5.6: Another trail contributing to the probability of the 7-step distinguisher of Shadow-512.

In order to obtain a 2-identical state the following relations have to be satisfied:

$$
\begin{aligned}
S(y^1) \oplus S(y^1 \oplus c) &= c', \\
S(y^0) \oplus S(y^0 \oplus c) &= c', \\
S(y'^1) \oplus S(y'^1 \oplus c) &= c', \\
S(y'^0) \oplus S(y'^0 \oplus c) &= c'.
\end{aligned}
$$

Given that we are looking at step number 4 we have $c = 0x5$ and $c' = 0xa$, so each equation is verified with probability 2^{-2}. Also, since we aim for a difference at the end of step 4 of the form $(\tau, \tau, 0, \kappa)$ with $\tau \neq 0$, we have to add the condition:

$$S(y^2) \oplus S(y^2 \oplus c) \oplus S(y'^2) \oplus S(y'^2 \oplus c) \neq 0.$$

That is verified with probability $2^{-0.3}$. Consequently, the probability of step 4 is equal to $2^{-8.3}$. Once these conditions are fulfilled, we automatically have an output difference of step 4 equal to $(\tau, \tau, 0, \kappa)$ and the actual value of τ is very sparse, only the second column is active:

$$\tau = (0, \cdots, \qquad 0, \qquad S(y'^2) \oplus S(y'^2 \oplus c) \oplus S(y^2) \oplus S(y^2 \oplus c), \qquad 0, \qquad 0)$$

This particular shape implies that step 5 is passed with probability 1.

Step 5. We denote the two input states by (u, u, v, x) and $(u \oplus \tau, u \oplus \tau, v, x')$. Our goal is to obtain a difference equal to zero in the last bundle at the end of the step. We first remark that after applying the first S-layer to the two states, we obtain two states (U, U, V, X) and (U', U', V, X') so that again the difference between U and U' is only positioned in the second column of the bundle (simply because the S-layer modifies each column independently). We denote the new difference by $T = U \oplus U'$.

Due to the linearity of the next step we can further trace the evolution of τ through the L layer: we have that $L(U) \oplus L(U') = L(T)$. Moreover, using the specification of L we observe that:

$$\begin{aligned} L(e_2, 0) &= (\texttt{805101b8},\ \ \texttt{6f0006c0}) \\ L(0, e_2) &= (\texttt{37800360},\ \ \texttt{805101b8}). \end{aligned}$$

These computations indicate that any difference positioned in column 2 does not propagate to any of the first 3 columns, and in particular that whatever the exact value of T the two first bundles of each state have the same value over their 3 first columns. To see how this leads to the required equality at the end of step 5, we can look at the formal expression of the two states. After applying L, the first round constant addition and the second non-linear layer we obtain:

$$\begin{array}{lrrrr} S(L(U)^{31}), \cdots, & S(L(U)^3), & S(L(U)^2), & S(L(U)^1), & S(L(U)^0 \oplus c), \\ S(L(U)^{31}), \cdots, & S(L(U)^3), & S(L(U)^2), & S(L(U)^1 \oplus c), & S(L(U)^0), \\ S(L(V)^{31}), \cdots, & S(L(V)^3), & S(L(V)^2 \oplus c), & S(L(V)^1), & S(L(V)^0), \\ S(L(X)^{31}), \cdots, & S(L(X)^3 \oplus c), & S(L(X)^2), & S(L(X)^1), & S(L(X)^0) \end{array}$$

for the first state, and the following for the second state:

$$\begin{array}{lrrrr} S(L(U')^{31}), \cdots, & S(L(U')^3), & S(L(U')^2), & S(L(U')^1), & S(L(U')^0 \oplus c), \\ S(L(U')^{31}), \cdots, & S(L(U')^3), & S(L(U')^2), & S(L(U')^1 \oplus c), & S(L(U')^0), \\ S(L(V)^{31}), \cdots, & S(L(V)^3), & S(L(V)^2 \oplus c), & S(L(V)^1), & S(L(V)^0), \\ S(L(X')^{31}), \cdots, & S(L(X')^3 \oplus c), & S(L(X')^2), & S(L(X')^1), & S(L(X')^0) \end{array}$$

The difference in the last bundle at the end of step 5 is thus given by the sum of the first 3 bundles of both states (since we are passing through D). It gives:

$$0, \cdots, 0, S(L(U)^1) \oplus S(L(U)^1 \oplus c) \oplus S(L(U')^1) \oplus S(L(U')^1 \oplus c), S(L(U)^0) \oplus S(L(U)^0 \oplus c) \oplus S(L(U')^0) \oplus S(L(U')^0 \oplus c).$$

We then use the previous observation which implies that $L(U')^1 = L(U)^1$ together with $L(U')^0 = L(U)^0$ to conclude that the bundle difference is null with probability 1.

5.4.3 A Distinguisher for 6-step `Shadow-384`

In this section we show how to build a similar distinguisher on 6 steps of the 384-bit variant of `Shadow` shifted by one round (i.e. which works for steps from 1 to 6 but for no steps from 0 to 5 because of the round constants). As explained in the preliminaries, `Shadow-384` is defined as a 3LS-design, and the D layer acts on three 128-bit bundles a, b, c as follows:

$$D(a, b, c) = \begin{pmatrix} I & I & I \\ I & 0 & I \\ I & I & 0 \end{pmatrix} \times \begin{pmatrix} a \\ b \\ c \end{pmatrix}$$

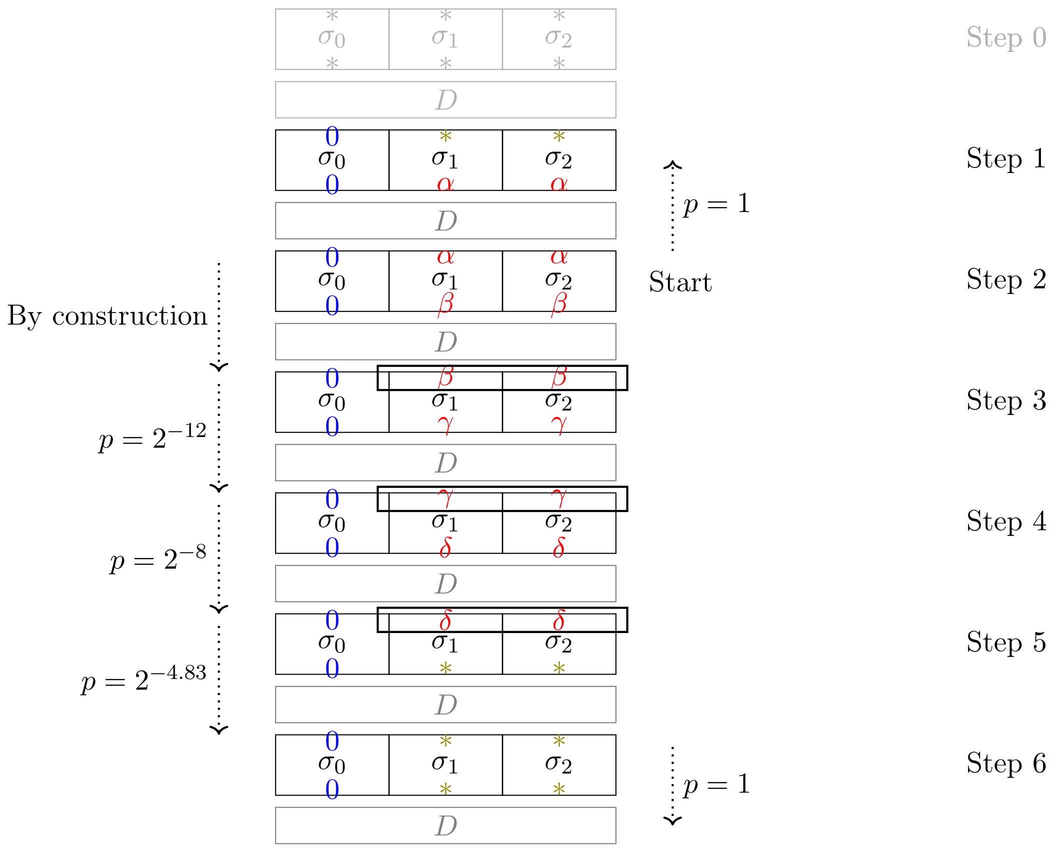


Figure 5.7: A (1-step shifted) 6-step distinguisher for `Shadow-384`. The thick rectangles depict 2-identical states.

Interestingly, propagating identical states remains possible with this layer, more specifically for states in which the last 2 bundles are equal. Using this property, one can exhibit pairs (x, x') such that $x \oplus x' = (0, *, *)$ at step 1 and $\pi(x) \oplus \pi(x') = D(0, *, *)$ at the end of step 6. Note that in this case, we cannot cover step 0. Hence this is not a distinguisher on the full version of `Shadow-384` for which we can cover only 5 steps. However, it shows that adding 1 more step at the end does not increase the security of `Shadow-384`. The distinguisher is summarized in Figure 5.7.

As previously described in Section 5.4.2, by picking an α in the vector space $\nabla = \{a \times e_{22} + b \times e_{23}, a \in \mathbb{F}_2^4, b \in \mathbb{F}_2^4\}$ we can easily find two states $(x_1, y_1, y_1 + \epsilon)$ and $(x_1, y_1 + \alpha, y_1 + \epsilon + \alpha)$ as inputs to step 2 that satisfy the following properties at input of step 3:

$$\begin{cases} \sigma_1(y_1) + \sigma_1(y_1 + \alpha) & = \beta \\ \sigma_2(y_1 + \epsilon) + \sigma_2(y_1 + \epsilon + \alpha) & = \beta \ , \end{cases} \tag{5.5}$$

and

$$\begin{cases} (AC \circ D)\big(x_1,\ \sigma_1(y_1),\ \sigma_2(y_1 + \epsilon)\big) & = (x_2,\ y_2,\ y_2) \\ (AC \circ D)\big(x_1,\ \sigma_1(y_1 + \alpha),\ \sigma_2(y_1 + \alpha + \epsilon)\big) & = (x_2,\ y_2 + \beta,\ y_2 + \beta) \ . \end{cases} \tag{5.6}$$

By inverting step 1, we obtain a difference $(0, *, *)$ with probability 1.

Now at step 3, the input difference equals $(0, \beta, \beta)$ and the last two bundles of each state are identical. With probability 2^{-12} and 2^{-8} respectively, the 2-identical states are preserved through step 3 and 4. Using the same notations as in Section 5.3.2, these probabilities are explained below.

Starting from a 2-identical state $X = (x, y, y)$, let $(w, z, z) = L \circ S(X)$ with $w = L \circ S(x)$ and $z = L \circ S(y)$. The first round constant c is then added on column j of bundle j for $j \in \{0, 1, 2\}$, and another S-box layer is applied, and we obtain the following:

$$S \circ AC(2i) \circ L \circ S(X) = \begin{bmatrix} \cdots & S(w^i) & \cdots & S(w^2) & S(w^1) & S(w^0 \oplus c) \\ \cdots & S(z^i) & \cdots & S(z^2) & S(z^1 \oplus c) & S(z^0) \\ \cdots & S(z^i) & \cdots & S(z^2 \oplus c) & S(z^1) & S(z^0) \end{bmatrix} .$$

At this stage, the last 2 bundles of each state differ only on the value of their second and third columns. After the D layer and the addition of the second round constant c' at column j of bundle j for $j \in \{0, 1, 2\}$) as before, the expression of the bundles of $AC(2i+1) \circ D \circ S \circ AC(2i) \circ L \circ S(X)$ becomes:

$$\begin{array}{llll} \cdots\, S(w^i) & \cdots\, S(w^2) \oplus S(z^2) \oplus S(z^2 \oplus c) & S(w^1) \oplus S(z^1 \oplus c) \oplus S(z^1) & S(w^0 \oplus c) \oplus c' \\ \cdots\, S(w^i) \oplus S(z^i) & \cdots\, S(w^2) \oplus S(z^2 \oplus c) & S(w^1) \oplus S(z^1) \oplus c' & S(w^0 \oplus c) \oplus S(z^0) \\ \cdots\, S(w^i) \oplus S(z^i) & \cdots\, S(w^2) \oplus S(z^2) \oplus c' & S(w^1) \oplus S(z^1 \oplus c) & S(w^0 \oplus c) \oplus S(z^0). \end{array}$$

Thus, to ensure a 2-identical state, the following 2 equations need to be satisfied:

$$S(z^2 \oplus c) = S(z^2) \oplus c', \quad S(z^1 \oplus c) = S(z^1) \oplus c' \ .$$

We can then compute the probability of following each step of the truncated pattern in Figure 5.7 starting from the end of step 2:

Step 3: each equation is satisfied with probability 2^{-3} for one state, thus 2^{-12} in total for the two states.

Step 4: the probability for one state becomes 2^{-2}, meaning 2^{-8} in total.

Step 5: the 2-identical property cannot be carried through because of the round constants. However, one can obtain a difference in the form $(0, *, *)$ between the two states with probability $2^{-4.83}$, as explained below.

By inverting the D layer of step 6, we should then observe a difference equal to 0 in the first bundle with a probability equal to $(2^{-2.415})^2 \times 2^{-8} \times 2^{-12} = 2^{-24.83}$.

Let us now compute the probability of going through step 5. If we denote (w, z, z) and (w, z', z') the two states after the application of S and L in step 5, then the expression of the column values at the end of that step for (w, z, z) becomes

$$\begin{array}{llll} \cdots\, S(w^i) & \cdots\, S(w^2) \oplus S(z^2) \oplus S(z^2 \oplus c) & S(w^1) \oplus S(z^1 \oplus c) \oplus S(z^1) & S(w^0 \oplus c) \oplus c' \\ \cdots\, S(w^i) \oplus S(z^i) & \cdots\, S(w^2) \oplus S(z^2 \oplus c) & S(w^1) \oplus S(z^1) \oplus c' & S(w^0 \oplus c) \oplus S(z^0) \\ \cdots\, S(w^i) \oplus S(z^i) & \cdots\, S(w^2) \oplus S(z^2) \oplus c' & S(w^1) \oplus S(z^1 \oplus c) & S(w^0 \oplus c) \oplus S(z^0) \end{array}$$

and it takes the following value for (w, z', z')

$$\begin{array}{llllll} \cdots & S(w^i) & \cdots & S(w^2) \oplus S(z'^2) \oplus S(z'^2 \oplus c) & S(w^1) \oplus S(z'^1 \oplus c) \oplus S(z'^1) & S(w^0 \oplus c) \oplus c' \\ \cdots & S(w^i) \oplus S(z'^i) & \cdots & S(w^2) \oplus S(z'^2 \oplus c) & S(w^1) \oplus S(z'^1) \oplus c' & S(w^0 \oplus c) \oplus S(z'^0) \\ \cdots & S(w^i) \oplus S(z'^i) & \cdots & S(w^2) \oplus S(z'^2) \oplus c' & S(w^1) \oplus S(z'^1 \oplus c) & S(w^0 \oplus c) \oplus S(z'^0) \end{array}$$

For the first bundles to be equal for both states the following relations have to be satisfied:

$$\begin{aligned} S(z'^2) \oplus S(z'^2 \oplus c) &= S(z^2) \oplus S(z^2 \oplus c) \\ S(z'^1) \oplus S(z'^1 \oplus c) &= S(z^1) \oplus S(z^1 \oplus c) \ , \end{aligned}$$

which occurs with probability $2^{-2.415}$ for each relation.

Experimental Results. Experiments showed that the probability of the distinguisher is very close to what we expected. By testing 2^{30} pairs, we obtained 31 successes, a probability close to 2^{-25}. Our unoptimized `C++` implementation took less than 70 minutes on a desktop computer to find these pairs, i.e. about 2 minutes per pair on average. Below is an example.

x_1 $\qquad\qquad\qquad\qquad$ x_2

```
62544d56 60b9af6e bd3ddabf    62544d56 48d12ffc bb391a7d
019d0421 569ad0d3 e543b03f    019d0421 5efa911b cb4ff1e5
5f8ba283 087f4892 f7b632d8    5f8ba283 265b098a ffd272c0
116bb908 eef0b58d 97dc955a    116bb908 e0d8f55f 9790d5da
```

$x_1 \oplus x_2$ $\qquad\qquad\qquad\qquad$ $\pi(x_1) \oplus \pi(x_2)$

```
00000000 28688092 0604c0c2    00000000 d7ddf0cd 87ed7095
00000000 086041c8 2e0c41da    00000000 c4e25bec df225a5c
00000000 2e244118 08644018    00000000 d3d67ba2 9416fab2
00000000 0e2840d2 004c4080    00000000 1cff6fdd 9cf7ed09
```

Difference with 7 Rounds. The main difference with our 7-round distinguisher on `Shadow-512` is our inability to cover step 0, and it stems from D. The middle rounds of the attack cannot be moved, as they depend on our ability to cancel out the constants and maintain 2-identical states. In the 7-round attack, rounds alternate between 3 and 1 active Super S-box (bundle). The inverse step 1 takes in input a difference $\alpha, \alpha, \alpha, 0$ and the inverse application of D gives a difference $0, 0, 0, \alpha$. But since this difference is active in only one bundle, we can traverse one more round and have a difference active in only three bundles. Here, we used a different path, with two active bundles at each round. The inverse step 1 takes as input a difference $0, \alpha, \alpha$, the inverse of D maps to a difference $0, \alpha, \alpha$, but after the inverse Super S-box, we obtain two unknown differences, and we cannot traverse round 0.

5.5 Forgeries with 4-step `Shadow` in the Nonce Misuse Setting

In this section, we show how to use the properties exploited in the distinguishers to create existential forgeries for the S1P mode of operation [BBB+19], in the single user setting, when used with 4-step `Shadow` (out of 6) shifted of two steps (starting at step 2 instead of 0). Hence, our attack targets the "aggressive parameters" specified in [BBB+19, Section 5].

One interesting feature of `Spook` is that it provides strong integrity guarantees in the presence of nonce misuse and leakage, which are formalized as CIML2 in the unbounded leakage model [BPPS17]. In our attack, we do not require leakage and instead exploit the nonce control. More specifically, we require the same nonce to be used three times. Our attack then creates two different messages with the same authentication tag. In particular, we are able to build collisions on the underlying hash function, which allows us to build the forgeries.

Attack Outline. S1P is a sponge-based mode of authenticated encryption with associated data represented in Figure 5.8, that uses `Shadow` as its underlying permutation. It has a rate of size 256 bits and a capacity of size 256 bits. If we number the bundles of `Shadow` as in the reference implementation, bundles 0 and 1 are the rate part and bundles 2 and 3 are the capacity part.

For the sake of simplicity, we consider a version of the S1P mode of operation without associated data, and we only consider two-block messages M_0, M_1. This situation is depicted on Figure 5.8, where π is the `Shadow` permutation, `Initialize` is a procedure combining π and the Clyde block cipher, that produces a 512-bit state from a nonce N and the secret key K, and `Finalize` is a procedure that, on input a 512-bit state, produces a 128-bit authentication tag.

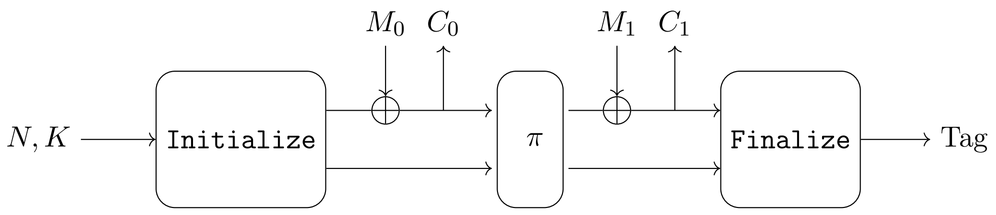


Figure 5.8: S1P mode in our attack setting

Our goal is to output two plaintexts $(M_0, M_1), (M'_0, M'_1)$ and a nonce N that yield the same authentication tag. In order to do that, we obtain a collision on the internal state before `Finalize`. This means that any pair $(M_0, M_1, x_2, ..., x_\ell)$, $(M'_0, M'_1, x_2, ..., x_\ell)$ of messages built by appending the same blocks to our colliding pair would also yield the same tag provided that the nonce is reused. We can find (M_0, M_1) and (M'_0, M'_1) thanks to the following algorithm, that we will prove later.

Let π be the `Shadow` permutation restricted to rounds 2 to 5. Informally, the first queries allow us to find the difference between the states before π, the second ones to figure out the difference after π, and the third to cancel it out. The whole attack is presented in details in Algorithm 16. Before describing it, we present its main subroutine whose success probability is given by the following lemma.

Algorithm 15 Algorithm to generate candidate pairs for our 4-step property.

Output: two pairs of $(x_1, y_1), (x'_1, y'_1)$ such that $\pi(x_1, y_1, a, b) \oplus \pi(x'_1, y'_1, a, b) = (*, *, 0, 0)$ with probability p.

```
1. Select a random 128-bit bundle $w_2$.
2. Invert step 2 on $(w_2, w_2, 0, 0)$, obtaining $(x_1, y_1, *, *)$
3. Return $(x_1, y_1), (x_1 \oplus \epsilon, y_1 \oplus \epsilon)$ where $\epsilon \in \nabla$ (a difference that intervenes only in columns
   22 and 23 of a bundle).
```

Lemma 5.4. *Let* $(*, *, a, b)$ *be a* `Shadow` *state. Then Algorithm 15 produces 4 bundles* $(x_1, y_1), (x'_1, y'_1)$ *such that* $\pi(x_1, y_1, a, b) \oplus \pi(x'_1, y'_1, a, b) = (*, *, 0, 0)$ *with a probability* $p \simeq 2^{-24.83}$.

In a nutshell, this property allows us to find a collision on the capacity part of the state after having applied π. Since we can control the differences in the rate before and after π, we then obtain a collision on the full 512-bit state. This is summarized in Algorithm 16. Notice that each plaintext and ciphertext "block" is comprised of two rate bundles.

4-step Path. We will now prove Lemma 5.4. We are interested in pairs of 2-identical states for `Shadow`, where the first two bundles are equal. The following lemma stems immediately from the results in Table 5.5 (as both states in the pair must remain 2-identical, we take the squared probabilities).

Algorithm 16 Collision attack on the S1P mode, with nonce reuse, and using 4-step `Shadow`.

1. Encrypt an arbitrary two-block (4-bundle) message, *e.g.* $(0,0),(0,0)$, and obtain ciphertexts $(d_0,d_1),(d_2,d_3)$. Let x_1,y_1,a,b be the 4-bundle state after `Initialize` (immediately before step 1). Then $d_0,d_1=x_1,y_1$.
2. Use Algorithm 15 to obtain two pairs of rate bundles $(x'_1,y'_1),(x''_1,y''_1)$ such that $\pi(x'_1,y'_1,a,b)\oplus\pi(x''_1,y''_1,a,b)=(*,*,0,0)$ with probability p.
3. Encrypt (with the same nonce) $(x_1\oplus x'_1,y_1\oplus y'_1),(0,0)$ and obtain $(c'_0,c'_1),(c'_2,c'_3)$. Then (c'_2,c'_3) is the value of the rate after the application of π on (x'_1,y'_1,a,b).
4. Encrypt (with the same nonce) $(x_1\oplus x''_1,y_1\oplus y''_1),(0,0)$ and obtain $(c''_0,c''_1),(c''_2,c''_3)$. Then (c''_2,c''_3) is the value of the rate after the application of π on (x''_1,y''_1,a,b).
5. Output the two 4-bundle plaintexts: $(x_1\oplus x'_1,y_1\oplus y'_1),(c'_2,c'_3)$ and $(x_1\oplus x''_1,y_1\oplus y''_1),(c''_2,c''_3)$ and the nonce N that was used. Then these plaintexts, encrypted with this nonce, yield the same internal state before the `Finalize` procedure, and the same tag, with probability $p\simeq 2^{-24.83}$.

Lemma 5.5. *Let t_1,t_2 be a pair of 2-identical states with difference $(\alpha,\alpha,0,0)$. Then after a step of* `Shadow`*, they remain 2-identical with probability 2^{-12} at step 3, 2^{-8} at step 4 and 0 otherwise.*

Using these probabilities, we can investigate Lemma 5.4.

Proof. We follow the convention of indexing bundles depending on the step that immediately precedes, *i.e.* w_2 is a bundle after step 2. The pattern used in this proof is summarized in Figure 5.9.

We consider `Shadow` reduced to steps 2 to 5. We start with an input state $(*,*,a_1,b_1)$. We select a random bundle w_2 and invert step 2 on $(w_2,w_2,0,0)$. We denote by $(x_1,y_1,*,*)$ the state obtained after this inversion.

Now we consider the "mixed" state (x_1,y_1,a_1,b_1) passing through step 2. Applying the super-sboxes, D and adding the second round constant we obtain the state: $(\sigma_2(a_1)\oplus\sigma_3(b_1)\oplus w_2,\sigma_2(a_1)\oplus\sigma_3(b_1)\oplus w_2,*,*)$ which is 2-identical.

We also consider the state $(x_1\oplus\epsilon,y_1\oplus\epsilon,a_1,b_1)$, where ϵ belongs to the set ∇ of 2^{16} differences that modify only the columns 22 and 23 of a bundle. Then, as shown in our analysis from the previous section, the difference does not interact with the part of the state dealing with the round constant. Hence, the state after step 2 is also 2-identical and it has the same bundles in the capacity part.

We now have a pair of 2-identical states (x_2,x_2,a_2,b_2) and (y_2,y_2,a_2,b_2) in the output of step 2. It is mapped to a pair of 2-identical states (x_3,x_3,a_3,b_3) and (y_3,y_3,a_3,b_3) through step 3 with probability 2^{-12} and similarly through step 4 with probability 2^{-8}.

Before step 5, we have a 2-identical pair $(x_4,x_4,a_4,b_4),(y_4,y_4,a_4,b_4)$. Our goal is to obtain a zero-difference in the capacity part. By an analysis analogous to the one of `Shadow`-384, this happens with a probability approximately equal to $2^{-4.83}$. We then have a total probability of $\underbrace{1}_{\text{Step 2}}\times\underbrace{2^{-12}}_{\text{Step 3}}\times\underbrace{2^{-8}}_{\text{Step 4}}\times\underbrace{2^{-4.83}}_{\text{Step 5}}$. □

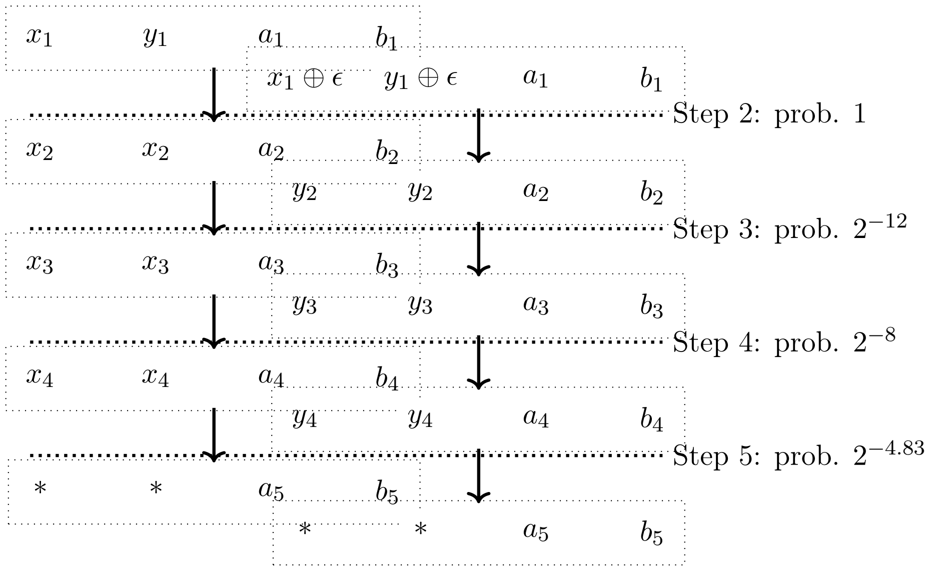


Figure 5.9: 4-step path

Experimental Results. Lemmas 5.4 and 5.5 have been verified independently. Furthermore, we have fully implemented the attack against S1P itself. Using the reference implementation of S1P (but taking out the two first steps) we obtained the following example for a zero-key and a zero-nonce.

$m_1 =$	aaf2fbf5334fdfc6c1ee182f593cc6e1	a5ebc70be994a1bc8b980410a3dae96a
	e93257859683265f20552e381b15c621	eb3257859783265f23552e381b15c621
$m_2 =$	aaf2bbf5334fdfc6c1ee182f593cc6e1	a5eb870be994a1bc8b980410a3dae96a
	2f160415c118c8c174200434e93c2e83	2d160415c018c8c177200434e93c2e83
$c_1 =$	75235998b09dcbe55a97db04e29622e4	4e73577cdacccc3520d6d6b03b5f2f51
	00000000000000000000000000000000	00000000000000000000000000000000
	c6461d8861f434500882ac5dc3490ce1	
$c_2 =$	75231998b09dcbe55a97db04e29622e4	4e73177cdacccc3520d6d6b03b5f2f51
	00000000000000000000000000000000	00000000000000000000000000000000
	c6461d8861f434500882ac5dc3490ce1	

After 2^{30} trials, we obtained 41 successful collisions, with an experimental probability of success of $2^{-24.64}$ which backs the theoretical $2^{-24.83}$. In practice, our un-optimized `C++` implementation needs about 15 minutes to find one collision.

Possible Extensions. Although using similar properties as the previous distinguishers (keeping 2-identical states with a cancellation of constants, using a difference in ∇), our attack suffers from the fact that we cannot control the input in the capacity part. This is the main reason why we cannot consider the steps before step 2, contrary to our distinguisher on full `Shadow-512`.

As a trivial extension, we remark that we can extend our reduced-step `Shadow` by one round (*i.e.* half a step) at the end of our 4-step path, since this round does not traverse D; but it falls outside the scope of the actual primitive. We could attack rounds 4 to 13 of `Shadow-512` instead of rounds 0 to 12.

Furthermore, the differences that we obtain at the input of step 2 are very sparse, since they belong to the space ∇. As the complexity of our attack is of the order of 2^{25}, and the generic complexity is of 2^{128} for a collision on the capacity, it might be possible to extend the attack 1 round at the beginning `Shadow`, but this seems far from trivial and would require advanced message modification techniques.

5.6 Conclusion

In this chapter we have shown some new cryptanalysis results on the second round candidate of the lightweight NIST competition `Spook` based on the limited birthday problem. We can distinguish 5-step `Shadow-512` from a random permutation using only 2 queries. If we exploit the round constants, we are able to distinguish the full (6-step) `Shadow-512`, and we could even distinguish 7 steps if the number of rounds was increased (and regardless of the round constant values chosen for this step). Regarding `Shadow-384`, we are able to efficiently distinguish the 6-step permutation if its round constants are shifted, and a round-reduced 5-step version otherwise.

Using similar ideas we could build collisions on the underlying hash function for a 4-step version of the permutation, which means we can build forgeries for the S1P mode with nonce misuse, which is allowed by the CIML2 security game considered by the authors [BPPS17].

All the analyses presented are practical and have been implemented and verified. The corresponding source code is publicly available.

An interesting extension of this work would be to reach 5-step forgeries: as we presented, extending it one round is easy, but one more round for reaching 5-steps might be possible using some advanced message modification techniques. In any case, 6-steps do seem out of reach with our current techniques.

New criterion. Our analysis provides a new simple criterion for choosing the round constants in LS-designs: besides trying to avoid invariant subspaces attacks, they should be introduced in such a way that their effect in the internal symmetries cannot be canceled out.

Tweaks for `Shadow`. Though our findings do not represent a threat on the full-round authenticated encryption primitive, the `Shadow` permutation can be tweaked to counter the low complexity distinguisher and improve the security margin of `Spook`.

The first tweak we suggested the designers was to use denser constants. This change would not affect the 5-step distinguishers, but would counter the 6-step ones and the 4-step forgeries. A perhaps more interesting alternative due to implementation reasons was to use one round constant per step instead of two, in order to prevent us from building identical bundle states by canceling out the constants inside a step.

A second tweak option was to change the D matrix in order to break the symmetry properties between the bundles. The designers chose the latter in order to improve the security bounds of `Spook`. They replaced the binary D operation of `Shadow` with an efficient MDS matrix proposed in [DL18] that no longer preserves the equality of two bundles and additionally improves diffusion. This limits the symmetry properties between the `Shadow` bundles and increases the bound on the number of active S-Boxes for differential and linear characteristics. To compensate for the performance overhead generated by this MDS D-box, the constant addition was changed. This resulted in a more efficient processing as well as denser constants. These two changes were combined into `Spook v2`, presented in [BBB+20].

6

Differential Analysis of SKINNY with Different Tools

Come on, skinny love, just last the year.
Pour a little salt, we were never here.

BON IVER

Evaluating resistance of ciphers against differential cryptanalysis is essential to define the number of rounds of new designs and to mount attacks derived from differential cryptanalysis. This part of the thesis covers some results obtained in this area together with Stéphanie Delaune, Patrick Derbez, Marine Minier, Victor Mollimard and Charles Prud'homme. More precisely, this chapter presents a comparison between existing automatic tools—previously introduced in Chapter 3, Section 3.4—to find the best differential characteristics on the `SKINNY` block cipher. As usually done in the literature, we split this search in two stages denoted by Step 1 and Step 2. In Step 1, each difference variable is abstracted with a Boolean variable and we search for the value that minimizes the trail weight, whereas Step 2 tries to instantiate each difference value while maximizing the overall differential characteristic probability. We model Step 1 using a MILP tool, a SAT tool, an ad-hoc method and a CP tool based on the Choco-solver library and provide performance results. Step 2 is modeled using the Choco-solver as it seems to outperform all previous methods on this stage.

Notably, for `SKINNY`-128 in the **SK** model and for 13 rounds, we retrieve the results of Abdelkhalek *et al.* [AST$^+$17] within a few seconds—while it took 16 days in [AST$^+$17]—and we provide, for the first time, the best differential related-tweakey characteristics up to respectively 14 and 12 rounds for the **TK1** and **TK2** models. These results have been submitted and are currently under review.

6.1 Introduction

Differential cryptanalysis [BS91b] evaluates the propagation of an input difference $\delta X = X \oplus X'$ between two plaintexts X and X' through the ciphering process. Indeed, differential attacks exploit the fact that the probability of observing a specific output difference given a specific input difference is not uniformly distributed. Today, differential cryptanalysis is public knowledge, and block ciphers such as the AES have proven bounds against differential attacks. A classical extension of differential cryptanalysis is the so called related-key differential cryptanalysis [Bih94] that allows an attacker to inject differences not only between the plaintexts X and X' but also between the keys K and K' (even if the secret key K stays unknown from the attacker). This attack has recently been extended to tweakable block ciphers [BJK+16]. As already mentioned in Chapter 2, those particular ciphers allow in addition to the key, a public value called a tweak. Thus, related-tweakey differential attacks allow related-key differences but also related-tweak differences (*i.e.* differences in a pair of tweaks (T, T')). In differential attacks, two notions are considered: first, differentials, where only the input and the output differences are known; and differential characteristics, where each difference after each round is completely specified. A classical approach to evaluate the resistance against differential attacks is to compute the probability of the best differential characteristic of the cipher.

Finding optimal (related-tweakey) differential characteristics is a highly combinatorial problem that hardly scales. To limit this explosion, a common solution consists in using a truncated representation [Knu95] for which cells are abstracted by single bits that indicate whether sequences contain differences or not. Typically, each cell (*i.e.* byte or nibble) is abstracted by a single bit (or, equivalently, a Boolean value). In this case, the goal is no longer to find the exact input and output differences, but rather to find the positions of these differences, *i.e.*, the presence or absence of a difference for every cell. When a difference is present at the input of an S-box, we talk about an active S-box or an active byte/nibble. However, some truncated representations may not be valid (*i.e.*, there do not exist actual byte values corresponding to these difference positions) because some constraints at the byte level are relaxed when reasoning on difference positions.

Hence, the optimal (related-tweakey) differential characteristic problem is usually solved in two steps [BN10, AST+17]. In the first one, every differential byte is abstracted by a Boolean variable that indicates whether there is a difference or not at this position, and we search for all truncated representations of low weight as the less differences passing through S-boxes there are, the more the probability is increased. Then, for each of these low weight truncated representations, the second step aims at deciding whether it is valid (*i.e.*, whether it is possible to find actual cell values for every Boolean variable) and, if it is valid, at finding the actual cell values that maximize the probability of obtaining the output difference given the input difference.

Many techniques have been proposed to search for the Step 1 solutions using automatic tools such as Mixed Integer Linear Programming (MILP) [SHW+14, ST17, BJK+16], Boolean satisfiability (SAT) [SNC09, MP13, SWW17] and Satisfiability Modulo Theories (SMT) [KLT15]. Dedicated solutions have also been proposed [Mat95]. However, only few tools have been proposed to find the best instantiation of a truncated characteristic [AST+17, Laf18, SWW18, FJP13, BN10, GLMS18, ENP19]. Notably, in [AST+17], authors introduce a MILP model of the non-linear part of a block cipher and present some results on SKINNY-n where the time required to find differential paths is about 16 days.

Our contribution. In this chapter, we compare several methods that implement Step 1 resolution on the SKINNY-n tweakable block cipher. Four attack models can be considered on SKINNY-n according the size of the tweakey: the **SK** model focuses on single-key attack, the **TK1** model considers related-tweakey attack when the tweakey has only one component, the **TK2** model in the related-tweakey settings considers 2 components and the **TK3** model, 3 components.

We first implement Step 1 using 4 different tools: a MILP model, a SAT model, an Ad-Hoc method and a CP model for the 4 attack settings. We also propose a CP model for Step 2 taking as inputs the solutions output by Step 1. We analyze and compare all the proposed methods through intensive computations dedicated to the SKINNY case. As a result we show that MILP is not always the best choice for both problems. First, for Step 1, the Ad-Hoc method is able to surpass the MILP model. Second, the CP model proposed for Step 2 is incomparably much faster than the MILP model proposed in [AST+17], reducing the execution time from several days to few minutes. Thus, we provide, for the first time, the best differential related-tweakey characteristics up to 14 rounds for the **TK1** model and up to 12 rounds for the **TK1** model of SKINNY-128. This is an important improvement compared to previous results. For instance, in [LGS17] Liu *et al.* could only find the best differential characteristics up to 7 and 9 rounds respectively. Finally we also show there is no differential characteristic with probability higher than 2^{-128} for 15 rounds in the **TK1** model and provide the best **TK2** related-tweakey differential characteristic we found for 16 rounds. All those results clearly show that SKINNY is much more resistant to differential cryptanalysis than one would expect when simply counting the number of active S-boxes.

All the codes for those models will be made public once the reviewing process of our paper is finished. These codes are meant to be easy to use and to adapt to other ciphers.

6.2 Cipher Under Study: SKINNY-n

In this section, we briefly review the tweakable block cipher SKINNY-n where n denotes the block size and can be equal to 64 or 128 bits. All the details that have been overlooked can be found in [BJK+16].

As its name indicates, it enciphers blocks of length 64 or 128 bits seen as a 4×4 matrix of cells (nibbles for $n = 64$ or bytes for $n = 128$). We denote $x_{i,j,k}$ the cell at row i and column j of the internal state at the beginning of round k (*i.e.* $0 \le i, j \le 3$ and $0 \le k \le r+1$, where r is the number of rounds depending on the tweak length and on the key length). SKINNY-n follows the TWEAKEY framework from [JNP14]. SKINNY-n has three main tweakey size versions: the tweakey size can be equal to $t = 64$ or 128 bits, $t = 128$ or 256 bits and $t = 192$ or 384 bits and we denote $z = t/n$ the tweakey size to block size ratio. Then, the number of rounds is directly derived from the z value: from 32 rounds for the 64/64 version up to 56 for the 128/384 version.

The tweakey state is also viewed as a collection of z 4×4 square arrays of cells (nibbles for $n = 64$ or bytes for $n = 128$). We denote these arrays $TK1$ when $z = 1$, $TK1$ and $TK2$ when $z = 2$, and finally $TK1$, $TK2$ and $TK3$ when $z = 3$. We also denote by $TKk_{i,j}$ the nibble or the byte at position $[i, j]$ in TKk. Moreover, we define the associated adversarial model **SK** (resp. **TK1**, **TK2** or **TK3**) where the attacker cannot (resp. can) introduce differences in the tweakey state.

One encryption round of `SKINNY` is composed of five operations applied in the following order: `SubCells (SC)`, `AddConstants (AC)`, `AddRoundTweakey (ART)`, `ShiftRows (SR)` and `MixColumns (MC)` (see Fig. 6.1).

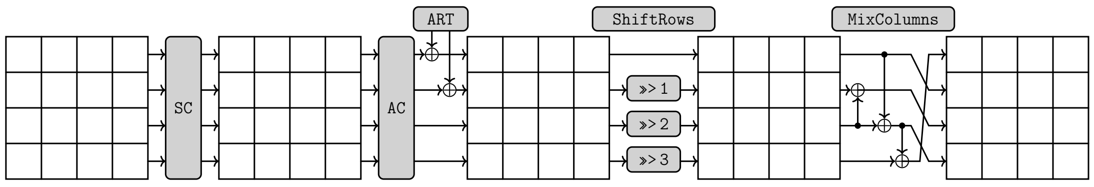


Figure 6.1: the `SKINNY` round function with its five transformations [Jea16].

`SubCells.` A 4-bit ($n = 64$) or an 8-bit ($n = 128$) S-box is applied to each cell of the state. See [BJK+16] for the details of the S-boxes.

`AddConstants.` A 6-bit affine LFSR is used to generate round constants c_0 and c_1 that are XORed to the state at position $[0, 0]$ and $[1, 0]$ whereas the constant $c_2 =$ `0x02` is XORed to the position $[2, 0]$.

`AddRoundTweakey.` The first and second rows of all tweakey arrays are extracted and bitwise exclusive-ored to the cipher internal state, respecting the array positioning. More formally, we have:

- $x_{i,j} = x_{i,j} \oplus TK1_{i,j}$ when $z = 1$,
- $x_{i,j} = x_{i,j} \oplus TK1_{i,j} \oplus TK2_{i,j}$ when $z = 2$,
- $x_{i,j} = x_{i,j} \oplus TK1_{i,j} \oplus TK2_{i,j} \oplus TK3_{i,j}$ when $z = 3$.

Then, the tweakey arrays are updated. First, a permutation P_T is applied on the cells positions of all tweakey arrays: if $\ell = 4*i+j$ where i is the row index and j is the column index, then the cell ℓ is moved to position $P_T(\ell)$ where $P_T = [9, 15, 8, 13, 10, 14, 12, 11, 0, 1, 2, 3, 4, 5, 6, 7]$. Second, every cell of the first and second rows of $TK2$ and $TK3$ are individually updated with an LFSR on 4 bits (when $n = 64$) or on 8 bits (when $n = 128$) with a period equal to 15.

`ShiftRows.` The rows of the cipher state cell array are rotated to the right. More precisely, the second (resp. third and fourth) cell row is rotated by 1 position (resp. 2 and 3 positions).

`MixColumns.` Each column of the cipher internal state array is multiplied by the 4×4 binary matrix M:

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

6.3 Overview of Solving Techniques

In this section, we briefly recall the different techniques previously introduced in Section 3.4 and that we used for performing the search of the best differential characteristic. Note that the Ad-Hoc method inspired from [FJP13] is standalone and will be only described in the next Section.

6.3.1 Mixed Integer Linear Programming

Many symmetric cryptanalysis problems on different ciphers have been tackled with MILP [SHW$^+$14, BJK$^+$16, ST17, MWGP11]. Note that MILP traditionally considers variables from discrete domains and from continuous domains. Here and as usually done in all the cryptanalytic contexts, we only consider integer variables, and instead, we should rather talk about ILP—for Integer Linear Programming—as the term Mixed designates continuous variables. However, since MILP is the term classically used in the cryptographic community, we decided to stick to this terminology.

The important point is that MILP models can only contain linear inequalities. Therefore, it is necessary to transform non-linear operators into sets of linear inequalities. Moreover, as done in [BJK$^+$16], we decided to use the Gurobi Mathematical Optimization solver [Opt18]. To be compatible with our code in Python 3 and to benefit from the the search options on the pool of solutions, a version greater than 9 is required.

6.3.2 Constraint Programming

Although less usual than MILP to tackle cryptanalytic problems, CP has already been used in *e.g.* [GMS16, ENP19]. We recall some basic principles of CP and we refer the reader to [RBW06] for more details.

CP is used to solve Constraint Satisfaction Problems (CSPs). A CSP is defined by a triple (X, D, C) such that $X = \{x_1, x_2, \ldots, x_n\}$ is a finite set of variables, D is a function that maps every variable $x_i \in X$ to its domain $D(x_i)$ and $C = \{c_1, c_2, \ldots, c_m\}$ is a set of constraints. $D(x_i)$ is a finite ordered set of integer values to which the variable x_i can be assigned to, whereas c_j defines a relation between some variables $vars(c_j) \subseteq X$. This relation restricts the set of values that may be assigned simultaneously to $vars(c_j)$. Each constraint is equipped with a filtering algorithm which removes from the domains of $vars(c_j)$, the values that cannot satisfy c_j.

In CP, constraints are classified in two categories. *Extensional constraints*, also called *table constraints*, explicitly define the allowed (or forbidden) tuples of the relation. *Intentional constraints* define the relation using mathematical operators. For instance, in a CSP with $X = \{x_1, x_2, x_3\}$ such that $D(x_1) = D(x_2) = D(x_3) = \{0, 1\}$, a constraint ensuring that the sum of the variables in X is different from 1 can be either expressed in extension (1) or in intention (2):

1. $\textsc{Table}(\langle x_1, x_2, x_3\rangle, \langle(0,0,0),(0,1,1),(1,0,1),(1,1,0),(1,1,1)\rangle)$
2. $x_1 + x_2 + x_3 \neq 1$

Actually, any intentional constraint can be encoded with an extensional one provided enough memory space, and conversely [DHL$^+$16]. However, they may offer different performances.

The purpose of a CSP is to find a *solution*, *i.e.* an assignment of all variables to a value from their respective domains such that all the constraints are simultaneously satisfied. When looking for a solution, a two-phase mechanism is operated: the *search space exploration* and the *constraint propagation*. The exploration of the search space is processed using a *depth-first search*. At each step, a decision is taken, *i.e.* a non-assigned variable is selected and its domain is reduced to a singleton. This modification requires checking the satisfiability of all the constraints. This is achieved thanks to constraint propagation which applies each constraint filtering algorithm. Any application may trigger modifications in turn; the propagation ends when either no modification occurs and all constraints are satisfied or a failure is thrown, *i.e.*, at least one constraint cannot be satisfied. In the former case, if all variables are assigned, a solution has been found. Otherwise a new decision is taken and the search is pursued. In the latter case, a backtrack to the first refutable decision is made and the search is resumed.

Turning a CSP into a Constrained Optimisation Problem (COP) is done by adding an objective function. Such a function is defined over variables of X, the purpose is then to find the solution that optimizes the objective function. Finding the optimal solution is done by repeatedly applying the two-phase mechanism above, and by adding a *cut* on the objective function that prevents from finding a same cost solution in the future.

6.3.3 SAT

Transforming cryptanalytic problems into a propositional Boolean logic formula is also a common technique [MP13, SWW17, KLT15, SNC09, SWW18]. To ease the modeling step, a high-level modeling language called MiniZinc [NSB+07] has been used: MiniZinc models are translated into a simple subset of MiniZinc called FlatZinc, using a compiler provided by MiniZinc, and supported by most existing CP solvers that have developed FlatZinc interfaces (currently, there are fifteen CP solvers which have FlatZinc interfaces). Evaluations select Picat-SAT as the best candidate SAT solver for Step 1. Picat-SAT translates CSPs into Boolean satisfiability formulae, and then uses the SAT solver Lingeling [Bie14] to solve it. Since Picat-SAT clearly outperforms all the other SAT solvers provided through the MiniZinc interface, we decided to discard the other ones when comparing with other techniques in Section 6.6.

6.4 Models for Step 1

As explained in the Introduction, in a first step called Step 1, we abstract each possible difference at byte level by a binary variable which symbolizes the presence/absence of a difference value at a given position of the cipher. The main concern regarding this step is the combinatorial explosion induced by the abstract XOR operation for which the sum of two non zero values can lead to the presence or the absence of a difference.

Note also that all the models described below are tuned to enumerate the solutions for a given number of active S-boxes and for a given number of rounds in the four possible attack models. We call this step *Step1-enum*. This phase comes after an initial step called *Step1-opt*, in which the minimal number of active S-boxes for a given number of rounds has been already found. Note also that all the models for **SK** discard symmetries up to column shift.

6.4.1 MILP Models

A MILP model has already been proposed in [BJK+16], but for comparison purposes on time benchmarks, and to better fit our needs, we re-implement it. Below, we only describe our modifications and refer to Appendix D in [BJK+16] for the original model.

First, we add constraints in the **SK** model to obtain all solutions up to column shifts in order to remove symmetries. Moreover, as the original model only describes the way to find the minimal number of active S-boxes, we add a constraint in each model to set a lower bound

Minimize

$$Obj_{Step1} = \sum_{r=1}^{n} \sum_{i=1}^{4} \sum_{j=1}^{4} \delta X_{r,i,j} \tag{6.1}$$

subject to

$$\begin{aligned} &\textsc{Table}(\langle \delta X_{r,0,j}, \delta X_{r,1,(j+3)\%4}, \delta X_{r,2,(j+2)\%4}, \delta X_{r,3,(j+1)\%4},\\ &\delta X_{r+1,0,j}, \delta X_{r+1,1,j}, \delta X_{r+1,2,j}, \delta X_{r+1,3,j}\rangle\\ &, \langle \text{MxC}\rangle), \Delta X_{r,i,j} \neq 0, \forall r \in 1..n-1, \forall j \in 1..4 \end{aligned} \tag{6.2}$$

where $\forall r \in 1..n$, $\forall i \in 1..4$, $\forall j \in 1..4$,

$$\delta X_{r,i,j} \in 0..1$$

and $\langle \text{MxC}\rangle$ encodes both `MixColumns` and `Shiftrows` constraints.

Model 1: Formulation of **SK** Step 1, without symmetry breaking constraints.

on the number of active S-boxes and thus, be able to enumerate all the Step 1 solutions given a particular lower bound for the number of active S-boxes. In MILP, the constraint for XOR $\mathcal{C}_{\oplus}[i_1, i_2, o, d]$ needs a dummy variable d to be correctly modeled. $\mathcal{C}_{\oplus}[i_1, i_2, o, d]$ is the set of the following linear constraints:

$$\{i_1 \leq d\} \cup \{i_2 \leq d\} \cup \{o \leq d\} \cup \{i_1 + i_2 + o \leq 2d\}.$$

Finally, regarding the resolutions of the MILP models, the parallelization is left to the Gurobi solver[1].

6.4.2 MiniZinc (SAT) Models

Due to the high-level modeling allowed by MiniZinc, the model is exactly the same as in the MILP model described in Subsection 6.4.1 except the way we model the XOR operation. Indeed, we simply use the method described in [GL16] where if a, b and c are Boolean variables, then the XOR operation verifies (considering addition over integers): $i_1 + i_2 + o \neq 1$. Thus, this model does not require the dummy variables d.

6.4.3 CP Models

A CP model for **SK** is depicted in Model 1. In comparison to other models (Subsection 6.4.1 and Subsection 6.4.2), the objective function (6.1) remains identical. Then, the model relies on TABLE constraints (6.2) with $\langle$MxC$\rangle$ as input parameter, the list of feasible combinations. In an early stage, $\langle$MxC$\rangle$ is computed based on a composition of `MixColumns` relation and `ShifRows` relation over two blocks and eight variables. Only the 34 combinations satisfying both `MixColumns` and `ShifRows` are retained among the 256 possible ones. Just like MILP and SAT, symmetry breaking constraints are added to models in order to prevent the calculation of solutions equivalent up to column shift.

TK1, **TK2** and **TK3** are modeled based on Model 1, *i.e.*, without the symmetry breaking constraints. We follow the same lines as the MILP model proposed in Appendix D in [BJK+16] to model cancellation in **TK2** and **TK3**.

In terms of solving configuration, a parallel portfolio is used to run resolutions simultaneously. We separate the different models according to the 2^{16} possible values (0 or 1 for each possible cell) taken in a given middle round, which turns the original COP into many CSPs. In practice, each independent sub-problem is assigned to a new thread. Threads send each other messages

[1]see: https://www.gurobi.com/documentation/9.0/refman/threads.html for more details.

containing the value of Obj_{Step1}, the best number of S-boxes found so far. Such an approach limits the number of messages passed but offers valuable data to running threads, and future ones. Indeed, bound sharing prevents from exploring sub-regions of the search where there is no chance to find better solutions.

In addition, each model defines a search strategy based on a lexicographic ordering. Once the middle round being instantiated, then, it goes one round by one round to the forward and to the backward direction.

6.4.4 Ad-Hoc Models

To the best of our knowledge, the most efficient algorithm to search for truncated representations is the one described in [FJP13] by Fouque *et al.*. The main idea is that round i is independent of the paths of rounds $0, 1, \dots, i-1$ and at each step we only have to save, for each truncated state, the minimal number of active S-boxes to reach it. Hence, the complexity of this algorithm is exponential in the state size but linear in the number of rounds. The algorithm is specified in Algorithm 17. At the end of the algorithm we obtain an array C such that $C[r][s]$ contains the minimal number of active sboxes required to reach state s after r rounds. Retrieving the truncated representations is then done quite easily using C, starting from the last state to the first.

Algorithm 17 Search for the best truncated representation (**SK**).

```
foreach state $s$ do
    $M[s] \longleftarrow$ list of states $s'$ reachable from $s$ through one round
end
foreach state $s$ do
    $C[0][s] \longleftarrow$ number of active cells of $s$
end
for $1 \leq r < R$ do
    foreach state $s$ do $C[r][s] \longleftarrow \infty$   foreach state $s$ do
        foreach state $s'$ in $M[s]$ do
            $c \longleftarrow C[r-1][s] +$ number of active cells of $s'$   if $c < C[r][s']$ then $C[r][s'] \longleftarrow c$
        end
    end
end
return $C$
```

The complexity of the algorithm in the single key model is very low, and we experimentally counted around $(R-1) \times 2^{20}$ simple operations for R rounds. A naive solution to search for truncated representations in the **TK1**, **TK2** and **TK3** models would be to apply the previous algorithm for each possible configuration of the key. While for **TK1** this would only increase the overall complexity by a factor 2^{16}, the search would not be practical for both the **TK2** and **TK3** models. Indeed, because of the possible cancellations occurring in the round keys, the number of configurations is very high:

$$\left(\sum_{k=0}^{8} \binom{8}{k} \left(\sum_{i=0}^{tk-1} \binom{\lfloor (R-1)/2 \rfloor}{i}\right)^k\right) \left(\sum_{k=0}^{8} \binom{8}{k} \left(\sum_{i=0}^{tk-1} \binom{\lceil (R-1)/2 \rceil}{i}\right)^k\right).$$

For instance, for $R = 30$, there are more than 2^{64} configurations in the **TK2** model.

In the following we present the first practical algorithm which tackles down the problem without relying on a black box solver as MILP, SAT or CP solvers. The idea is quite similar to the one used in the single key model. Actually, to compute the minimal number of active S-boxes at round $r+1$ we only need to know the minimal number of active S-boxes for each possible state at round r together with the number of cancellations for each key cell. Indeed, we do not need to know at which rounds the cancellations occurred but only how many times they did. A simplified version of this algorithm is described in Algorithm 18. In practice, we found it was better to proceed step by step. First we pick a key cell and guess whether it is active or not. Then we apply the algorithm partially and guess another key cell if and only if it seems possible to find a better representation.

Remarks. Note that while our ad-hoc tool gave the best running times, it may require a lot of memory to store the array C. For instance, for 30 rounds in **TK3** mode, our tool required up to 500 GB of RAM to finish the search. It is also important to note that it did not fully take advantage of the 128 cores of our server, and most often used less than 40 cores.

Algorithm 18 Search for the best truncated representation (**TK**).

```
foreach state $s$, round key $k$ do
    $M[k][s] \longleftarrow$ list of states $s'$ reachable from $s$ and $k$ through one round
end
foreach state $s$ do
    $C[0][s] \longleftarrow \{(\text{number of active cells of } s, 0)\}$
end
for $1 \leq r < R$ do
    foreach state $s$ do $C[r][s] \longleftarrow \emptyset$   foreach state $s$, round key $k$ do
        foreach state $s'$ in $M[k][s]$ do
            foreach $(cost, cancelled) \in C[r-1][s]$ do
                if cancelled compatible with $k$ then
                    $c \longleftarrow$ cost + number of active cells of $s'$   $C[r][s'] \longleftarrow C[r][s'] \cup$
                    $\{(c, \text{update}(cancelled, k))\}$
                end
            end
        end
    end
    foreach state $s$ do keepOptimals($C[r][s]$)
end
return $C$
```

6.5 Modeling Step 2 with CP

The aim of Step 2 is to try to instantiate the abstracted solutions provided by Step 1 while maximizing the probability of the differential characteristic. Thus, Step 2 takes as input a solution of Step 1 with the objective function of maximizing the probability of the differential characteristic. However, some solutions of Step 1 could not be instantiated in Step 2 as refining the abstraction level of Step 2 will induce *non-consistent* solutions. In the literature, this Step has been modeled using Ad-Hoc methods [BN10], MILP [AST$^+$17], SAT [SWW18] or CP [GLMS20]. As MILP [AST$^+$17] and SAT [SWW18] seem to hardly scale due to prohibitive computational times (linked with the size of the 8-bit S-boxes that must be represented in the form of linear inequalities or of clauses), we focus here on a dedicated CP method implemented using the Choco solver [PFL16].

Given a Boolean solution for Step 1, Step 2 aims at searching for the byte-consistent solution with the highest (related-tweakey) differential characteristic probability (or proving that there is no byte-consistent solution). In this section, Model 2 describes the CP model we used for `SKINNY-128` (**SK**). Actually, the ones used to model the other variants, as well as `SKINNY-64`, are rather similar.

Minimize

$$Obj_{Step2} = \sum_{r=1}^{n}\sum_{i=1}^{4}\sum_{j=1}^{4} P_{r,i,j} \tag{6.3}$$

subject to

$$20 \times n \leq \sum_{r=1}^{n}\sum_{i=1}^{4}\sum_{j=1}^{4} P_{r,i,j} \leq \min(70 \times n, O^*) \tag{6.4}$$

$$\forall r \in 1..n, \forall i \in 1..4, \forall j \in 1..4$$
$$\begin{cases} \delta X_{r,i,j} = 0 \wedge \delta SB_{r,i,j} = 0 \wedge P_{r,i,j} = 0 & \text{if } \Delta X_{r,i,j} = 0 \\ \delta X_{r,i,j} \geq 1 \wedge \delta SB_{r,i,j} \geq 1 \wedge P_{r,i,j} \geq 20 & \text{otherwise} \end{cases} \tag{6.5}$$

$$\forall r \in 1..n, \forall i \in 1..4, \forall j \in 1..4$$
$$\textsc{Table}(\langle \delta X_{r,i,j}, \delta SB_{r,i,j}, P_{r,i,j}\rangle, \langle \text{SBox}\rangle) \quad \text{if } \Delta X_{r,i,j} \neq 0 \tag{6.6}$$

$$\forall r \in 1..n-1, \forall j \in 1..4 \qquad \delta SB_{r,0,j} = \delta X_{r+1,1,j} \tag{6.7}$$

$$\forall r \in 1..n-1, \forall j \in 1..4$$
$$\begin{cases} \delta SB_{r,2,(2+j)\%4} = \delta X_{r+1,2,j} & \text{if } \Delta SB_{r,1,(3+j)\%4} = 0 \\ \delta SB_{r,1,(3+j)\%4} = \delta X_{r+1,2,j} & \text{if } \Delta SB_{r,2,(2+j)\%4} = 0 \\ \delta SB_{r,1,(3+j)\%4} = \delta SB_{r,2,(2+j)\%4} & \text{if } \Delta X_{r+1,2,j} = 0 \\ \textsc{Table}(\langle \delta SB_{r,1,(3+j)\%4}, \delta SB_{r,2,(2+j)\%4}, \delta X_{r+1,2,j}\rangle, \langle \text{XOR}\rangle) & \text{otherwise} \end{cases} \tag{6.8}$$

$$\forall r \in 1..n-1, \forall j \in 1..4$$
$$\begin{cases} \delta SB_{r,2,(2+j)\%4} = \delta X_{r+1,3,j} & \text{if } \Delta SB_{r,0,j} = 0 \\ \delta SB_{r,0,j} = \delta X_{r+1,3,j} & \text{if } \Delta SB_{r,2,(2+j)\%4} = 0 \\ \delta SB_{r,0,j} = \delta SB_{r,2,(2+j)\%4} & \text{if } \Delta X_{r+1,3,j} = 0 \\ \textsc{Table}(\langle \delta SB_{r,0,j}, \delta SB_{r,2,(2+j)\%4}, \delta X_{r+1,3,j}\rangle, \langle \text{XOR}\rangle) & \text{otherwise} \end{cases} \tag{6.9}$$

$$\forall r \in 1..n-1, \forall j \in 1..4$$
$$\begin{cases} \delta X_{r+1,0,j} = \delta X_{r+1,3,j} & \text{if } \Delta SB_{r,3,(1+j)\%4} = 0 \\ \delta SB_{r,3,(1+j)\%4} = \delta X_{r+1,3,j} & \text{if } \Delta X_{r+1,0,j} = 0 \\ \delta SB_{r,3,(1+j)\%4} = \delta X_{r+1,0,j} & \text{if } \Delta X_{r+1,3,j} = 0 \\ \textsc{Table}(\langle \delta SB_{r,3,(1+j)\%4}, \delta X_{r+1,0,j}, \delta X_{r+1,3,j}\rangle, \langle \text{XOR}\rangle) & \text{otherwise} \end{cases} \tag{6.10}$$

where $\forall r \in 1..n$, $\forall i \in 1..4$, $\forall j \in 1..4$,

$$\delta X_{r,i,j} \in 0..255, \ \delta SB_{r,i,j} \in 0..255, \ P_{r,i,j} \in \{0, 20, .., 70\},$$

and $\langle$XOR$\rangle$ encodes $\oplus$ relation and $\langle$SBox$\rangle$ the S-box constraint.

Model 2: Formulation of **SK** Step2.

For each Boolean variable $\Delta X_{r,i,j}$ of Step 1, we define an integer variable $\delta X_{r,i,j}$. The domain of this integer variable depends on the value of the Boolean variable in the Step 1 solution: if

$\Delta X_{r,i,j} = 0$, then the domain is $D(\delta X_{r,i,j}) = \{0\}$ (*i.e.*, $\delta X_{r,i,j}$ is also assigned to 0); otherwise, the domain is $D(\delta X_{r,i,j}) = [1, 255]$ (6.5).

For each byte that passes through an S-box, we define an integer variable $\delta SB_{r,i,j}$ which corresponds to the difference after the S-box. Its domain is $D(\delta SB_{r,i,j}) = \{0\}$ if $\Delta X_{r,i,j}$ is assigned to 0 in the Step 1 solution; Otherwise, it is $D(\delta SB_{r,i,j}) = [1, 255]$ (6.5).

Finally, as we look for a byte-consistent solution with maximal probability, we also add an integer variable $P_{r,i,j}$ for each byte in an S-Box: this variable corresponds to the absolute value of the base 2 logarithm of the probability of the transition through the S-Box. Actually, a factor 10 has been applied to avoid considering floats. Thus we define a TABLE constraint (6.6) composed of valid triplets of the form $(\delta X_{r,i,j}, \delta SB_{r,i,j}, P_{r,i,j})$. Note that these triplets only contain non-zero values and that $P_{r,i,j}$ only takes 2 different values for the 4-bit S-box (`SKINNY`-64) and 7 different values for the 8-bit S-box (`SKINNY`-128). There are roughly 2^{14} triplet elements in the TABLE constraint for the `SKINNY`-128 case. As the S-box layer is the only non-linear layer, the other operations could be directly implemented in a deterministic way at the cell level. The associated constraints thus follow the `SKINNY`-128 linear operations. When possible, we replace XOR constraints (encoded using TABLE constraints) by a simple equality constraint. This corresponds to TABLE constraints (6.7), (6.8), (6.9) and (6.10) in Model 2.

The overall goal is finally to find a byte-consistent solution which maximizes differential characteristic probability. Thus, we define an integer variable Obj_{Step2} to minimize the sum of all $P_{r,i,j}$ variables (6.3). This value mainly depends on the number of S-boxes output by Step1 Obj_{Step1} and can be bounded to $[\![20 \cdot Obj_{Step1}, 70 \cdot Obj_{Step1}]\!]$ (6.4).

The differences for the models **TK1**, **TK2** and **TK3** are the modeling of the XORs induced by the lanes of the tweakey through XOR table constraints. Each XOR constraint depicted in Model 2 provides high quality filtering but requires 65536 tuples to be stored which results in prohibitive memory usage. This may limit the number of threads that can be used for the resolution, which was the case for **TK2**. To get around this issue, we encoded the XOR constraint in intention (by defining filtering rules), providing a more memory efficient algorithm, at the expense of filtering strength. This last choice was applied only for **TK2** (`SKINNY`-128 only). We also rely on TABLE constraints to model the LFSR applied on TK2 and TK3.

Concerning the search space strategy, for the **TK2** and the **TK3** attack settings, Step 1 only outputs the truncated value of the sum of the TKi. Thus, the search space strategy first looks at the cancellation places of the sum of the TKi and then instantiates the TKi values according to those positions. For the **TK1** setting, we just apply the default Choco-solver strategy.

Regarding parallelization, we affect one solution output by Step 1 per thread and we share the value of Obj_{Step2} between the threads.

6.6 Results

Regarding Step 1, we run our different tools on the four attack scenarios (**SK**, **TK1**, **TK2**, and **TK3**). Then, Step 2 is performed on the two versions of `SKINNY` (`SKINNY`-64 and `SKINNY`-128) using our CP models written in Choco-solver.

We conduct all our experiments on our server composed of 2×AMD EPYC 7742 64-Core and 1TB of RAM. All the reported times are `real` system times and then take advantage of tools that are properly designed for parallelism. We first detail here the time results obtained for the different tools modeling Step 1 and then move to the Step 2 time results.

6.6.1 Step 1 strategies comparison

In this Subsection, we compare the time results obtained by all the Step 1 tools using the function *Step1-enum*. *Step1-enum* comes after a first process called *Step1-opt* that searches for a solution that optimizes the value of the variable Obj_{Step1} whereas *Step1-enum* enumerates all solutions when the variable Obj_{Step1} is assigned to a given value, here the optimal one v^* where v^* corresponds to the minimal number of active S-boxes. This optimal value is of course the same for the different models as the abstractions made in the different models are the same.

Results for *Step1-opt.*

Finding the minimal number of active S-boxes on a given number of rounds is most often the only result which is interesting for designers as it allows to derive a lower bound on the probability of the best differential trail. However, showing that the minimal number of active S-boxes is n is actually similar to enumerating all characteristics with at most $n-1$ active S-boxes and finding no solution. Thus, the running times required by each tool to find the minimal number of active S-boxes are very close to the running times reported on Table 6.2. In **SK**, **TK1** and **TK2** our ad-hoc tool gives the best running times by far, while in **TK3**, our MILP model is also competitive. In particular, we are able to complete the security analysis made in [BJK+16, ABI+18] and claim that the minimal number of active S-boxes in **TK1** for 28, 29 and 30 rounds are 105, 109 and 113 respectively (as shown in Table 6.1).

# Rounds	28	29	30
TK1	105	109	113

Table 6.1: Lower bounds on the number of active S-boxes in SKINNY.

Results for *Step1-enum.*

Table 6.2 reports the different `real` times we obtained to enumerate all the solutions for the optimal value $Obj_{Step1} = v^*$ (*i.e.* with v^* active S-boxes). Those computations are done on our server for the 4 different Step 1 tools (MILP, MiniZinc/SAT, Ad-Hoc, Choco-solver), for the different attack scenarios (**SK**, **TK1**, **TK2** and **TK3**) on SKINNY when varying the number of rounds between 3 and 20. The first column specifies the number of solutions we found for the Obj_{Step1} value. Those solutions correspond to solutions given without the symmetries, thus are computed up to the columns shifts (for **SK**). As one can see, these numbers are really low and hide a different reality when Obj_{Step1} increases. Indeed, the optimal solution of Step 2 in terms of differential characteristic probability, could be obtained for a value v of Obj_{Step1} which is not optimal ($v > v^*$). For example, imagine that when processing Step 2, one obtains a differential characteristic with the best probability equal to $2^{-3 \cdot 6} = 2^{-18}$ with $Obj_{Step1} = 6$ and whereas the optimal differential probability of the S-box is 2^{-2}. It means that one has to test all solutions output by Step 1 until $Obj_{Step1} = 18/2 = 9$ to be sure that none has a better differential characteristic probability. This is exactly what happened for the case of SKINNY-128 in the **TK1** model for 15 rounds as we will detail in the next Section. We only want to stress here that computing the optimal bounds is often not enough and we need to go further. However, increasing the value of Obj_{Step1} induces an increase of the possible number of Step 1 solutions as illustrated in the third column of Table 6.8. As one can see, this number of solutions tends to grow exponentially as v increases. For example, for SKINNY-128 with 14 rounds in the **TK1** model, for the optimal value $v^* = 45$, Step 1 outputs only 3 solutions; whereas we have 897 solutions for $v = v^* + 5 = 50$; 137 019 solutions for $v = v^* + 10 = 55$ and finally 7 241 601 solutions for $v = 59$. So, the time required for the Step 2 computations on 1 solution output by Step 1 becomes the bottleneck of the overall process.

Analyzing the results and understanding the tools.

Table 6.2 reports the time required to enumerate all the solutions with v^* active S-boxes where v^* is the optimal value of Obj_{Step1} found by *Step1-opt*. This value and the corresponding number of solutions is reported in the second column of Table 6.2.

SAT is clearly disadvantaged by the choice we made *i.e.* using a high level modeling language: MiniZinc. Indeed, SAT could not perform clause learning as the constraint `SolveAll` is not implemented from MiniZinc to SAT. Thus, once a solution for Step 1 has been found, the program has to be rerun by adding a constraint that discards this valid Step 1 solution. This is why MiniZinc performs less efficiently than one could expect. The previous fact could be directly seen on Table 6.2 as the time required to solve instances with many solutions is much bigger than the one required to solve instances with only few Step 1 solutions.

Choco-solver does not seem to be a good candidate either as for all instances greater than 16 rounds it requires more than 24 hours to solve the problem. This is mainly linked with the nature of the variables themselves. Choco-solver (and more generally CP) is efficient when domains are subset of integers. Here, Choco-solver can not efficiently propagate lower bounds and upper bounds on Boolean variables as MILP or SAT could do.

Actually, the Step 1 model could be completely linearized and of course, the branch-and-cut method used by MILP eliminates uninteresting branches quite quickly. SAT behaves better than CP because the problem is purely Boolean and CP/Choco-solver does not benefit from the conflict-driven clause learning (the CDCL). Thus, CP, that performs very well on integer domains, is less well suited when regarding Boolean or linear problems.

Moreover, regarding the way CP cuts the problem, this division produces mainly trivial problems, the few remaining ones have a very large search space (therefore, those are the ones that should be cut). But once more, MILP and SAT perform better.

Contrary to the two previous tools, MILP and the Ad-Hoc method seem to perform well and, as shown in Table 6.2, could solve all the problems on all the instances in reasonable times. As shown in the previous paragraph, the Ad-Hoc method is able to outperform MILP. For MILP, and as previously said, this is clearly linked with the nature of the problem to solve: Step 1 only models Boolean variables for which values propagate very well in the MILP model. The Ad-Hoc method is finally a dedicated one and manages to surpass even MILP.

Thus, in conclusion, we think that if one does not have the time to think of a solution, MILP is a good candidate to quickly have direct Step 1 bounds. If one has time, the Ad-Hoc method clearly outperform previous results. We also think that SAT could perform well even if it is not proven here. The idea behind is to directly use a SAT solver without any other interfaces to be closer to the clauses.

#Rounds	*Step1-enum*																			
	Obj_{Step1} (Nb sols)				MILP				MiniZinc/SAT				Ad-Hoc				Choco			
	SK	**TK1**	**TK2**	**TK3**	**SK**	**TK1**	**TK2**	**TK3**	**SK**	**TK1**	**TK2**	**TK3**	**SK**	**TK1**	**TK2**	**TK3**	**SK**	**TK1**	**TK2**	**TK3**
3	5 (4)	1 (12)	0 (1)	0 (1)	1s	1s	-	-	1s	1s	-	-	1s	21s	69s	69s	7s	1s	-	-
4	8 (3)	2 (9)	0 (1)	0 (1)	1s	1s	-	-	4s	1s	-	-	1s	22s	25s	76s	7s	1s	-	-
5	12 (2)	3 (2)	1 (12)	0 (1)	1s	1s	1s	-	4s	1s	1s	-	1s	21s	22s	103s	7s	1s	1s	-
6	16 (1)	6 (2)	2 (10)	0 (1)	1s	1s	1s	-	6s	1s	1s	-	1s	22s	22s	25s	7s	3s	1s	-
7	26 (4)	10 (2)	3 (2)	1 (12)	1s	1s	1s	1s	17s	8s	1s	1s	1s	21s	22s	22s	7s	7s	1s	1s
8	36 (17)	13 (1)	6 (2)	2 (11)	1s	1s	1s	1s	140s	7s	4s	2s	1s	22s	31s	23s	7s	8s	3s	1s
9	41 (2)	16 (1)	9 (1)	3 (3)	2s	2s	2s	2s	57s	11s	7s	1s	1s	22s	24s	26s	8s	9s	7s	1s
10	46 (2)	23 (1)	12 (2)	6 (3)	7s	5s	3s	2s	97s	46s	15s	10s	1s	22s	24s	27s	9s	60s	55s	2s
11	51 (2)	32 (2)	16 (1)	10 (3)	8s	11s	4s	3s	312s	29m	22s	24s	1s	23s	25s	32s	23s	188m	86s	34s
12	55 (2)	38 (7)	21 (1)	13 (2)	13s	35s	7s	3s	468s	> 24h	113s	35s	1s	24s	27s	25s	75s	> 24h	43m	288s
13	58 (6)	41 (2)	25 (2)	16 (2)	9s	53s	17s	6s	14m		14m	104s	1s	24s	30s	27s	249s		> 24h	56m
14	61 (2)	45 (3)	31 (1)	19 (1)	23s	93s	27s	8s	491s		72m	148s	1s	24s	39s	28s	10m			> 24h
15	66 (2)	49 (1)	35 (1)	24 (4)	69s	245s	75s	21s	27m		> 24h	157m	1s	25s	46s	34s	85m			
16	75 (8)	54 (1)	40 (2)	27 (1)	12m	423s	148s	39s	128m			251m	1s	25s	57s	38s	> 24h			
17	82 (4)	59 (5)	43 (1)	31 (2)	46m	22m	213s	53s	106m			> 24h	1s	27s	59s	48s				
18	88 (4)	62 (1)	47 (1)	35 (1)	178m	31m	535s	64s	403m				1s	27s	76s	73s				
19	92 (4)	66 (1)	52 (1)	43 (14)	529m	56m	29m	218s	436m				1s	28s	110s	283s				
20	96 (2)	70 (2)	57 (2)	45 (2)	16h	87m	33m	340s	174m				1s	28s	193s	326s				

Table 6.2: Comparison of the times of the different Step 1 tools for solving $Step\ 1 - enum$ (`SKINNY`), *i.e.* to enumerate all solutions for the optimal Onj_{Step1} bound given in the first column in each scenario: **SK**, **TK1**, **TK2** and **TK3**. We report the `real` time on our server.

6.6.2 Step 2 performance results

Up to our knowledge, we only found [AST+17] that gives time results concerning finding the best **SK** differential characteristic probability on SKINNY-128 using a MILP tool based on Gurobi. More precisely, the authors say:

> *"In our experiments, we used Gurobi Optimizer with Xeon Processor E5-2699 (18 cores) in 128 GB RAM."*

and, for 13 rounds in the **SK** model:

> *"In our environment, the test of 6 classes [Step 1 solutions with 58 active S-boxes without symmetries] finished in 16 days [. . .] Finally, it is proven that the tight bound on the probability of differential characteristic for 13 rounds is* 2^{-123}*."*

Concerning the use of SAT, [SWW18] implements a SAT model for differential cryptanalysis based on `Cryptominisat5` [SNC09] for `Midori64` and `LED64`. This model implies a sufficiently small number of clauses to model the non-zero values of the DDT and to be applicable. However, no result concerning 8-bit S-boxes are given. As SAT uses Boolean formulas, it seems that the same problem than for MILP appears for modeling S-box: a huge number of Boolean formulas will be necessary to correctly model this step even if dedicated tools as Logic Friday or the Expresso algorithm [AST+17] are used. Thus, we discard the use of a SAT model.

Results for SKINNY-64.

We sum up in Table 6.3 all the results we obtain for SKINNY-64 in the four different attack models (**SK**,**TK1**,**TK2** and **TK3**). The overall time, in this case, is not a bottleneck. We only give results concerning number of rounds that are at the limit (just under and just upper) when considering the number of active S-boxes, which is equal to 32 in the case of SKINNY-64, as the state size is 64 bits and the best differential probability of the S-box is equal to 2^{-2}. Thus, the best overall differential characteristic probability must be under 2^{-64}.

Note that sometimes, we need to browse several Obj_{Step1} bounds to find the optimal differential characteristic probability when the number of rounds is fixed. Indeed, we need to proactively adapt the probability bound we found. For example, in the case of **TK2** SKINNY-64 with 13 rounds, the optimal Obj_{Step1} is equal to 25 and when providing the Step 2 process with this Obj_{Step1} bound, we find a best differential characteristic probability equal to 2^{-55}. Thus, we need to run again *Step1-enum* with $Obj_{Step1} = 26$ and $Obj_{Step1} = 27$ to be sure that the previous probability is really the best one. Then, before running again Step 2 on those new results we adapt the best probability to the new bound equal to 2^{-55} instead of the old bound equal to 2^{-64}.

	Nb Rounds	Obj_{Step1}	Nb sol. Step 1	Step 2 time	Best Pr
SK	7	26	2	1s	2^{-52}
SK	8	36	17	1s	$< 2^{-64}$
TK1	10	23	1	1s	2^{-46}
TK1	11	32	2	1s	$= 2^{-64}$
TK2	13	$25 \to 27$	10	1s	2^{-55}
TK2	14	31	1	1s	$< 2^{-64}$
TK3	15	$24 \to 26$	46	2s	2^{-54}
TK3	16	$27 \to 31$	87	4s	$= 2^{-64}$
TK3	17	31	2	1s	$< 2^{-64}$

Table 6.3: Overall results concerning `SKINNY`-64 in the four attack models. Step 2 time corresponds to the Step 2 time taken over all solutions of *Step1-enum* when Obj_{step1} takes the values detailed in the second column. Best Pr corresponds to the best probability of a differential characteristic found.

Best (related-tweakey) differential characteristics for `SKINNY-64`. In the following, we provide the details of the best differential characteristics we found for `SKINNY-64`. The best **SK** differential characteristic for 7 rounds with probability equal to 2^{-52} is given in Table 6.4. Then, in the **TK1** setting, the best differential characteristics for 10 rounds with probability equal to 2^{-46} is described in Table 6.5. For **TK2**, Table 6.6 details the best differential characteristic obtained for 13 rounds with probability equal to 2^{-55}. Finally, Table 6.7 describes the best **TK3** differential trail on 15 rounds of `SKINNY-64` with probability equal to 2^{-54} is given in Table 6.7.

Round	$\delta X_i = X_i \oplus X'_i$ (before `SB`)	δSBX_i (after `SB`)	Pr(States)
$i = 1$	`0040 4444 4440 4400`	`0020 2222 2220 2200`	$2^{-2 \cdot 10}$
2	`0000 0020 0200 2002`	`0000 0010 0100 1001`	$2^{-2 \cdot 4}$
3	`0010 0000 0000 0001`	`0080 0000 0000 0008`	$2^{-2 \cdot 2}$
4	`0000 0080 0000 0080`	`0000 0040 0000 0040`	$2^{-2 \cdot 2}$
5	`0400 0000 0004 0000`	`0200 0000 0002 0000`	$2^{-2 \cdot 2}$
6	`0000 0200 0200 0000`	`0000 0100 0100 0000`	$2^{-2 \cdot 2}$
7	`0001 0000 0011 0001`	`0008 0000 0088 0008`	$2^{-2 \cdot 4}$

Table 6.4: The best **SK** differential characteristic on 7 rounds of `SKINNY-64` with probability equal to 2^{-52}. The four words represent the four rows of the state and are given in hexadecimal notation.

Round	$\delta X_i = X_i \oplus X'_i$ (before `SB`)	δSBX_i (after `SB`)	$\delta TK1_i$	Pr(States)
$i = 1$	`0000 0002 0020 0200`	`0000 0001 0010 0100`	`1000 0000 0B80 0000`	$2^{-2 \cdot 3}$
2	`1000 1000 0000 0000`	`B000 8000 0000 0000`	`B000 8000 1000 0000`	$2^{-2 \cdot 2}$
3	`0000 0000 0000 0000`	`0000 0000 0000 0000`	`0010 0000 B000 8000`	–
4	`0010 0010 0000 0010`	`00B0 00A0 0000 00B0`	`00B0 0080 0010 0000`	$2^{-2 \cdot 3}$
5	`0B00 0000 0002 0000`	`0100 0000 0001 0000`	`0000 1000 00B0 0080`	$2^{-2 \cdot 2}$
6	`0000 0100 0000 0000`	`0000 0800 0000 0000`	`0000 B800 0000 1000`	$2^{-2 \cdot 1}$
7	`0000 0000 0B00 0000`	`0000 0000 0100 0000`	`0000 0010 0000 B800`	$2^{-2 \cdot 1}$
8	`0001 0000 0000 0001`	`0008 0000 0000 0008`	`0008 00B0 0000 0010`	$2^{-2 \cdot 2}$
9	`0080 0000 000B 0000`	`0040 0000 0001 0000`	`0000 0100 0008 00B0`	$2^{-2 \cdot 2}$
10	`0140 0040 0110 0140`	`0820 0020 0880 0820`	`0000 0B08 0000 0100`	$2^{-2 \cdot 7}$

Table 6.5: The best **TK1** differential characteristic on 10 rounds of `SKINNY-64` with probability equal to 2^{-46}. The four words represent the four rows of the state and are given in hexadecimal notation.

Round	$\delta X_i = X_i \oplus X'_i$ (before SB)	δSBX_i (after SB)	$\delta TK1_i$	$\delta TK2_i$	Pr(States)
$i = 1$	0000 8200 0080 0000	0000 4100 0040 0000	0000 0008 0502 0000	0000 000C 060C 0000	$2^{-2 \cdot 3}$
2	4000 0000 0410 4000	2000 0000 02A0 2000	5000 0002 0000 0008	D000 0008 0000 000C	$2^{-2 \cdot 4}$
3	0000 A000 0002 0002	0000 6000 0006 0003	0800 0000 5000 0002	0800 0000 D000 0008	$2^{-2 \cdot 3}$
4	0630 0000 0000 0600	03F0 0000 0000 0100	0250 0000 0800 0000	01A0 0000 0800 0000	$2^{-3 \cdot 3}$
5	1000 0000 0000 0000	9000 0000 0000 0000	8000 0000 0250 0000	1000 0000 01A0 0000	2^{-2}
6	0000 0000 0000 0000	0000 0000 0000 0000	2000 5000 8000 0000	2000 5000 1000 0000	–
7	0000 0000 0000 0000	0000 0000 0000 0000	0080 0000 2000 5000	0020 0000 2000 5000	–
8	00A0 00A0 0000 00A0	0060 0050 0000 0050	0020 0050 0080 0000	0040 00B0 0020 0000	$2^{-2 \cdot 3}$
9	0500 0000 000B 0000	0C00 0000 000C 0000	0000 8000 0020 0050	0000 4000 0040 00B0	$2^{-3 \cdot 2}$
10	0000 0C00 0000 0000	0000 0200 0000 0000	0000 2500 0000 8000	0000 9700 0000 4000	2^{-2}
11	0000 0000 0B00 0000	0000 0000 0100 0000	0000 0080 0000 2500	0000 0090 0000 9700	2^{-2}
12	0001 0000 0000 0001	000A 0000 0000 0008	0005 0020 0000 0080	000F 0030 0000 0090	$2^{-2 \cdot 2}$
13	0080 0000 0001 0000	0040 0000 0008 0000	0000 0800 0005 0020	0000 0300 000F 0030	$2^{-2 \cdot 2}$

Table 6.6: The best **TK2** differential characteristic on 13 rounds of SKINNY-64 with probability equal to 2^{-55}. The four words represent the four rows of the state and are given in hexadecimal notation.

Round	$\delta X_i = X_i \oplus X'_i$ (before SB)	δSBX_i (after SB)	$\delta TK1_i$	$\delta TK2_i$	$\delta TK3_i$	Pr(States)
$i = 1$	0000 0001 4000 0004	0000 0008 2000 0002	0000 080D 0000 0800	0000 0408 0000 0500	0000 0E0D 0000 0C00	$2^{-2 \cdot 3}$
2	0000 0000 0000 0020	0000 0000 0000 0010	0008 0000 0000 080D	000B 0000 0000 0408	000E 0000 0000 0E0D	2^{-2}
3	010D 000D 0000 000D	0A0E 0002 0000 0002	0D08 0000 0008 0000	0109 0000 000B 0000	060F 0000 000E 0000	$2^{-2 \cdot 3} 2^{-3}$
4	0020 0000 2000 0000	0030 0000 3000 0000	0000 0008 0D08 0000	0000 0007 0109 0000	0000 000F 060F 0000	$2^{-2 \cdot 2}$
5	0000 0030 0030 0000	0000 00C0 00C0 0000	D000 0008 0000 0008	2000 0003 0000 0007	3000 0007 0000 000F	$2^{-3 \cdot 2}$
6	0000 C000 000C 0000	0000 2000 0002 0000	0800 0000 D000 0008	0F00 0000 2000 0003	0700 0000 3000 0007	$2^{-2 \cdot 2}$
7	0200 0000 0000 0200	0500 0000 0000 0300	08D0 0000 0800 0000	0640 0000 0F00 0000	0B90 0000 0700 0000	$2^{-2 \cdot 2}$
8	3000 0000 0000 0000	D000 0000 0000 0000	8000 0000 08D0 0000	E000 0000 0640 0000	B000 0000 0B90 0000	2^{-3}
9	0000 0000 0000 0000	0000 0000 0000 0000	8000 D000 8000 0000	D000 9000 E000 0000	5000 4000 B000 0000	–
10	0000 0000 0000 0000	0000 0000 0000 0000	0080 0000 8000 D000	00C0 0000 D000 9000	0050 0000 5000 4000	–
11	0010 0010 0000 0010	0080 0090 0000 00A0	0080 00D0 0080 0000	00A0 0030 00C0 0000	00A0 0020 0050 0000	$2^{-2 \cdot 3}$
12	0A00 0000 0005 0000	0A00 0000 000A 0000	0000 8000 0080 00D0	0000 8000 00A0 0030	0000 A000 00A0 0020	$2^{-2} 2^{-3}$
13	0000 0A00 0000 0000	0000 0A00 0000 0000	0000 8D00 0000 8000	0000 5600 0000 8000	0000 D100 0000 A000	2^{-3}
14	0000 0000 0000 0000	0000 0000 0000 0000	0000 0080 0000 8D00	0000 0010 0000 5600	0000 00D0 0000 D100	–
15	0000 0000 0004 0000	0000 0000 0002 0000	000D 0080 0000 0080	000D 00B0 0000 0010	0008 0060 0000 00D0	2^{-2}

Table 6.7: The best **TK3** differential characteristic on 15 rounds of SKINNY-64 with probability equal to 2^{-54}. The four words represent the four rows of the state and are given in hexadecimal notation.

Results for SKINNY-128.

In the same way, we provide in Table 6.8 the best differential characteristic probability with the total time required for this search for the 4 different attack models. As one can see, we also verify all the possible values for Obj_{Step1} for a given number of rounds, depending on the probability value previously found. Thus, this time, the number of solutions output by Step 1 could be huge when we move away from the optimal Step 1 value v^*. However, as the time spent to solve one solution is reasonable, our model scales reasonably well: the worst case requires 25 days of `real` time on our server on 8 threads and 31 GB of RAM[2]. Our **TK2** model is based on XOR constraints encoded in intention (and not using tables) and these experiences have been launched using the 128 threads of our server. Table 6.8 shows the results obtained with the best configurations for **SK**, **TK** and **TK2**.

	Nb Rounds	Obj_{step1}	Nb sol. Step 1	Step 2 time	Best Pr
SK	9	$41 \to 43$	52	16s	2^{-86}
SK	10	$46 \to 48$	48	11s	2^{-96}
SK	11	$51 \to 52$	15	4s	2^{-104}
SK	12	$55 \to 56$	11	6s	2^{-112}
SK	13	$58 \to 61$	18	2m27s	2^{-123}
SK	14	$61 \to 63$	6	21s	$< 2^{-128}$
TK1	8	$13 \to 16$	14	4s	2^{-33}
TK1	9	$16 \to 20$	6	3s	2^{-41}
TK1	10	$23 \to 27$	6	4s	2^{-55}
TK1	11	$32 \to 36$	531	37s	2^{-74}
TK1	12	$38 \to 46$	186 482	213m	2^{-93}
TK1	13	$41 \to 53$	2 385 482	2 days	$2^{-106.2}$
TK1	14	$45 \to 59$	11 518 612	20 days	2^{-120}
TK1	15	$49 \to 63$	7 542 053	25 days	$< 2^{-128}$
TK2	9	$9 \to 10$	7	3s	2^{-20}
TK2	10	$12 \to 17$	132	11s	$2^{-34.4}$
TK2	11	$16 \to 25$	4203	6m	$2^{-51.4}$
TK2	12	$21 \to 35$	1 922 762	512m	$2^{-70.4}$
TK2	13	$25 \to 44$	-	not solved	$> 2^{-89.7}$
TK2	14	$31 \to 54$	-	not solved	$> 2^{-108.4}$
TK2	15	$35 \to 56$	-	not solved	$> 2^{-113.2}$
TK2	16	$40 \to 63$	-	not solved	$> 2^{-127.6}$
TK2	17	$43 \to 63$	-	not solved	-
TK2	18	$47 \to 63$	62 681 709	not solved	-
TK2	19	$52 \to 63$	772 163	280m	$< 2^{-128}$

Table 6.8: Overall results concerning SKINNY-128 in the four attack models. Step 2 time corresponds to the Step 2 time taken over all solutions of *Step1-enum* when Obj_{step1} takes the values precise in the first column. Best Pr corresponds to the best found probability of a differential characteristic.

Concerning **TK2**, for 18 rounds, we have not performed the computations due to the huge

[2]It seems that the use of the 128 threads was prohibited by the memory usage of XOR tables (i.e. XOR in extension).

number of Step 1 solutions. For the same reasons, the full computations for 15, 16 and 17 rounds have not been performed. Nonetheless, in the following, we provide the best **TK2** differential characteristic we found for 16 rounds. Note that we do not know if this differential characteristic is optimal in terms of probability as we do not test all the solutions from Step 1.

Best (related-tweakey) differential characteristics for SKINNY-128. Regarding the best **SK** differential characteristics on 13 rounds of SKINNY-128, we obtain the same trail with probability equal to 2^{-123} given in Table 11 of Appendix D of [AST+17]. The best **TK1** differential characteristic on 14 rounds of SKINNY-128 with probability equal to 2^{-120} is given in Table 6.9. The best **TK2** differential characteristic on 16 rounds of SKINNY-128 that we found is given in Table 6.10. The probability of this characteristic equals $2^{-127.6}$, but we do not know if this value is optimal since we did not test all Step 1 solutions.

Round	$\delta X_i = X_i \oplus X'_i$ (before `SB`)	δSBX_i (after `SB`)	$\delta TK1_i$	Pr(States)
$i = 1$	02000002 00000200 00020000 00020040	08000008 00000800 00080000 00080004	00000000 00000000 01000000 00000000	$2^{-2 \cdot 6}$
2	00000400 08000008 00000000 08000000	00000100 10000010 00000000 10000000	00000100 00000000 00000000 00000000	$2^{-2 \cdot 4}$
3	00000010 00000000 10100000 00000000	00000040 00000000 40400000 00000000	00000000 00000000 00000100 00000000	$2^{-2 \cdot 3}$
4	00004000 00000040 00004040 00004000	00000400 00000004 00000404 00000400	00000000 01000000 00000000 00000000	$2^{-2 \cdot 5}$
5	04000400 00000400 00050000 04040400	05000500 00000100 00050000 05050500	00000000 00000000 00000000 01000000	$2^{-3 \cdot 6} 2^{-2}$
6	00050500 05000500 00000004 05000505	00050500 01000100 00000005 05000505	00000000 00000100 00000000 00000000	$2^{-3 \cdot 6} 2^{-2 \cdot 2}$
7	00050005 00050500 00040000 00000500	00050005 00050500 00050000 00000500	00000000 00000000 00000000 00000100	$2^{-3 \cdot 6}$
8	00000000 00050005 00000500 00050000	00000000 00010005 00000500 00050000	00000000 00010000 00000000 00000000	$2^{-3 \cdot 3} 2^{-2}$
9	00000000 00000000 00000000 05000000	00000000 00000000 00000000 05000000	00000000 00000000 00000000 00010000	2^{-3}
10	00000005 00000000 00000000 00000000	00000001 00000000 00000000 00000000	00000001 00000000 00000000 00000000	2^{-2}
11	00000000 00000000 00000000 00000000	00000000 00000000 00000000 00000000	00000000 00000000 00000001 00000000	–
12	00000000 00000000 00000000 00000000	00000000 00000000 00000000 00000000	00000000 00000001 00000000 00000000	–
13	00000000 00000000 01000000 00000000	00000000 00000000 20000000 00000000	00000000 00000000 00000000 00000001	2^{-2}
14	00002000 00000000 00002000 00002000	00008000 00000000 00008000 00008000	00010000 00000000 00000000 00000000	$2^{-2 \cdot 3}$

Table 6.9: The best **TK1** differential characteristic on 14 rounds of `SKINNY-128` with probability equal to 2^{-120}. The four words represent the four rows of the state and are given in hexadecimal notation.

Round	$\delta X_i = X_i \oplus X'_i$ (before SB) δSBX_i (after SB)	$\delta TK1_i$ $\delta TK2_i$	Pr(States)
$i = 1$	00000000 00404010 40400000 40000000 00000000 00040440 04040000 04000000	00000000 00000000 00000000 00007700 00000000 00000000 00000000 00003900	$2^{-2 \cdot 6}$
2	00000400 00000000 40000000 00000404 00000500 00000000 04000000 00000101	00000000 00770000 00000000 00000000 00000000 00730000 00000000 00000000	$2^{-2 \cdot 3} 2^{-3}$
3	00010000 00000500 00000000 00000100 00200000 00000500 00000000 00002000	00000000 00000000 00000000 00770000 00000000 00000000 00000000 00730000	$2^{-2 \cdot 2} 2^{-3}$
4	00000000 00200000 00000005 00200000 00000000 00800000 00000005 00800000	00000077 00000000 00000000 00000000 000000E7 00000000 00000000 00000000	$2^{-2 \cdot 2} 2^{-3}$
5	80050090 00000090 00058000 00050090 03010002 00000002 00010200 00010003	00000000 00000000 00000077 00000000 00000000 00000000 000000E7 00000000	$2^{-2 \cdot 8}$
6	00010303 03010002 00000001 01010003 00202020 20200009 00000020 20200020	00000000 00000077 00000000 00000000 00000000 000000CE 00000000 00000000	$2^{-2 \cdot 6} 2^{-3 \cdot 4}$
7	20000000 00202020 B0002000 00002020 80000000 00808080 80008000 00009380	00000000 00000000 00000000 00000077 00000000 00000000 00000000 000000CE	$2^{-2 \cdot 6} 2^{-2.4} 2^{-3}$
8	00930000 80000000 00000080 00008000 00EA0000 03000000 00000003 00000300	00770000 00000000 00000000 00000000 009D0000 00000000 00000000 00000000	$2^{-2 \cdot 3} 2^{-6}$
9	00000000 00000000 00000000 00030000 00000000 00000000 00000000 00BC0000	00000000 00000000 00770000 00000000 00000000 00000000 009D0000 00000000	2^{-5}
10	BC000000 00000000 00000000 00000000 4C000000 00000000 00000000 00000000	77000000 00000000 00000000 00000000 3B000000 00000000 00000000 00000000	2^{-6}
11	00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000	00000000 00000000 77000000 00000000 00000000 00000000 3B000000 00000000	–
12	00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000	00007700 00000000 00000000 00000000 00007700 00000000 00000000 00000000	–
13	00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000	00000000 00000000 00007700 00000000 00000000 00000000 00007700 00000000	–
14	0000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000	00000000 77000000 00000000 00000000 00000000 EF000000 00000000 00000000	–
15	00000000 00000000 00980000 00000000 00000000 00000000 00420000 00000000	00000000 00000000 00000000 77000000 00000000 00000000 00000000 EF000000	2^{-5}
16	00000042 00000000 00000042 00000042 00000008 00000000 00000008 00000008	–	$2^{-2.4 \cdot 3}$

Table 6.10: The Best **TK2** differential characteristics we found on 16 rounds of SKINNY-128 with probability equal to $2^{-127.6}$. The four words represent the four rows of the state and are given in hexadecimal notation.

Lessons learnt.

The overall gap is not to find the optimal value of $Obj_{Step1} = v^*$ for a given number of rounds and to enumerate the corresponding overall solutions if the Step 1 model is sufficiently tight. The real gap is if the value obtained for Obj_{Step2} (here equal to $2 \times v^*$ as the best differential probability for the S-box is equal to 2^{-2}) is far from the optimal bound then we have to increase Obj_{Step1} up to the bound $\lfloor Obj_{Step2}/2 \rfloor$. The further we are from v^* in the Step 1 resolution, the more Step 1 solutions there are (in fact this number grows exponentially as can be seen in Table 6.8). Thus, the time for the Step 2 resolution becomes the bottleneck.

We have seen that CP is outperformed by MILP, SAT and Ad-Hoc methods when trying to model and solve the Step 1 problem. This is mainly linked with the nature of the problem where only Boolean variables are considered and where dedicated tools such as MILP or SAT perform very well in this case. Thus, no-one could think that it could be helpful for modeling Step 2. However, one of the main advantage of CP is the existing implementation of TABLE constraints that are well suited to the problem of modeling S-boxes and their DDT. Note that, in this case, especially when modeling 8-bit S-boxes, MILP and SAT imply a big number of equations that do not scale very well. Thus, CP could be a useful tool in this case.

One of the strenghts of CP—having TABLE constraints—can also become a weakness as their number and size increase. Our solution to code the XOR in intention is only possible when weaker filtering is compensated by a more constrained model containing the search space (like in **TK2**).

6.7 Conclusion

In this chapter, we have compared several tools searching for (related-tweakey) differential characteristics on the block cipher SKINNY. As usually done, we have divided the search procedure into two steps: Step 1 which abstracts the difference values into Boolean variables and finds the truncated characteristics with the smallest number of active S-boxes; and Step 2 which inputs the results of Step 1 to output the best possible probability instantiating the abstract solutions output by Step 1. Of course, each solution of Step 1 could not always be instantiated in Step 2.

This study shows that for Step 1, our ad-hoc tool which heavily uses the structure of the problem, consistently has the best running times. However, SAT is also quite good in **SK** and MILP runs well in both **TK2** and **TK3** settings. Furthermore, both the SAT and MILP models required much less work than our ad-hoc tool. Regarding Step 2, our Choco-solver model is much faster than any other approaches we tried. It allowed us to find, for the first time, the best (related-tweakey) differential characteristics in the **TK1** model up to 14 rounds for SKINNY-128 and to show that there is no differential trail on 15 rounds with a probability better than 2^{-128}. Regarding the **TK2** model, we were able to find the best differential trails up to 12 rounds. Note that in [LGS17] Liu *et al.* were only able to reach 7 and 9 rounds in the **TK1** and **TK2** model respectively. Our approach is thus an important improvement.

Part III

General Results on Feistel Constructions

7

Introducing the FBCT: A Cryptanalysis Tool for Feistel constructions

Le vrai boomerang, le boomerang authentique, pas celui que l'on trouve dans les bazars pour touristes de Sydney et de Camberra ! Le boomerang primitif, celui qui armait les fiers guerriers australiens depuis la nuit des temps, ce boomerang là a une dynamique, un mouvement dans l'espace quand il est bien lancé, que je qualifierais de miraculeux.

TANNER

At Eurocrypt 2018, Cid *et al.* introduced the Boomerang Connectivity Table (BCT), a tool to compute the probability of the middle round of a boomerang distinguisher from the description of the cipher's S-box(es). Their new table and the following works led to a refined understanding of boomerangs, and resulted in a series of improved attacks. Still, these works only addressed the case of Substitution Permutation Networks, and completely left out the case of ciphers following a Feistel construction. Together with Hamid Boukerrou, Virginie Lallemand, Bimal Mandal and Marine Minier, we addressed this lack by introducing the `FBCT`, the Feistel counterpart of the BCT. This chapter summarizes our results, which have been presented at FSE 2020 [BHL+20]. We show that the coefficient at row Δ_i, column ∇_o corresponds to the number of times the second order derivative at points (Δ_i, ∇_o) cancels out. We explore the properties of the `FBCT` and compare it to what is known on the BCT. Taking matters further, we show how to compute the probability of a boomerang switch over multiple rounds with a generic formula.

7.1 Introduction

Boomerang attacks date back to 1999, when David Wagner introduced them at FSE to break COCONUT98 [Wag99]. When presented, this variant of differential attacks [BS91a] shook up the conventional thinking that consisted in believing that a cipher with only small probability differentials is secure. Indeed, boomerang attacks make use of two small differentials covering half of the attacked rounds each, and can beat differential cryptanalysis when no high probability differential exists for the whole cipher.

In the basic form of the distinguisher, (represented on the left in Figure 7.1), the attacker has access to the encryption (E) and decryption (E^{-1}) oracles, and studies particular quartets of messages. First, she chooses M^1 and constructs $M^2 = M^1 \oplus \alpha$; using E, she obtains the corresponding ciphertexts C^1 and C^2 from which she deduces two additional ciphertexts by computing: $C^3 = C^1 \oplus \delta$ and $C^4 = C^2 \oplus \delta$. By calling the decryption oracle she retrieves the corresponding plaintexts M^3 and M^4 and then checks if $M^3 \oplus M^4 = \alpha$. A boomerang distinguisher is obtained if the probability that $M^3 \oplus M^4 = \alpha$ is higher for the cipher than for a random permutation.

In summary, a boomerang distinguisher is based on a couple of plaintext and ciphertext differences (α, δ) for which the following property among quartets of messages has a high probability:

$$E^{-1}(E(M^1) \oplus \delta) \oplus E^{-1}(E(M^1 \oplus \alpha) \oplus \delta) = \alpha.$$

In the original approach, the attacked cipher E is written as the composition of two sub-ciphers E_0 and E_1: $E = E_1 \circ E_0$. If for the sub-cipher E_0 the input difference α leads to the output difference β with probability p (and similarly γ leads to δ with probability q over E_1) the previous event was thought to have a probability of p^2q^2.

Following this breakthrough some variants were proposed including a related-key version [KKH+04, BDK05] and an impossible-differential one (see [CY09]). Improvements were also proposed on top of this, like a version that does not require access to the decryption oracle (named *amplified boomerang attack* [KKS01]) that was further developed into the so-called *rectangle attack* [BDK01].

The validity of boomerang attacks and in particular of the p^2q^2 formula were later questioned by Murphy [Mur11] with an example of distinguisher that seemed valid but was in fact of probability zero. The opposite phenomenon, that is distinguishers that happen with probability higher than what is expected, was also presented by Biryukov and Khovratovich in [BK09], and some particular cases (termed *Ladder Switch*, *S-box Switch* and *Feistel Switch*) were explained.

All these observations were later formalized in a framework called *sandwich attack* [DKS10] for which the cipher is divided in 3 parts instead of 2, as represented on the right of Figure 7.1: a middle part E_m (termed *boomerang switch*) is introduced between E_0 and E_1. Dunkelman *et al.* applied this framework to KASUMI.

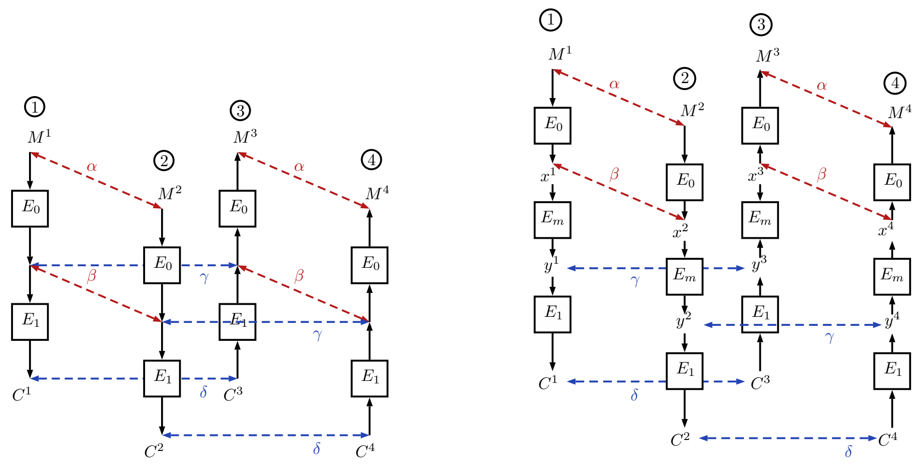


Figure 7.1: Configuration of the basic boomerang attack (left) and of the sandwich attack (right). Circled numbers correspond to a numbering that helps referencing states in the following discussions.

Cid *et al.* [CHP+18] recently introduced a tool called the *Boomerang Connectivity Table* that allows to easily evaluate the probability of the middle part E_m in the case where it covers one round and when the studied ciphers follows an SPN construction. Their technique reduces the problem of computing the probability of the boomerang switch over one round function to the one of computing it over one S-box only.

Equally as an S-box with a Difference Distribution Table with small coefficients provides resistance against differential attacks, an S-box with a Boomerang Connectivity Table (BCT) with small coefficients prevents an attacker from building efficient boomerang-style attacks. A study of some important families of S-boxes has been made in [BC18], [BPT19] and [LQSL19], just to cite a few. Another interesting line of work that followed the paper of Cid *et al.* is the determination of the probability of a boomerang switch E_m that covers more than one round, and that was addressed for SPN ciphers in [WP19, SQH19].

Still, to the best of our knowledge a similar analysis has not been provided yet for Feistel constructions [Fei74], while it cannot be denied that it is an equally important type of block cipher design, instantiated for instance by the widely used 3-DES and by CLEFIA [SSA+07] (ISO/IEC 29192-2).

Our Contributions. In this work, we address this lack and investigate what can be said on a boomerang switch when the studied cipher follows a Feistel construction. In case the Feistel round function contains some affine layers and a single S-box layer we introduce the FBCT, the Feistel counterpart of the Boomerang Connectivity Table and show that it is related to the second order derivative of the S-box at play. Our model elucidates the last switch that is not explained by the BCT by showing that the Feistel Switch corresponds to the diagonal of our table.

We study the properties of the FBCT for some categories of cryptographic S-boxes (in particular APN S-boxes and S-boxes based on the inverse mapping) and investigate if the maximum in the FBCT is invariant for S-boxes that are in the same equivalence classes for an equivalence that is affine, extended-affine and CCZ.

In a bottom-up approach, we start from this notion of FBCT (that covers switches of one round) and then introduce the FBDT to deal with a 2-round switch and finally propose the FBET that treats the case of an arbitrary number of rounds. We explain the relation between all these new notions and give examples of their application.

Finally, we illustrate our approach by applying it to the cipher LBlock-s (used in the CAESAR candidate LAC), and provide a 16-round distinguisher which probability is evaluated to be higher than $2^{-56.14}$.

7.2 Motivation: Disproving the Validity of a Previous Boomerang Distinguisher on LBlock

As a warm up, we study the related-key boomerang distinguisher devised by Liu *et al.* on LBlock [LGW12] and prove that the middle part contains a contradiction that invalidates the proposed boomerang distinguisher.

7.2.1 Specification of LBlock

LBlock was proposed at ACNS 2011 [WZ11a] by Wenling Wu and Lei Zhang. The cipher is lightweight and works on blocks of 64 bits and requires a key of 80 bits. It follows a Feistel structure and has the particularity to rely on 10 different 4-bit S-boxes. We give a short description of its design below and refer to [WZ11a] for more details and in particular for the description of the key schedule.

One LBlock encryption requires to iterate 32 times a round function that follows a 2-branch balanced Feistel structure with a twist, that is the right branch is modified by a rotation of 8 bit positions (see Figure 7.2). The other half of the internal state is modified by the F function that takes as parameter the 32-bit round key K_i. If the plaintext is denoted $M = X_1||X_0$ (where $||$ denotes the concatenation), we have for all $33 \geq i \geq 2$:

$$X_i = F(X_{i-1}, K_{i-1}) \oplus (X_{i-2} \lll 8).$$

More into details, the function F is defined as:

$$\begin{aligned} F : \{0,1\}^{32} \times \{0,1\}^{32} &\rightarrow \{0,1\}^{32} \\ (X, K_i) &\rightarrow U = P(S(X \oplus K_i)). \end{aligned}$$

S is an S-box layer that transforms each nibble Y_i into the nibble $Z_i = S_i(Y_i)$:

$$\begin{aligned} S : \{0,1\}^{32} &\rightarrow \{0,1\}^{32} \\ Y_7||Y_6||Y_5||Y_4||Y_3||Y_2||Y_1||Y_0 &\rightarrow Z_7||Z_6||Z_5||Z_4||Z_3||Z_2||Z_1||Z_0, \quad Z_i = S_i(Y_i). \end{aligned}$$

The S-boxes are detailed in Section 7.2.1. P is a permutation given by:

$$\begin{aligned} P : \{0,1\}^{32} &\rightarrow \{0,1\}^{32} \\ Z_7||Z_6||Z_5||Z_4||Z_3||Z_2||Z_1||Z_0 &\rightarrow U = Z_6||Z_4||Z_7||Z_5||Z_2||Z_0||Z_3||Z_1. \end{aligned}$$

	0x0	0x1	0x2	0x3	0x4	0x5	0x6	0x7	0x8	0x9	0xa	0xb	0xc	0xd	0xe	0xf
S_0	0xe	0x9	0xf	0x0	0xd	0x4	0xa	0xb	0x1	0x2	0x8	0x3	0x7	0x6	0xc	0x5
S_1	0x4	0xb	0xe	0x9	0xf	0xd	0x0	0xa	0x7	0xc	0x5	0x6	0x2	0x8	0x1	0x3
S_2	0x1	0xe	0x7	0xc	0xf	0xd	0x0	0x6	0xb	0x5	0x9	0x3	0x2	0x4	0x8	0xa
S_3	0x7	0x6	0x8	0xb	0x0	0xf	0x3	0xe	0x9	0xa	0xc	0xd	0x5	0x2	0x4	0x1
S_4	0xe	0x5	0xf	0x0	0x7	0x2	0xc	0xd	0x1	0x8	0x4	0x9	0xb	0xa	0x6	0x3
S_5	0x2	0xd	0xb	0xc	0xf	0xe	0x0	0x9	0x7	0xa	0x6	0x3	0x1	0x8	0x4	0x5
S_6	0xb	0x9	0x4	0xe	0x0	0xf	0xa	0xd	0x6	0xc	0x5	0x7	0x3	0x8	0x1	0x2
S_7	0xd	0xa	0xf	0x0	0xe	0x4	0x9	0xb	0x2	0x1	0x8	0x3	0x7	0x5	0xc	0x6
S_8	0x8	0x7	0xe	0x5	0xf	0xd	0x0	0x6	0xb	0xc	0x9	0xa	0x2	0x4	0x1	0x3
S_9	0xb	0x5	0xf	0x0	0x7	0x2	0x9	0xd	0x4	0x8	0x1	0xc	0xe	0xa	0x3	0x6

Table 7.1: The 10 S-boxes used in LBlock.

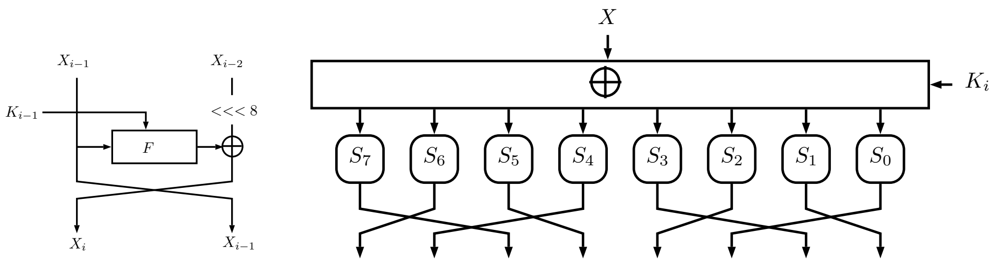


Figure 7.2: High-level description of one round of LBlock (left) and description of the F function (right).

7.2.2 Attack of Liu *et al.*

In 2012, Liu *et al.* [LGW12] proposed a 16-round related-key boomerang distinguishing attack on LBlock based on two 8-round related-key characteristics of low weight (that is, with very few active S-boxes). This attack is supposed to work in some (very large) weak-key class as it includes a key condition. With their parameters (that we recall in the next s), the probability of E_0 is $p = 2^{-14}$, while the probability of E_1 is $q = 2^{-16}$. They next computed the probability of the obtained distinguisher with the approximation p^2q^2, that gives 2^{-60}. Unfortunately, Section 7.2.3 details why the two characteristics E_0 and E_1 are in fact incompatible, meaning that the actual probability that the boomerang returns along these differential characteristics is 0.

Parameters of Liu *et al.*'s Related-Key Boomerang Distinguisher on LBlock

For E_0, there are seven active S-box transitions all of probability 2^{-2}, and no active S-boxes in the key schedule, resulting in $p = 2^{-14}$. E_1 is defined by the same 8-round characteristic positioned from round 9 to 16 with the change that for the master key difference to reach the required difference at the output of E_1 (in round 9), one S-box is activated in the key schedule (transition probability of 2^{-2}), meaning that $q = 2^{-16}$.

More into details and as depicted in Figure 7.3, the parameters of E_0 and E_1 are:

$$\Delta_i^L = 0x00000000,\ \Delta_i^R = 0x00000020,\ \Delta_o^L = 0x80001508,\ \Delta_o^R = 0x00000490.$$

Regarding the differences in the keys, we have:

$$\Delta_K^0 = 0x00000200000000000000,\ \Delta_K^1 = 0x0000c000000000000000.$$

The key derivation gives the following subkeys from $\Delta_K^0 = 0x00000200000000000000$:

$\Delta K_1 = 00000200$	$\Delta K_5 = 00000000$
$\Delta K_2 = 00000000$	$\Delta K_6 = 00000000$
$\Delta K_3 = 00000000$	$\Delta K_7 = 00800000$
$\Delta K_4 = 00010000$	$\Delta K_8 = 00000000$

While from $\Delta_K^1 = 0x0000c000000000000000$ we get:

$\Delta K_1 : 0000c000$	$\Delta K_5 : 00000000$	$\Delta K_9 : 00000200$	$\Delta K_{13} : 00000000$
$\Delta K_2 : 00000000$	$\Delta K_6 : 00000001$	$\Delta K_{10} : 00000000$	$\Delta K_{14} : 00000000$
$\Delta K_3 : 00000000$	$\Delta K_7 : 80000000$	$\Delta K_{11} : 00000000$	$\Delta K_{15} : 00800000$
$\Delta K_4 : 00600000$	$\Delta K_8 : 00000000$	$\Delta K_{12} : 00010000$	$\Delta K_{16} : 00000000$

The active S-box in the key schedule of E_1 appears in the round key 7 and uses the transition $S_9(0x3) = 0x8$ with probability 2^{-2}.

7.2.3 Incompatibility in the Distinguisher Proposed by Liu *et al.*

To help visualize the following discussion, we provide in Figure 7.4 a representation of the end of E_0 and of the beginning of E_1. In the following, we assume that the required transition happened in the key schedule (that is 0x3 $\rightarrow_{S_9}$ 0x8 in round 7).

Suppose that the quartet (M^1, M^2, M^3, M^4) follows the characteristics defining the boomerang specified by Liu *et al.* When looking at the beginning of the characteristic over E_1 we see that we expect a transition through the second S-box from a difference of 0x2 to 0x2, while by extending with probability 1 the differential characteristic over E_0 we see that the entering difference for this same S-box is 0x5 (see Figure 7.4). If we denote by t^i the nibble that enters the S-box number 2 of round 9 for M^i, this means that $t^1 \oplus t^3 = t^2 \oplus t^4$ =0x2 and that $S_2(t^1) \oplus S_2(t^3) = S_2(t^2) \oplus S_2(t^4)$ =0x2. Also, we have $t^1 \oplus t^2 = t^3 \oplus t^4$ =0x5.

By referring to S_2, we can list the possible input nibbles that make the transition from an input difference of 0x2 to an output difference of 0x2, and we obtain that $(t^1, t^3) \in$ {(0x1,0x3), (0x3,0x1), (0x8,0xa), (0xa,0x8)}. Since t^2 and t^4 are separated from t^1 and t^3 by a difference of 0x5 we can deduce that their values are in the following set: $(t^2,\ t^4) \in$ {(0x4,0x6), (0x6,0x4), (0xd,0xf), (0xf,0xd)}.

The contradiction comes from the fact that none of these pairs allows the required transition from 0x2 to 0x2, an observation that can be rewritten as: {(0x1, 0x3), (0x3, 0x1), (0x8, 0xa), (0xa, 0x8)} $\cap$ 0x5 $\oplus$ {(0x1, 0x3), (0x3, 0x1), (0x8, 0xa), (0xa, 0x8)} $= \varnothing$ since we want that the shifted values (t^1, t^3) also allows the desired S-box transition.

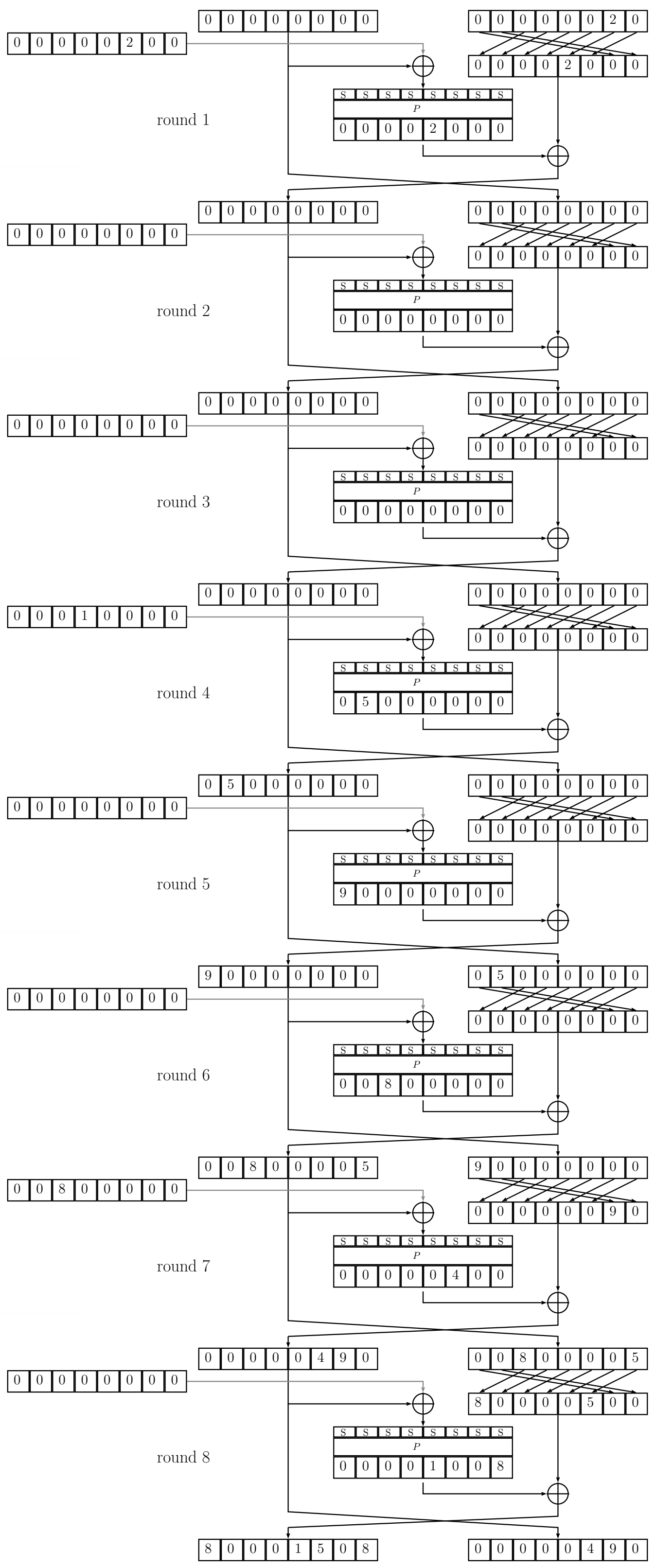


Figure 7.3: The 8-round related-key characteristic used in Liu *et al.*'s attack, used for both E_0 and E_1 with $\Delta K_1 = \Delta K_9, \Delta K_2 = \Delta K_{10}, \cdots, \Delta K_8 = \Delta K_{16}$.

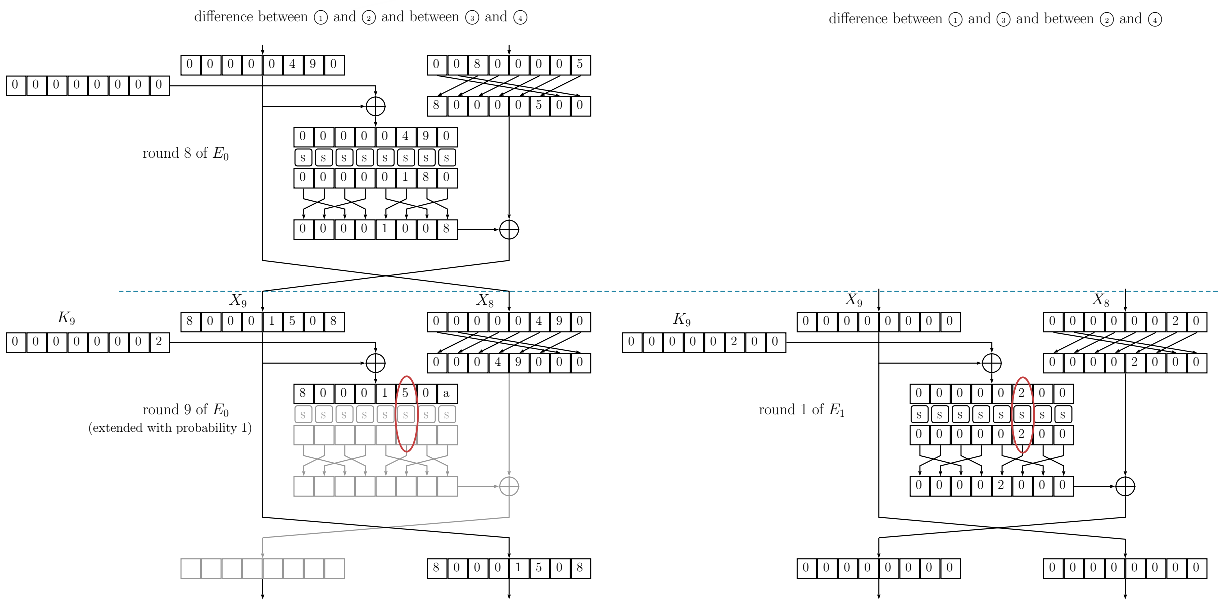

difference between ① and ② and between ③ and ④
difference between ① and ③ and between ② and ④
round 8 of E_0
round 9 of E_0
(extended with probability 1)
round 1 of E_1
X_9
X_8
K_9

Figure 7.4: Middle rounds of the boomerang distinguisher proposed in [LGW12].

This incompatibility implies that the boomerang over the middle round never returns,[1] and consequently the related-key distinguisher proposed by Liu *et al.* is invalid as no quartet can follow the required characteristic.

7.3 FBCT: the Feistel Counterpart of the BCT

The inconsistency found in the previous section (that is reminiscent of the examples given by Murphy in [Mur11]) calls for a tool to automatically study the behavior of the junction between E_0 and E_1.

In fact, this problem has recently been addressed in the case of Substitution-Permutation Networks with the introduction of the *Boomerang Connectivity Table (BCT)* by Cid et al [CHP+18]. However, no similar tool has been devised so far to deal with boomerang attacks on Feistel Networks. We address this shortfall in this section by introducing the[2] FBCT.

7.3.1 Definition of the FBCT

The (SPN) Boomerang Connectivity Table. The essence of the Boomerang Connectivity Table introduced by Cid *et al.* is similar to the one of the well-known Difference Distribution Table: instead of looking at the property of a round as a whole (thus at a function of usually 64 or 128 bits), the problem is reduced to one we can easily study given its small size: examining each S-box of the S-layer independently. While the DDT describes the differential properties of

[1]Note that we confirmed this experimentally by verifying that for a sample of 2^{10} keys and 2^{10} messages the boomerang never comes back along the announced differences in one or even two middle rounds.

[2]To stress the similarity between the notions we introduce here and the ones that have been previously introduced in the case of boomerang switches on SPN, we basically use the same acronyms, simply adding the letter "F" in front of them to recall that we are looking at the Feistel case.

each S-box from which are deduced the ones of the round, the BCT gives the probability of a boomerang switch over each S-box from which is deduced the one of the round.

The formal definition of the BCT is recalled below: it is a table that gives at line Δ_i, column ∇_o the number of values for which a boomerang of input Δ_i and output difference ∇_o comes back. It corresponds to the following formula, depicted in Figure 7.5:

Definition 7.1 (Boomerang Connectivity Table [CHP+18])**.** *Let S be a permutation of $\mathbb{F}_2^n$, and $\Delta_i, \nabla_o \in \mathbb{F}_2^n$. The Boomerang Connectivity Table (BCT) of S is given by a $2^n \times 2^n$ table, in which the entry for the (Δ_i, ∇_o) position is given by:*

$$BCT(\Delta_i, \nabla_o) = \#\{x \in \mathbb{F}_2^n | S^{-1}(S(x) \oplus \nabla_o) \oplus S^{-1}(S(x \oplus \Delta_i) \oplus \nabla_o) = \Delta_i\}.$$

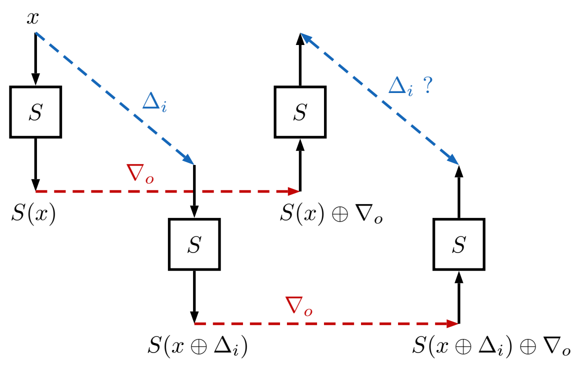


Figure 7.5: The value of $BCT(\Delta_i, \nabla_o)$ corresponds to the number of S-box inputs x that make the boomerang over 1 round come back.

The Feistel Boomerang Connectivity Table. The previous definition is only valid for an S-box that is part of an S-layer in an SPN cipher: the objective of this work is to address the need of the counterpart for a Feistel cipher.

As a hint of what we introduce below, remember that Feistel ciphers have the practical advantage that decryption is performed by executing the same function as for encryption, simply changing the order of the round keys. This change is at the heart of the Feistel counterpart of the BCT that we introduce now: here, the inverse of the S-box is never at play.

We start by illustrating our theory on the generic Feistel cipher represented in Figure 7.6: it is a balanced Feistel with two branches, that we denote L and R. The output of one round is given by $F(L) \oplus R||L$, where the F function is defined by a round key addition, an S-layer and a linear layer L. Note that the details of the linear layers of F play no role in our discussion, and that the only important point is that F contains one S-layer made by the concatenation of t n-bit S-boxes.

We are interested in the probability of the following 1-round boomerang switch: we have an input difference equal to $\beta^L||\beta^R$ between state ① and ②, an output difference equal to $\gamma^L||\gamma^R$ between state ① and ③, and ② and ④, and we want that the input difference between state ③ and ④ is equal to $\beta^L||\beta^R$.

Left part of the difference. We start by studying the cost of obtaining that the left difference between state ③ and ④ has the desired value of β^L.

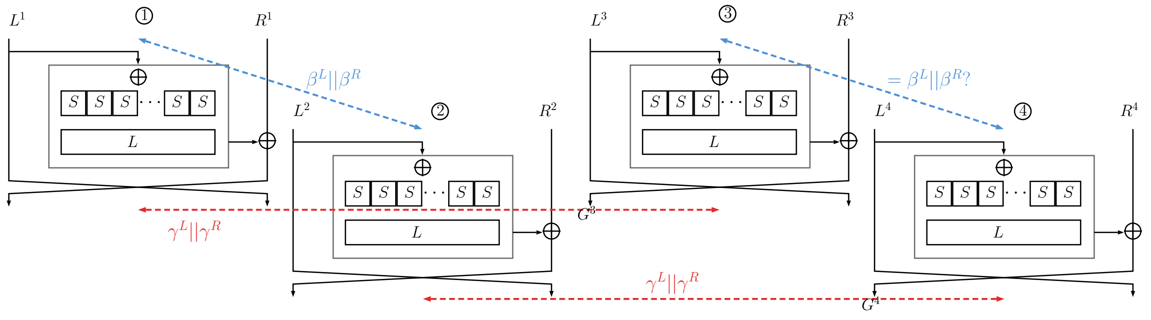


Figure 7.6: Boomerang switch over a generic Feistel round.

Given the fact that the left branch is the one that is not modified through one round of Feistel we can easily conclude that the desired difference comes for free:

$$\begin{aligned} L^3 \oplus L^4 &= (L^3 \oplus L^1) \oplus (L^1 \oplus L^2) \oplus (L^2 \oplus L^4) \\ &= \gamma^R \oplus \beta^L \oplus \gamma^R = \beta^L. \end{aligned}$$

Right part of the difference. We now focus on obtaining a difference of β^R between the right part of state number ③ and ④. By naming G^3 and G^4 the left output after one round in state ③ and ④ (see Figure 7.6), we obtain the following simplification:

$$\begin{aligned} R^3 \oplus R^4 &= F(L^1 \oplus \gamma^R) \oplus G^3 \oplus F(L^1 \oplus \gamma^R \oplus \beta^L) \oplus G^4 \\ &= F(L^1 \oplus \gamma^R) \oplus F(L^1) \oplus R^1 \oplus \gamma^L \oplus F(L^1 \oplus \gamma^R \oplus \beta^L) \\ &\quad \oplus F(L^1 \oplus \beta^L) \oplus R^1 \oplus \beta^R \oplus \gamma^L \\ &= F(L^1 \oplus \gamma^R) \oplus F(L^1) \oplus F(L^1 \oplus \gamma^R \oplus \beta^L) \\ &\quad \oplus F(L^1 \oplus \beta^L) \oplus \beta^R. \end{aligned}$$

For this difference to be equal to β^R we need that

$$F(L^1) \oplus F(L^1 \oplus \gamma^R) \oplus F(L^1 \oplus \beta^L) \oplus F(L^1 \oplus \gamma^R \oplus \beta^L) = 0.$$

We use the fact that the only non-linear function of F is an S-layer made by a concatenation of small S-boxes to rewrite this condition as a set of independent conditions on smaller parts of the states, and obtain t independent equations of the form:

$$S(x) \oplus S(x \oplus \Delta_i) \oplus S(x \oplus \nabla_o) \oplus S(x \oplus \Delta_i \oplus \nabla_o) = 0.$$

Where Δ_i is the difference at the input of the considered S-box between state ① and ②, deduced from β^L, and ∇_o is the difference at the input of the considered S-box between state ① and ③ and ② and ④, deduced from γ^R.

The resulting probability of the boomerang switch over one round is then the product of the probabilities for each S-box, that are of the form

$$2^{-n} \times \#\{x \in \mathbb{F}_2^n | S(x) \oplus S(x \oplus \Delta_i) \oplus S(x \oplus \nabla_o) \oplus S(x \oplus \Delta_i \oplus \nabla_o) = 0\}.$$

This discussion leads us to the introduction of the following definition:

Definition 7.2 (FBCT)**.** *Let S be a function from $\mathbb{F}_2^n$ to $\mathbb{F}_2^m$, and $\Delta_i, \nabla_o \in \mathbb{F}_2^n$. The* FBCT *of S is given by a $2^n \times 2^n$ table T, in which the entry for the (Δ_i, ∇_o) position is given by:*

$$\mathtt{FBCT}_S(\Delta_i, \nabla_o) = \#\{x \in \mathbb{F}_2^n | S(x) \oplus S(x \oplus \Delta_i) \oplus S(x \oplus \nabla_o) \oplus S(x \oplus \Delta_i \oplus \nabla_o) = 0\}.$$

Since we do not have to consider a bijective S-box, we define $\mathtt{FBCT}_S$ for any S-box S from $\mathbb{F}_2^n$ to $\mathbb{F}_2^m$ with possibly $n \neq m$. In the following we leave out the S index and simply write FBCT when the S-box we are referring to is clear from the context.

Once the table is built, the probability that a boomerang comes back over 1 round of a Feistel scheme is simply the product of the corresponding coefficients of the FBCT divided by 2^n. An example of FBCT is provided in Table 7.2 in the case of the S-box[3] S_2 of LBlock.

Table 7.2: FBCT of the S-box S_2 of LBlock.

	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
0	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16
1	16	16	0	0	0	0	0	0	0	0	8	8	0	0	0	0
2	16	0	16	0	0	0	0	0	0	8	0	8	0	0	0	0
3	16	0	0	16	8	8	8	8	0	0	0	0	0	0	0	0
4	16	0	0	8	16	0	0	8	0	0	0	0	0	0	0	0
5	16	0	0	8	0	16	8	0	0	0	0	0	0	0	0	0
6	16	0	0	8	0	8	16	0	0	0	0	0	0	0	0	0
7	16	0	0	8	8	0	0	16	0	0	0	0	0	0	0	0
8	16	0	0	0	0	0	0	0	16	0	0	0	0	0	0	0
9	16	0	8	0	0	0	0	0	0	16	0	8	0	0	0	0
a	16	8	0	0	0	0	0	0	0	0	16	8	0	0	0	0
b	16	8	8	0	0	0	0	0	0	8	8	16	0	0	0	0
c	16	0	0	0	0	0	0	0	0	0	0	0	16	0	0	0
d	16	0	0	0	0	0	0	0	0	0	0	0	0	16	0	0
e	16	0	0	0	0	0	0	0	0	0	0	0	0	0	16	0
f	16	0	0	0	0	0	0	0	0	0	0	0	0	0	0	16

It is easy to see that the formula of the FBCT corresponds to the number of times the order 2 derivative with respect to Δ_i and ∇_o of the vectorial Boolean function S cancels out. We formalize this and study its properties in Section 7.4. First, we show that the same reasoning applies to variants of this construction.

7.3.2 Some Variants of Feistel Constructions for which the FBCT Applies

We show here that the FBCT tool covers more constructions than the classical Feistel cipher, by providing three examples: the type I and II variants introduced by Zheng, Matsumoto and Imai in [ZMI90] and the source-heavy (also called contracting) construction as implemented in SMS4 [Dt08]. A representation of the round structure of these types is given in Figure 7.7 in the case of 4-branch networks.

The only assumption we make is that the F and F' functions used in theses constructions are composed of some linear or affine operations (like for instance matrix multiplication, permutations, Xor of constants or of round keys) and of one S-layer. We show below that the relations that need to be fulfilled in these cases can also be expressed as a product of FBCT coefficients.

[3]Note that the FBCT of the 10 S-boxes of LBlock are the same. This is not a direct implication of the fact that the S-boxes are affine equivalent (counter-examples to this can easily be found).

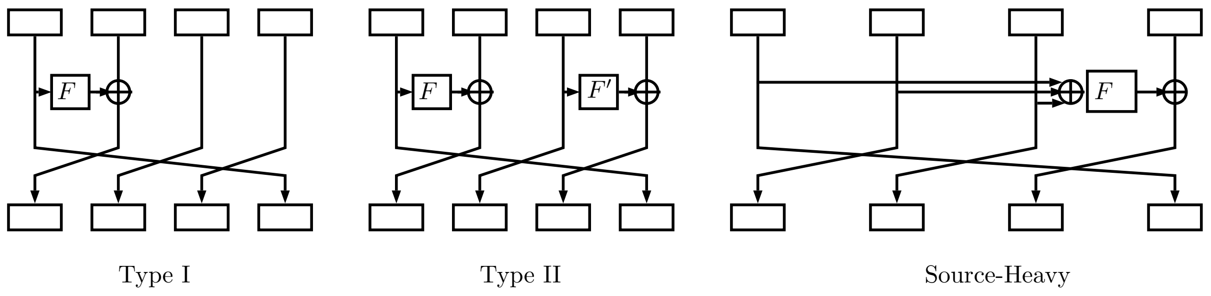


Figure 7.7: One round of the Feistel construction of type I, type II and source-heavy for variants with 4 branches.

Type I: Referring to Figure 7.7, one round of type I can be seen as one round of classic Feistel with some (2 in the picture) additional branches that are independent and not affected by a F function. Thus, the reasoning made in Section 7.3.1 can be extended to the case of type I construction and the probability that the boomerang switch over one round happens as required is the product of the `FBCT` coefficients corresponding to the S-boxes contained in F.

Type II: In a similar way, one round of type II can be seen as the concatenation of several (2 in the picture) classical Feistels that are independent one from the others. The reasoning made in Section 7.3.1 applies to this case and the probability of the boomerang switch is made by the product of the `FBCT` coefficients of the S-boxes of the F functions at play.

Source-Heavy: This case can also easily be treated with the `FBCT`. It can be shown that the one-round boomerang switch represented in Figure 7.8 comes back if the following condition is fulfilled:

$$F(x_1^1 \oplus x_2^1 \oplus x_3^1) \oplus F(x_1^1 \oplus x_2^1 \oplus x_3^1 \oplus (A \oplus B \oplus C)) \oplus$$
$$F(x_1^1 \oplus x_2^1 \oplus x_3^1 \oplus (a \oplus b \oplus d)) \oplus F(x_1^1 \oplus x_2^1 \oplus x_3^1 \oplus (A \oplus B \oplus C) \oplus (a \oplus b \oplus d)) = 0$$

Which can be rewritten as a product of the `FBCT` coefficients of the S-box of F, with the two parameters depending on A, B, and C for one, and a, b and d for the other.

Note that the discussion above has to be nuanced as the application of the `FBCT` to other cases (as for instance type III construction) might not be straightforward.

7.3.3 Evaluation of the 1-round Boomerang Switch of Liu *et al.*'s Attack with the `FBCT`

We focus on the S-box S_2 of round 9 of the cipher. From E_0, the input difference of S-box 2 is 0x5, so following previous notation we have $\Delta_i = 0\text{x}5$. When referring to E_1 and taking into account the difference coming from the round key we have $\nabla_o = 0\text{x}2$. The `FBCT` coefficient we are interested in is then $\texttt{FBCT}_{S_2}(0\text{x}5, 0\text{x}2)$. Referring to Table 7.2, we see that the corresponding cell has a value equal to 0, meaning that the 1-round boomerang switch is impossible.

Note that this incompatibility is even more general than the one we discussed in Section 7.2.3, as in Section 7.2.3 we fixed an additional parameter namely one S-box output. This vision corresponds to what we introduce in Section 7.5.1 under the name of Feistel Boomerang Difference Table (`FBDT`).

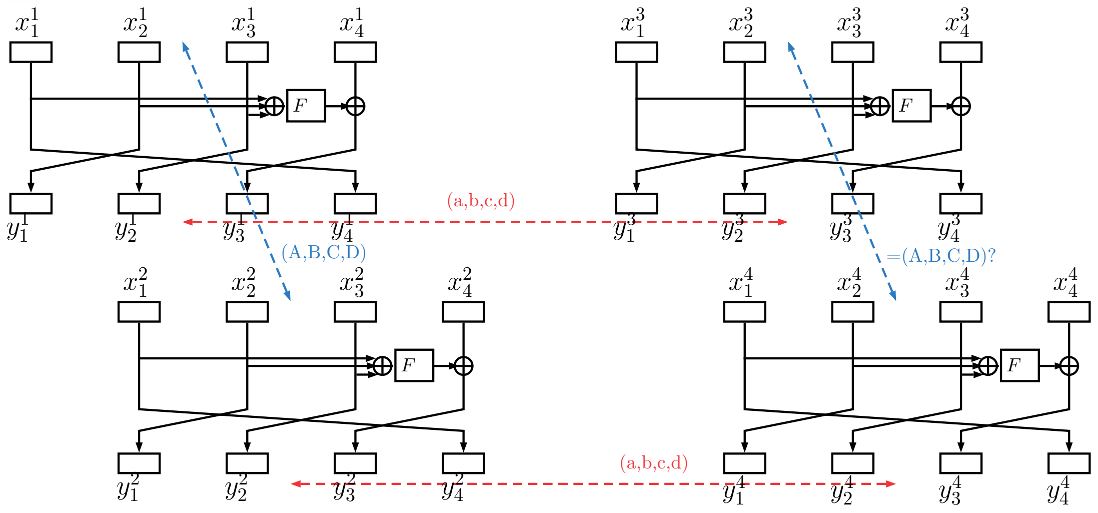


Figure 7.8: Boomerang switch over one round of the source-heavy construction.

7.3.4 Relation Between the FBCT and the Feistel Switch

While the Feistel case is not covered by the Boomerang Connectivity Table, a first step in understanding the case of boomerang distinguishers for Feistel constructions has been made by Wagner himself while analyzing Khufu [Wag99]. His observation was later referred under the name of *Feistel Switch*, for instance in the related-key cryptanalysis of the AES-192 and AES-256 by Biryukov and Khovratovich [BK09], in which one can read:

> Surprisingly, a Feistel round with an arbitrary function (e.g., an S-box) can be passed for free in the boomerang attack (this was first observed in the attack on cipher Khufu in [Wag99]). Suppose the internal state (X, Y) is transformed to $(Z = X \oplus f(Y), Y)$ at the end of E_0. Suppose also that the E_0 difference before this transformation is $(\Delta X, \Delta Y)$, and that the E_1 difference after this transformation is $(\Delta Z, \Delta Y)$. [...] Therefore, the decryption phase of the boomerang creates the difference ΔX in X at the end of E_0 "for free".

By analyzing this setting in the way we did in Section 7.3.1 we can show that an internal state (X, Y) allows the boomerang to come back if Y verifies:

$$f(Y) \oplus f(Y \oplus \Delta Y) \oplus f(Y \oplus \Delta Y) \oplus f(Y \oplus \Delta Y \oplus \Delta Y) = 0,$$

(we have $\gamma^R = \beta^L = \Delta Y$ with our previous notation) which is always true. Moreover if the Feistel round function is made of some linear operations and an S-layer, the previous setting means that for every S-box we are looking at coefficients that are on the diagonal of the FBCT.

7.4 Properties of the FBCT

This section gives a review of the most important properties of the Feistel boomerang connectivity table. We start by listing the constants of the table and then investigate the properties of the FBCT of two crucial classes of vectorial function, namely APN functions and functions based on the inverse mapping. We also study if the so-called Feistel boomerang uniformity is constant for S-boxes belonging to the same equivalence classes, for various definitions of equivalence. We conclude this section by giving a comparison of the BCT and FBCT properties.

7.4.1 Basics on vectorial Boolean Functions

Let $S : \mathbb{F}_2^n \longrightarrow \mathbb{F}_2^m$ be a vectorial Boolean function. The set of all vectorial Boolean functions from $\mathbb{F}_2^n$ to $\mathbb{F}_2^m$ is denoted $\mathcal{B}(n,m)$. The derivative of $S \in \mathcal{B}(n,m)$ at $\Delta_i \in \mathbb{F}_2^n$ is defined as

$$D_{\Delta_i} S(x) = S(x) \oplus S(x \oplus \Delta_i)$$

for all $x \in \mathbb{F}_2^n$.

The first derivative is at the basis of the Difference Distribution Table (DDT) of a given vectorial function S, defined as:

$$\texttt{DDT}_S(\Delta_i, \delta) = \#\{x \in \mathbb{F}_2^n : \ S(x) \oplus S(x \oplus \Delta_i) = \delta\}.$$

The value of $\max_{\Delta_i \neq 0, \delta}\{\texttt{DDT}_S(\Delta_i, \delta)\}$ is called the (differential) uniformity of S.

This definition is extended to higher-order derivatives as follows: let $\Delta_i^1, \Delta_i^2, \dots, \Delta_i^k$ be a basis of a k-dimensional subspace V of $\mathbb{F}_2^n$. The k-th derivative of S with respect to V, denoted by $D_V S$, is defined as $D_V S(x) = D_{\Delta_i^1} D_{\Delta_i^2} \cdots D_{\Delta_i^k} S(x)$, for all $x \in \mathbb{F}_2^n$.

Given this definition, it is direct to see that for an $n \times m$ S-box seen as an element of $\mathcal{B}(n,m)$, the value of $\texttt{FBCT}(\Delta_i, \nabla_o)$ corresponds to the number of zeroes of the function $D_{\Delta_i} D_{\nabla_o} S$ extended to the cases where Δ_i and ∇_o are not linearly independent.

7.4.2 Some Direct Properties of any FBCT

We start with a series of simple properties that are easily observable from the definition:

Property 7.3. *The coefficients of the* FBCT *of $S \in \mathcal{B}(n,m)$ verify the following:*

1. *Symmetry: for all $0 \leq \Delta_i, \nabla_o \leq 2^n - 1$, $\texttt{FBCT}(\Delta_i, \nabla_o) = \texttt{FBCT}(\nabla_o, \Delta_i)$.*
2. *Fixed values:*
   (a) *First line: for all $0 \leq \nabla_o \leq 2^n - 1$, $\texttt{FBCT}(0, \nabla_o) = 2^n$ (ladder switch),*
   (b) *First column: for all $0 \leq \Delta_i \leq 2^n - 1$, $\texttt{FBCT}(\Delta_i, 0) = 2^n$ (ladder switch),*
   (c) *Diagonal: for all $0 \leq \Delta_i \leq 2^n - 1$, $\texttt{FBCT}(\Delta_i, \Delta_i) = 2^n$ (Feistel switch).*
3. *Multiplicity: for all $0 \leq \Delta_i, \nabla_o \leq 2^n - 1$, $\texttt{FBCT}(\Delta_i, \nabla_o) \equiv 0 \bmod 4$.*
4. *Equalities: for all $0 \leq \Delta_i, \nabla_o \leq 2^n - 1$, $\texttt{FBCT}(\Delta_i, \nabla_o) = \texttt{FBCT}(\Delta_i, \Delta_i \oplus \nabla_o)$.*

Proof. All the properties are easily deduced from Definition 7.2:

(1) and (4) are proven by writing the expressions of the coefficients at play. Note that from symmetry we also have $\texttt{FBCT}(\Delta_i, \Delta_i \oplus \nabla_o) = \texttt{FBCT}(\nabla_o, \Delta_i \oplus \nabla_o)$.

(2)a. and (2)b. correspond to the ladder switch proposed in [BK09] that works the same way for Feistel and SPN ciphers: if either Δ_i or ∇_o is zero, it means that two pairs of messages inside the quartet share the same S-box input, and the boomerang comes back with probability 1. This is formally shown as follows:

$$\begin{aligned} \texttt{FBCT}(0, \nabla_o) &= \#\{x \in \mathbb{F}_2^n | S(x) \oplus S(x) \oplus S(x \oplus \nabla_o) \oplus S(x \oplus \nabla_o) = 0\} \\ &= 2^n, \end{aligned}$$

and similarly $\texttt{FBCT}(\Delta_i, 0) = 2^n$. The Feistel switch recalled in Section 7.3.4 is also easily proven: if $\Delta_i = \nabla_o$ the FBCT coefficients correspond to the number of $x \in \mathbb{F}_2^n$ that are solutions to $S(x) \oplus S(x \oplus \Delta_i) \oplus S(x \oplus \Delta_i) \oplus S(x) = 0$. Since every value of x fulfills this, $\texttt{FBCT}(\Delta_i, \Delta_i) = 2^n$.

(3) The property is verified for the case $\Delta_i = \nabla_o$ (since we can reasonably assume $n > 1$), so we focus on the case where $\Delta_i \neq \nabla_o$. If no x is solution the property is verified, while if there is at least one $x \in \mathbb{F}_2^n$ that is a solution then three more distinct values $x \oplus \Delta_i$, $x \oplus \nabla_o$ and $x \oplus \Delta_i \oplus \nabla_o$ also are, which proves the multiplicity. □

Given that the coefficients in the first line, first column and diagonal of the FBCT are always equal to the maximum that is 2^n, we define the *boomerang uniformity* a bit differently from what has been done for the BCT[4]:

Definition 7.4 (F-Boomerang Uniformity)**.** *The F-Boomerang uniformity corresponds to the highest value in the* FBCT *without considering the first row, the first column and the diagonal:*

$$\beta^F = \max_{\Delta_i \neq 0, \nabla_o \neq 0, \Delta_i \neq \nabla_o.} \texttt{FBCT}(\Delta_i, \nabla_o).$$

From the designer point of view, it is preferable to use an S-box with a small F-boomerang uniformity. This goal can be reached by opting for an APN function, as we show below.

7.4.3 On the FBCT of APN Functions

A function $S \in \mathcal{B}(n, n)$ is called almost perfect nonlinear (APN) if for any $\Delta_i, \delta \in \mathbb{F}_2^n$ with $\Delta_i \neq 0$ the equation $S(x) \oplus S(x \oplus \Delta_i) = \delta$ has either 0 or 2 solutions. Alternatively, we know (refer for instance to [Car10], page 417) that S is APN if and only if for any non-zero $\Delta_i, \nabla_o \in \mathbb{F}_2^n$ with $\Delta_i \neq \nabla_o$, $D_{\Delta_i} D_{\nabla_o} S(x) \neq 0$ for all $x \in \mathbb{F}_2^n$. This directly implies the following theorem:

Theorem 7.5. *Let $S \in \mathcal{B}(n, n)$. S is an APN function if and only if its* FBCT *verifies* $\texttt{FBCT}(\Delta_i, \nabla_o) = 0$ *for all* $1 \leq \Delta_i \neq \nabla_o \leq 2^n - 1$.

A direct implication of this theorem is that any non-APN function has a non-zero coefficient at a position that is not in the first row, first column or diagonal of its FBCT, so in particular a Feistel boomerang uniformity higher or equal to 4.

7.4.4 On the FBCT of S-boxes based on the Inverse Mapping

Another important and widely used set of S-boxes are the ones based on the inverse mapping, which include (among others) the 8-bit S-boxes of CAMELLIA [AIK+01], Clefia [SSA+07] and SMS4 [Dt08] and the 4-bit S-box of Twine [SMMK13].

We know that $\mathbb{F}_2^n$ and $\mathbb{F}_{2^n}$ are vector isomorphic over $\mathbb{F}_2$, i.e., with respect to a fixed basis α_i, $1 \leq i \leq n$, of $\mathbb{F}_{2^n}$, any element of $x \in \mathbb{F}_{2^n}$ can be uniquely written as $x = \oplus_{i=1}^n x_i \alpha_i$, where $(x_1, x_2, \ldots, x_n) \in \mathbb{F}_2^n$. A mapping $S : \mathbb{F}_{2^n} \to \mathbb{F}_{2^n}$ of the form $S(x) = x^{2^n - 2}$ is called an inverse mapping.

The importance of this family of functions comes from its very good cryptographic properties (that led to it being selected to build the AES [AES01] S-box for instance). Indeed, Nyberg [Nyb94] showed that if n is odd, then the inverse function over $\mathbb{F}_{2^n}$ is APN, and if n is even, then each

[4]Recall that for the BCT the boomerang uniformity of an S-box is $\beta = \max_{\Delta_i \neq 0, \nabla_o \neq 0} BCT(\Delta_i, \nabla_o)$.

row of its DDT has exactly one 4 and $(2^{n-1} - 2)$ occurrences of the number 2 (and in particular that the S-box is differentially 4-uniform). Given that we already discussed the case of APN functions in Section 7.4.3, we focus here on the case where n is even.

Property 7.6. *In each row (except the first) of the* FBCT *of the inverse mapping over an even number of bits, the values* 2^n*,* 4 *and* 0 *occur* 2*,* 2 *and* $2^n - 4$ *times, respectively.*

Proof. Using a *reductio ad absurdum* argument, we start by showing that the only possible values in the FBCT of the inverse mapping over an even number of bits are 0, 4 and 2^n. Since for every FBCT the coefficients in the first line, first column and diagonal are equal to 2^n, we focus on the other positions.

Suppose that for given non-zero Δ_i and ∇_o verifying $\Delta_i \neq \nabla_o$ we have $\texttt{FBCT}(\Delta_i, \nabla_o) > 4$. Given that the coefficients of the FBCT are multiple of 4 this implies that we have at least 8 distinct values $\mathbf{x}, \mathbf{x} \oplus \Delta_i, \mathbf{x} \oplus \nabla_o, \mathbf{x} \oplus \Delta_i \oplus \nabla_o, \mathbf{y}, \mathbf{y} \oplus \Delta_i, \mathbf{y} \oplus \nabla_o, \mathbf{y} \oplus \Delta_i \oplus \nabla_o$ that are solutions of:

$$S(x) \oplus S(x \oplus \Delta_i) \oplus S(x \oplus \nabla_o) \oplus S(x \oplus \Delta_i \oplus \nabla_o) = 0.$$

This can be rewritten as:

$$S(\mathbf{x}) \oplus S(\mathbf{x} \oplus \Delta_i) = S(\mathbf{x} \oplus \nabla_o) \oplus S(\mathbf{x} \oplus \Delta_i \oplus \nabla_o) = \delta_1, \tag{7.1}$$
$$S(\mathbf{y}) \oplus S(\mathbf{y} \oplus \Delta_i) = S(\mathbf{y} \oplus \nabla_o) \oplus S(\mathbf{y} \oplus \Delta_i \oplus \nabla_o) = \delta_2. \tag{7.2}$$

The first line indicates that the equation $S(x) \oplus S(x \oplus \Delta_i) = \delta_1$ has at least 4 distinct solutions, that is $\texttt{DDT}(\Delta_i, \delta_1) \geq 4$. Similarly, the second line shows that $\texttt{DDT}(\Delta_i, \delta_2) \geq 4$. There are two possibilities: if $\delta_1 = \delta_2$ we obtain that $\texttt{DDT}(\Delta_i, \delta_1) \geq 8$ which contradicts that the differential uniformity of the considered S-box is 4, while if $\delta_1 \neq \delta_2$ we obtain that there are two coefficients in the same line of the DDT with a coefficient higher or equal to 4. In both cases we obtain a contradiction, so we conclude that the only possible values in the FBCT are 0, 4 and 2^n.

To conclude on the number of occurrences of each coefficient we need to prove that there are only 2 coefficients equal to 4 in each line. We consider $\Delta_i, \delta \in \mathbb{F}_2^n$ so that $\texttt{DDT}(\Delta_i, \delta) = 4$. There exist $\mathbf{x}$, $\mathbf{x} \oplus \Delta_i$, $\mathbf{y}$, $\mathbf{y} \oplus \Delta_i \in \mathbb{F}_2^n$ with $\mathbf{x} \neq \mathbf{y}$ and $\mathbf{x} \neq \mathbf{y} \oplus \Delta_i$ such that:

$$\begin{aligned} & S(\mathbf{x}) \oplus S(\mathbf{x} \oplus \Delta_i) = \delta = S(\mathbf{y}) \oplus S(\mathbf{y} \oplus \Delta_i) \\ \textit{i.e.} \quad & S(\mathbf{x}) \oplus S(\mathbf{x} \oplus \Delta_i) \oplus S(\mathbf{x} \oplus \nabla_o) \oplus S(\mathbf{x} \oplus \nabla_o \oplus \Delta_i) = 0, \text{ where } \nabla_o = \mathbf{x} \oplus \mathbf{y}. \end{aligned}$$

As a consequence, $\texttt{FBCT}(\Delta_i, \nabla_o) = \texttt{FBCT}(\Delta_i, \Delta_i \oplus \nabla_o) = 4$ so there are at least two '4' in each line of the FBCT. Suppose there is one more coefficient equal to 4 in this line, that is there exists $\mathbf{c} \in \mathbb{F}_2^n$ with $\mathbf{c} \neq \nabla_o$ and $\mathbf{c} \neq \Delta_i \oplus \nabla_o$ such that $\texttt{FBCT}(\Delta_i, \mathbf{c}) = 4$ and let $\mathbf{z} \in \{x \in \mathbb{F}_2^n | S(x) \oplus S(x \oplus \Delta_i) \oplus S(x \oplus c) \oplus S(x \oplus \Delta_i \oplus c) = 0\}$. We have:

$$\begin{aligned} & S(\mathbf{z}) \oplus S(\mathbf{z} \oplus \Delta_i) \oplus S(\mathbf{z} \oplus \mathbf{c}) \oplus S(\mathbf{z} \oplus \Delta_i \oplus \mathbf{c}) = 0 \\ \textit{i.e.} \quad & S(\mathbf{z}) \oplus S(\mathbf{z} \oplus \Delta_i) = \delta' = S(\mathbf{w}) \oplus S(\mathbf{w} \oplus \Delta_i), \text{ where } \mathbf{w} = \mathbf{z} \oplus \mathbf{c}. \end{aligned}$$

$\texttt{FBCT}(\Delta_i, \mathbf{c}) = 4$ yields $\mathbf{c} \neq \Delta_i$ and $\mathbf{c} \neq 0$ and thus $\texttt{DDT}(\Delta_i, \delta') = 4 = \texttt{DDT}(\Delta_i, \delta)$. Since each row of the considered S-box DDT has exactly one entry that equals 4, it follows that $\delta = \delta'$ and that $\mathbf{z}, \mathbf{w} \in \{\mathbf{x}, \mathbf{x} \oplus \Delta_i, \mathbf{y}, \mathbf{y} \oplus \Delta_i\}$, which leads to the contradiction that $\mathbf{c} \in \{\Delta_i, \nabla_o, \Delta_i \oplus \nabla_o\}$. □

7.4.5 On the `FBCT` of Equivalent S-boxes

Various notions of equivalence are frequently used when studying S-boxes, among which linear, affine, extended-affine and CCZ equivalence [CCZ98]. These various concepts play an important role to categorize sets of S-boxes since central cryptographic properties (differential, linear and sometimes algebraic degree) are constant for equivalent S-boxes. In this section we investigate if the F-boomerang uniformity is preserved under these various notions of equivalence.

Linear, Affine and Extended-Affine Equivalence. As their names suggest, the three first flavors we start with are related as follows: linear equivalence is a sub-case of affine equivalence, and affine equivalence is a particular case of extended-affine equivalence.

Definition 7.7 (Linear, Affine and Extended-Affine Equivalence)**.** *Two vectorial Boolean functions $F, G \in \mathcal{B}(n,m)$ are called extended-affine equivalent if there exist two nonsingular matrices $A \in GL(n,\mathbb{F}_2)$, $B \in GL(m,\mathbb{F}_2)$, $(a,b) \in \mathbb{F}_2^n \times \mathbb{F}_2^m$ and an affine function $C : \mathbb{F}_2^n \to \mathbb{F}_2^m$ such that for all $x \in \mathbb{F}_2^n$*

$$G(x) = B(F(A(x) \oplus a)) \oplus C(x) \oplus b,$$

where $GL(n,\mathbb{F}_2)$ is the set of all nonsingular binary matrices of order n. If $C = 0$, then F and G are affine equivalent, and if in addition a and b are equal to zero then they are linear equivalent.

Theorem 7.8. *The multi-set composed of all values in the* `FBCT` *is preserved under extended-affine nonsingular transformation. Namely, we have that $\mathtt{FBCT}_G(u,v) = \mathtt{FBCT}_F(u',v')$ where $u' = A(u)$ and $v' = A(v)$.*

Proof. Suppose that $G(x) = B(F(A(x) \oplus a)) \oplus C(x) \oplus b$ for all $x \in \mathbb{F}_2^n$, where $A \in GL(n,\mathbb{F}_2)$, $B \in GL(m,\mathbb{F}_2)$, $(a,b) \in \mathbb{F}_2^n \times \mathbb{F}_2^m$ and $C \in \mathcal{B}(n,m)$ is an affine function. Using the fact that $C(x) \oplus C(x \oplus u) \oplus C(x \oplus v) \oplus C(x \oplus u \oplus v) = 0$, for all $x \in \mathbb{F}_2^n$ and $u, v \in \mathbb{F}_2^n$, we obtain the following relations:

$$\begin{aligned}
\mathtt{FBCT}_G(u,v) &= \#\{x \in \mathbb{F}_2^n :\ G(x) \oplus G(x \oplus u) \oplus G(x \oplus v) \oplus G(x \oplus u \oplus v) = 0\} \\
&= \#\{x \in \mathbb{F}_2^n :\ B(F(A(x) \oplus a)) \oplus B(F(A(x \oplus u) \oplus a)) \oplus B(F(A(x \oplus v) \oplus a)) \\
&\qquad\qquad \oplus B(F(A(x \oplus u \oplus v) \oplus a)) = 0\} \\
&= \#\{x \in \mathbb{F}_2^n :\ B(F(A(x) \oplus a) \oplus F(A(x \oplus u) \oplus a) \oplus F(A(x \oplus v) \oplus a) \\
&\qquad\qquad \oplus F(A(x \oplus u \oplus v) \oplus a)) = 0\} \\
&= \#\{x \in \mathbb{F}_2^n :\ F(A(x) \oplus a) \oplus F(A(x) \oplus u' \oplus a) \oplus F(A(x) \oplus v' \oplus a) \\
&\qquad\qquad \oplus F(A(x) \oplus u' \oplus v' \oplus a) = 0\} \\
&= \#\{y \in \mathbb{F}_2^n :\ F(y) \oplus F(y \oplus u') \oplus F(y \oplus v') \oplus F(y \oplus u' \oplus v') = 0\} \\
&= \mathtt{FBCT}_F(u',v'),
\end{aligned}$$

where $u' = A(u)$, $v' = A(v)$ and the new variable y corresponds to $A(x) \oplus a$. □

In particular, the F-boomerang uniformity is constant among S-boxes in the same linear, affine or extended-affine equivalence class.

CCZ Equivalence. The last equivalent relation we discuss here is the CCZ equivalence [CCZ98]. Concluding on this case is rather easy: it is known that every permutation is CCZ-equivalent to its inverse, and we show in next subsection that Feistel boomerang uniformity is not necessarily the same for an S-box and its inverse. Consequently, S-boxes that are CCZ equivalent might not share the same boomerang uniformity.

7.4.6 FBCT and Inversion

In the case of the BCT, it has been shown that the boomerang uniformity of S and its inverse are the same [BC18]. Before studying the case of the FBCT, let us recall that since we are looking at Feistel constructions the S-boxes at play do not have to be invertible (the most famous example in this category being the DES [DES77]).

For an invertible S-box, we can find some special cases for which the property is preserved, for instance for APN functions (since the inverse is also APN). Still, in the general case this property does not hold, and one example of this is for instance the 4-bit S-box SS_0 used in CLEFIA [SSA⁺07]:

$$\mathrm{SS}_0 = [\mathrm{0xe, 0x6, 0xc, 0xa, 0x8, 0x7, 0x2, 0xf, 0xb, 0x1, 0x4, 0x0, 0x5, 0x9, 0xd, 0x3}].$$

The F-boomerang uniformity of SS_0 is equal to 8, while the one of its inverse is 4.

7.4.7 Set-based Formulation of the FBCT

In this section, we identify the set[5]

$$\chi_{\mathtt{FBCT}}(\Delta_i, \nabla_o) = \{x \in \mathbb{F}_2^n | S(x) \oplus S(x \oplus \Delta_i) \oplus S(x \oplus \nabla_o) \oplus S(x \oplus \Delta_i \oplus \nabla_o) = 0\}$$

with the union for all $\delta \in \mathbb{F}_2^n$ of the intersection of $\chi_{DDT}(\Delta_i, \delta)$ and its coset $\chi_{DDT}(\Delta_i, \delta) \oplus \nabla_o$.

First, we recall the definition of $\chi_{DDT}(\Delta_i, \delta)$, a notion that has been introduced in [CLN⁺17] and used in the context of boomerang attacks in [SQH19] and that corresponds to the set of all $x \in \mathbb{F}_2^n$ that make a given S-box transition possible:

$$\chi_{DDT}(\Delta_i, \delta) = \{x \in \mathbb{F}_2^n | S(x) \oplus S(x \oplus \Delta_i) = \delta\}.$$

The alternative formulation is given in the following theorem:

Theorem 7.9. *For any $\Delta_i, \nabla_o \in \mathbb{F}_2^n$ and $S \in \mathcal{B}(n,n)$,*

$$\chi_{\mathtt{FBCT}}(\Delta_i, \nabla_o) = \bigcup_{\delta \in \mathbb{F}_2^n} (\chi_{DDT}(\Delta_i, \delta) \cap (\chi_{DDT}(\Delta_i, \delta) \oplus \nabla_o))$$

[5] We have $\#\chi_{\mathtt{FBCT}}(\Delta_i, \nabla_o) = \mathtt{FBCT}(\Delta_i, \nabla_o)$.

Proof. For any $\Delta_i, \nabla_o \in \mathbb{F}_2^n$,

$$\begin{aligned}
\chi_{\texttt{FBCT}}(\Delta_i, \nabla_o) =& \{x \in \mathbb{F}_2^n : \; S(x) \oplus S(x \oplus \Delta_i) \oplus S(x \oplus \nabla_o) \oplus S(x \oplus \Delta_i \oplus \nabla_o) = 0\} \\
=& \{x \in \mathbb{F}_2^n : \; S(x) \oplus S(x \oplus \Delta_i) = S(x \oplus \nabla_o) \oplus S(x \oplus \Delta_i \oplus \nabla_o)\} \\
=& \bigcup_{\delta \in \mathbb{F}_2^n} \{x \in \mathbb{F}_2^n : \; S(x) \oplus S(x \oplus \Delta_i) = S(x \oplus \nabla_o) \oplus S(x \oplus \Delta_i \oplus \nabla_o) = \delta\} \\
=& \bigcup_{\delta \in \mathbb{F}_2^n} \{x \in \mathbb{F}_2^n : \; S(x) \oplus S(x \oplus \Delta_i) = \delta\} \\
& \quad \cap \{x \in \mathbb{F}_2^n : \; S(x \oplus \nabla_o) \oplus S(x \oplus \Delta_i \oplus \nabla_o) = \delta\} \\
=& \bigcup_{\delta \in \mathbb{F}_2^n} \chi_{DDT}(\Delta_i, \delta) \cap \{\nabla_o \oplus x \in \mathbb{F}_2^n : \; S(x) \oplus S(x \oplus \Delta_i) = \delta\} \\
=& \bigcup_{\delta \in \mathbb{F}_2^n} \chi_{DDT}(\Delta_i, \delta) \cap (\chi_{DDT}(\Delta_i, \delta) \oplus \nabla_o).
\end{aligned}$$

□

Note here that for any fixed Δ_i the equality $\{x, x \oplus \Delta_i\} = \nabla_o \oplus \{x, x \oplus \Delta_i\}$ is satisfied for any x if and only if $\nabla_o = 0$ or $\Delta_i = \nabla_o$.

This reformulation leads to the following rewriting of the `FBCT` coefficient:

Corollary 7.10.

$$\texttt{FBCT}(\Delta_i, \nabla_o) = \sum_{\delta \in \mathbb{F}_2^n} \#(\chi_{DDT}(\Delta_i, \delta)) \cap (\chi_{DDT}(\Delta_i, \delta) \oplus \nabla_o)$$

Proof. This comes directly from the previous theorem by remarking that once Δ_i is fixed we have $\chi_{DDT}(\Delta_i, \delta) \cap \chi_{DDT}(\Delta_i, \delta') = \varnothing$ for all $\delta \neq \delta'$, which justifies that the unions in Theorem 7.9 are disjoint and hence that we have a sum. □

Let us again stress the parallel with a similar reformulation of the BCT:

Corollary 7.11 ([BC18])**.** *Let $S \in \mathcal{B}(n,n)$. We define $\mathcal{Y}_{DDT}$ as the set of all S-box outputs that make a given transition possible, that is: $\mathcal{Y}_{DDT}(\Delta_i, \delta) = \{S(x) \in \mathbb{F}_2^n | S(x) \oplus S(x \oplus \Delta_i) = \delta\}$. The expression of the BCT coefficient becomes:*

$$BCT(\Delta_i, \nabla_o) = \sum_{\delta \in \mathbb{F}_2^n} \#(\mathcal{Y}_{DDT}(\Delta_i, \delta)) \cap (\mathcal{Y}_{DDT}(\Delta_i, \delta) \oplus \nabla_o)$$

7.4.8 Comparison of the properties of the BCT and of the `FBCT`

We conclude this section by comparing in Table 7.3 the main properties explored by Boura and Canteaut [BC18] regarding the BCT with what we proved in the case of the `FBCT`.

Table 7.3: Comparison of the properties of the BCT and of the `FBCT` of n-bit functions.

Property	BCT	`FBCT`
Boomerang uniformity preserved under affine equivalence	yes	yes
Boomerang uniformity preserved under extended-affine equivalence	no	yes
Boomerang uniformity preserved under CCZ equivalence	no	no
Boomerang uniformity preserved under inversion	yes	no
Value of the boomerang uniformity of an APN function	2	0
Value of the boomerang uniformity of the inverse mapping (n even)	4 or 6	4

Note that another family of S-boxes was studied by Boura and Canteaut, namely the set of quadratic permutations. In the case of the `FBCT` this instance is rather easy to solve: for any non-zero $\Delta_i, \nabla_o \in \mathbb{F}_2^n$ with $\Delta_i \neq \nabla_o$, $D_{\Delta_i} D_{\nabla_o} S$ is constant. If this constant is not equal to zero we have that $\texttt{FBCT}(\Delta_i, \nabla_o) = 0$, otherwise $\texttt{FBCT}(\Delta_i, \nabla_o) = 2^n$. We can conclude that either the quadratic permutation is APN and then its Feistel boomerang uniformity is equal to 0, or the quadratic permutation is not APN (this is the case of all the quadratic permutations on an even number of variables) and then its Feistel boomerang uniformity is equal to 2^n.

Next, we provide a (rather intricate) formula linking the `FBCT` and the recently introduced *Differential-Linear Connectivity Table (DLCT)* [BDKW19]. We expect that other relations can be obtained.

A Relation Between the DLCT and the `FBCT`

As before, we consider n, m two positive integers and $S \in \mathcal{B}(n, m)$. The set of all non-zero elements of $\mathbb{F}_2^n$ is denoted by $\mathbb{F}_2^{n*}$. For x and $\lambda \in \mathbb{F}_2^n$ we denote by $\lambda \cdot x$ the canonical inner product.

The Differential-Linear Connectivity Table (DLCT) was introduced by Achiya Bar-On *et al.* in [BDKW19] and is defined as follows:

Definition 7.12 ([BDKW19])**.** *For a vectorial Boolean function $S : \mathbb{F}_2^n \to \mathbb{F}_2^m$, the differential-linear connectivity table (DLCT) of S is an $2^n \times 2^m$ table whose rows correspond to input differences to S and whose columns correspond to bit masks of outputs of S. The DLCT entry (Δ, λ), where $\Delta \in \mathbb{F}_2^n$ is a difference and $\lambda \in \mathbb{F}_2^m$ is a mask, is*

$$DLCT_S(\Delta, \lambda) = \#\{x \in \mathbb{F}_2^n : \ \lambda \cdot S(x) = \lambda \cdot S(x \oplus \Delta)\} - 2^{n-1}.$$

Recall that the autocorrelation of an n-variable Boolean function f at point $\Delta \in \mathbb{F}_2^n$, denoted $C_f(\Delta)$, is defined as:

$$C_f(\Delta) = \sum_{x \in \mathbb{F}_2^n} (-1)^{f(x) \oplus f(x \oplus \Delta)}.$$

It can be easily proven that $DLCT_S(\Delta, \lambda) = \frac{1}{2} \mathrm{C}_{\lambda \cdot S}(\Delta)$.

In the following, we consider a vectorial Boolean function S and derive a relation between its `FBCT` and the autocorrelation of its component functions. Using this relation, we provide a relation between the `FBCT` and the DLCT of S.

Theorem 7.13. *Let $S \in \mathcal{B}(n,m)$. Then for any non-zero $\Delta \in \mathbb{F}_2^{n*}$*

$$\sum_{\lambda \in \mathbb{F}_2^{m*}} C^2_{\lambda \cdot S}(\Delta) = 2^m \sum_{\nabla \in \mathbb{F}_2^{n*}:\ \nabla \neq \Delta} \mathtt{FBCT}_S(\Delta, \nabla) + 2^n(2^{m+1} - 2^n).$$

Proof. For any non-zero $\Delta \in \mathbb{F}_2^{n*}$, we have

$$\begin{aligned}\sum_{\lambda \in \mathbb{F}_2^{m*}} C^2_{\lambda \cdot S}(\Delta) &= \sum_{\lambda \in \mathbb{F}_2^{m*}} \Big(\sum_{x \in \mathbb{F}_2^n} (-1)^{\lambda \cdot S(x) \oplus \lambda \cdot S(x \oplus \Delta)}\Big)\Big(\sum_{y \in \mathbb{F}_2^n} (-1)^{\lambda \cdot S(y) \oplus \lambda \cdot S(y \oplus \Delta)}\Big) \\ &= \sum_{x \in \mathbb{F}_2^n} \sum_{y \in \mathbb{F}_2^n} \sum_{\lambda \in \mathbb{F}_2^{m*}} (-1)^{\lambda \cdot (S(x) \oplus S(x \oplus \Delta) \oplus S(y) \oplus S(y \oplus \Delta))} \\ &= \sum_{x \in \mathbb{F}_2^n} \sum_{\nabla \in \mathbb{F}_2^n} \Big(\sum_{\lambda \in \mathbb{F}_2^m} (-1)^{\lambda \cdot (S(x) \oplus S(x \oplus \Delta) \oplus S(x \oplus \nabla) \oplus S(x \oplus \Delta \oplus \nabla))} - 1\Big) \\ &= 2^m \sum_{\nabla \in \mathbb{F}_2^n} \#\{x \in \mathbb{F}_2^n : S(x) \oplus S(x \oplus \Delta) \oplus S(x \oplus \nabla) \oplus S(x \oplus \Delta \oplus \nabla) = 0\} - 2^{2n} \\ &= 2^m \sum_{\nabla \in \mathbb{F}_2^{n*}:\ \nabla \neq \Delta} \mathtt{FBCT}_S(\Delta, \nabla) + 2^n(2^{m+1} - 2^n).\end{aligned}$$

□

From this theorem and the relation between the DLCT and the autocorrelation of the component functions of S, we directly deduce the following corollary:

Corollary 7.14. *Let $S \in \mathcal{B}(n,m)$. Then for any non-zero $\Delta \in \mathbb{F}_2^{n*}$*

$$\sum_{\lambda \in \mathbb{F}_2^{m*}} DLCT^2_S(\Delta, \lambda) = 2^{m-2} \sum_{\nabla \in \mathbb{F}_2^{n*}:\ \nabla \neq \Delta} \mathtt{FBCT}_S(\Delta, \nabla) + 2^n(2^{m-1} - 2^{n-2}).$$

7.5 Extending our Analysis to Two Rounds

Similarly to what has been done in [WP19, SQH19] for SPN constructions, this section discusses the probability of a boomerang switch E_m that covers two rounds.

7.5.1 The Feistel counterpart of the BDT

When studying how to extend the BCT theory to boomerang switches on more rounds, Wang and Peyrin [WP19] introduced the BDT (standing for *Boomerang Difference Table*), a variant of the BCT with one supplementary variable fixed, namely the S-box output difference:

Definition 7.15 (Boomerang Difference Table [WP19])**.** *Let S be an invertible function in $\mathbb{F}_2^n$, and $(\Delta_i, \delta, \nabla_o)$ be elements of $(\mathbb{F}_2^n)^3$. The boomerang difference table (BDT) of S is a three-dimensional table, in which the entry for $(\Delta_i, \delta, \nabla_o)$ is computed by:*

$$\begin{aligned}BDT(\Delta_i, \delta, \nabla_o) = \#\{x \in \mathbb{F}_2^n | S^{-1}(S(x) \oplus \nabla_o) \oplus S^{-1}(S(x \oplus \Delta_i) \oplus \nabla_o) = \Delta_i, \\ S(x) \oplus S(x \oplus \Delta_i) = \delta\}.\end{aligned}$$

As we show next, the counterpart of this table for the Feistel case turns out to be useful to study a switch over two rounds. Following the idea of [WP19], we define it as follows (it can be visualized in Figure 7.9):

Definition 7.16 (FBDT). *Let S be a function from $\mathbb{F}_2^n$ to itself, and $(\Delta_i, \delta, \nabla_o)$ be elements of $(\mathbb{F}_2^n)^3$. The Feistel boomerang difference table (*FBDT*) of S is a three-dimensional table, in which the entry for $(\Delta_i, \delta, \nabla_o)$ is computed by:*

$$\texttt{FBDT}(\Delta_i, \delta, \nabla_o) = \#\{x \in \mathbb{F}_2^n | S(x) \oplus S(x \oplus \Delta_i) \oplus S(x \oplus \nabla_o) \oplus S(x \oplus \Delta_i \oplus \nabla_o) = 0, \\ S(x) \oplus S(x \oplus \Delta_i) = \delta\}.$$

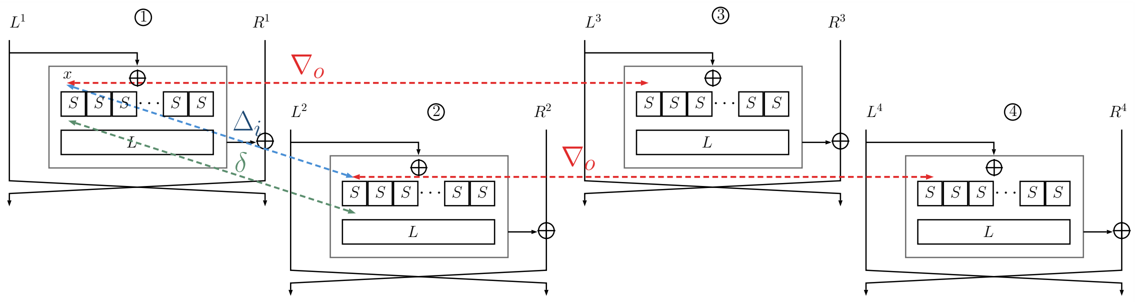


Figure 7.9: View of the parameters of the FBDT: Δ_i is the input difference and δ is the output difference of S when looking at the difference between state ① and ②. ∇_o is the input difference of the same S-box S when looking at the difference between state ① and ③ (which is the same as the one between state ② and ④).

Given the discussion made in Section 7.4.7, we can rewrite the FBDT as:

$$\texttt{FBDT}(\Delta_i, \delta, \nabla_o) = \#\{(\chi_{DDT}(\Delta_i, \delta)) \cap (\chi_{DDT}(\Delta_i, \delta) \oplus \nabla_o)\}.$$

This is rather straightforward to see that the FBDT follows similar relations as the BDT does, namely:

Property 7.17 (Relations between the DDT, FBCT and FBDT).

1. *$DDT(\Delta_i, \delta) = \texttt{FBDT}(\Delta_i, \delta, 0) = \texttt{FBDT}(\Delta_i, \delta, \Delta_i)$ and in the general case $DDT(\Delta_i, \delta) \geq \texttt{FBDT}(\Delta_i, \delta, \nabla_o)$.*
2. *$\texttt{FBCT}(\Delta_i, \nabla_o) = \sum_{\delta=0}^{2^n-1} \texttt{FBDT}(\Delta_i, \delta, \nabla_o)$.*
3. *$\texttt{FBDT}(0, 0, \nabla_o) = 2^n$.*

7.5.2 Probability of a 2-round Boomerang Switch

The theorem we discuss next gives the probability that a boomerang comes back over 2 rounds of a classic Feistel cipher, that is a balanced one with 2 branches. We consider that the input difference between state ① and ② is (Δ_i^L, Δ_i^R), that the output difference between state ② and ④ and ① and ③ is equal to (∇_o^L, ∇_o^R), and we want that the input difference between state ③ and ④ is again (Δ_i^L, Δ_i^R).

Again, we consider a very generic case where the round function is composed of one S-box layer made of t parallel n-bit S-boxes and of some linear or affine operations, which implies in particular that if the input difference of one round is known together with the output difference of the S-box layer, then the difference at the input of the next round S-box layer can be computed. To keep our explanation as generic as possible we introduce the following notations, that can be visualized in Figure 7.10:

- Δ_i represents the difference at the input of the first round S-box layer, between state ① and ②. It is fixed to a certain value since it can be deduced from the first round input difference Δ_i^L.
- δ denotes the corresponding output difference of this S-box layer, but is not specified.
- Δ_i' corresponds to the difference at the input of the second S-box layer (again with respect to state ① and ②). Its value is deduced from δ and from Δ_i^R.
- In a similar way, the difference at the input of the second round S-box layer, between state ② and ④ is set to a certain value denoted ∇_o, deduced from ∇_o^R.
- The corresponding output difference is denoted α, but again is not fixed.
- ∇_o' represents the input difference of the first round S-box layer for these states, and is computed from ∇_o^L and α.

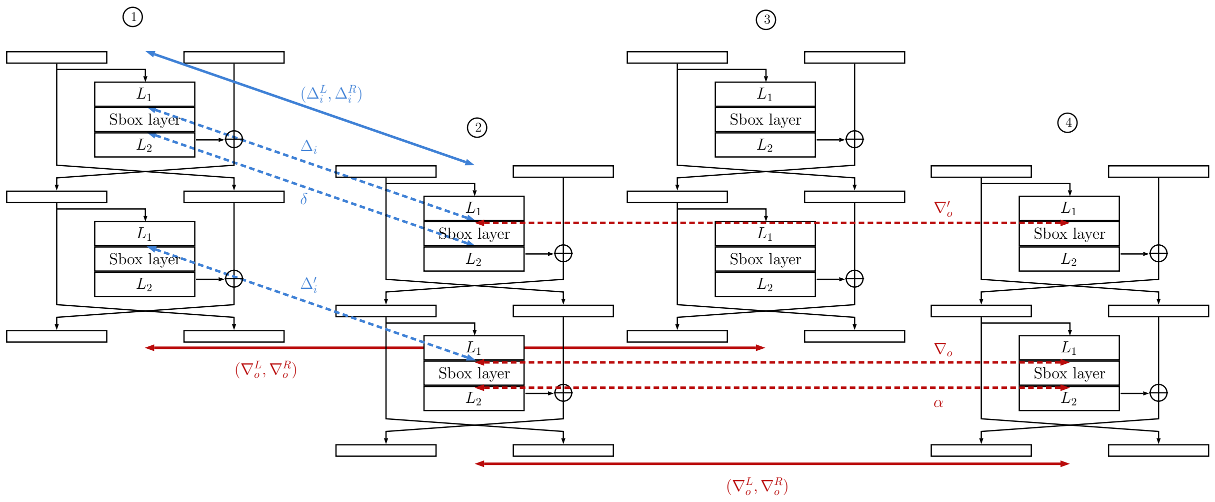


Figure 7.10: Boomerang Switch over two rounds of a balanced Feistel with two branches. The differences denoted with straight lines are imposed and fixed.

Given this notation we can find a formula for the probability of a 2-round boomerang switch over a Feistel, see Theorem 7.18. Note that to simplify its writing we extended the definition of the FBDT to the case of the S-box layer (instead of one S-box only). Naturally, this simply corresponds to the product of the FBDT of each S-box that composes the S-box layer.

Theorem 7.18 (Probability of a 2-round Switch)**.** *With the previous notation, the probability that a boomerang comes back over 2 rounds is equal to:*

$$2^{-2tn} \times \sum_{0\leq\delta,\alpha<2^n} \mathtt{FBDT}(\Delta_i, \delta, \nabla'_o) \times \mathtt{FBDT}(\nabla_o, \alpha, \Delta'_i). \tag{7.3}$$

Proof. In order to cover most constructions, in what follows we consider a Feistel cipher as depicted in Figure 7.10, that is with a round function made of one linear (or affine) layer L_1, followed by one S-box layer of t n-bit S-boxes and again a linear (or affine) layer L_2.

We start by observing that if the second round S-box layer output difference between state ② and ④ is equal to a given value α then the same difference is required between state ① and ③ for the boomerang to return.

Denote by α' the second S-box layer output difference between state ① and ③. Given that the output difference between ① and ③ and ② and ④ is equal to (∇_o^L, ∇_o^R) we deduce that the input difference in the left branch between states ① and ③ and ② and ④ are respectively equal to $\nabla_o^L \oplus L_2(\alpha')$ and $\nabla_o^L \oplus L_2(\alpha)$. The input difference between the left branches of state ① and ② is equal to Δ_i^L so we deduce that the left branch difference between state ③ and ④ is equal to: $\Delta_i^L \oplus \nabla_o^L \oplus L_2(\alpha') \oplus \nabla_o^L \oplus L_2(\alpha) = \Delta_i^L \oplus L_2(\alpha') \oplus L_2(\alpha)$. For the boomerang to return this has to be equal to Δ_i^L, which proves that we must have $\alpha' = \alpha$.

We now demonstrate the formula by first looking at the case where the values of α and δ are fixed. The theorem is deduced by summing over all their possible values.

We focus on the second round of the switch, and more precisely on the difference between state ② and ④. To obtain the required output difference, the S-box layer must transition from $\nabla_o = L_1(\nabla_o^R)$ to α, an event that is of probability[a]:

$$2^{-nt} \times \mathtt{DDT}(\nabla_o, \alpha).$$

If we denote by X the input value of the second round S-box layer of state ②, We know that the corresponding value of state ① has to be equal to $X \oplus \Delta'_i$, value that should also allow the transition from ∇_o to α according to the previous discussion. The probability that it is the case is:

$$\frac{\#\chi_{DDT}(\nabla_o, \alpha) \cap (\chi_{DDT}(\nabla_o, \alpha) \oplus \Delta'_i)}{\#\chi_{DDT}(\nabla_o, \alpha)}.$$

Assuming that the previous conditions are fulfilled, the boomerang returns in the first round if the S-box layer transitions from Δ_i to δ given that the input difference of this S-box layer between state ② and ④ and ① and ③ is equal to ∇'_o. The probability of this event is $\mathtt{FBDT}(\Delta_i, \delta, \nabla'_o) \times 2^{-nt}$.

Putting things together, we obtain

$$2^{-2tn} \times \sum_{0\leq\delta,\alpha<2^n} \mathtt{FBDT}(\Delta_i, \delta, \nabla'_o) \times \mathtt{DDT}(\nabla_o, \alpha) \times \frac{\#\chi_{DDT}(\nabla_o, \alpha) \cap (\chi_{DDT}(\nabla_o, \alpha) \oplus \Delta'_i)}{\#\chi_{DDT}(\nabla_o, \alpha)}$$

Given that $\mathtt{DDT}(\nabla_o, \alpha) = \#\chi_{DDT}(\nabla_o, \alpha)$ and that $\#(\chi_{DDT}(\nabla_o, \alpha) \cap (\chi_{DDT}(\nabla_o, \alpha) \oplus \Delta'_i)) = \mathtt{FBDT}(\nabla_o, \alpha, \Delta'_i)$, we obtain the required expression.

□

[a] We again make the shortcut of considering the S-box layer instead of each individual S-box.

Note that our formula is very reminiscent of what is used in the SPN case, as Wang and

Peyrin [WP19] proposed to use the product of the BDT and BDT' coefficients to cover the case of a 2-round switch where the same S-box is active with respect to E_0 and E_1. As a side note, we also remark here that the somewhat more intricate formulation proposed by Song *et al.* can be rewritten as the product of the BDT and BDT' in the case of 2 rounds, as in particular the $\mathcal{D}_{BCT}$ coefficient of [SQH19] is in fact equal to the BDT' coefficient.

Example of a 2-round Switch on LBlock

This section shows a concrete example of how the 2-round formula given by Equation (7.3) can applied to a cipher, namely LBLOCK. We consider a 2-round boomerang switch that is deduced from the proposed boomerang distinguisher of the paper by Chen and Miyaji [CM13].

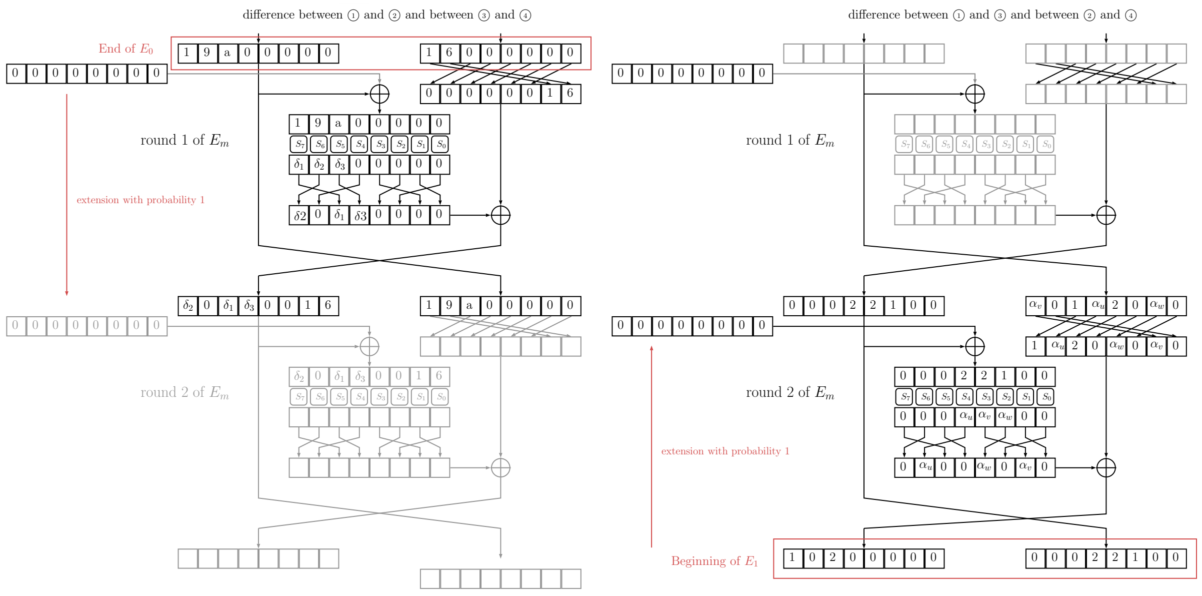

difference between ① and ② and between ③ and ④
difference between ① and ③ and between ② and ④
End of E_0
round 1 of E_m
extension with probability 1
round 2 of E_m
extension with probability 1
Beginning of E_1

Figure 7.11: Concrete 2-round boomerang switch on LBLOCK, derived from [CM13].

The switch and the notation used below are represented in Figure 7.11. By careful identification of the differences at play, Equation (7.3) gives:

$$
\begin{aligned}
P = 2^{-2\times 8\times 4} \times \sum_{0\leq\delta,\alpha<2^n} \quad & \mathtt{FBDT}_{S_7}(0\text{x}1,\delta_1,\alpha_v) \times \mathtt{FBDT}_{S_7}(0,0,\delta_2) \times \\
& \mathtt{FBDT}_{S_6}(0\text{x}9,\delta_2,0) \times \mathtt{FBDT}_{S_6}(0,0,0) \times \\
& \mathtt{FBDT}_{S_5}(0\text{x}a,\delta_3,0\text{x}1) \times \mathtt{FBDT}_{S_5}(0,0,\delta_1) \times \\
& \mathtt{FBDT}_{S_4}(0,0,\alpha_u) \times \mathtt{FBDT}_{S_4}(0\text{x}2,\alpha_u,\delta_3) \times \\
& \mathtt{FBDT}_{S_3}(0,0,0\text{x}2) \times \mathtt{FBDT}_{S_3}(0\text{x}2,\alpha_v,0) \times \\
& \mathtt{FBDT}_{S_2}(0,0,0) \times \mathtt{FBDT}_{S_2}(0\text{x}1,\alpha_w,0) \times \\
& \mathtt{FBDT}_{S_1}(0,0,\alpha_w) \times \mathtt{FBDT}_{S_1}(0,0,0\text{x}1) \times \\
& \mathtt{FBDT}_{S_0}(0,0,0) \times \mathtt{FBDT}_{S_0}(0,0,0\text{x}6)
\end{aligned}
$$

Using the properties of the FBDT, this can be simplified into:

$$P = 2^{-6\times 4} \times \sum_{0\leq \delta,\alpha<2^n} \texttt{FBDT}_{S_7}(0\text{x}1, \delta_1, \alpha_v) \times DDT_{S_6}(0\text{x}9, \delta_2) \times \texttt{FBDT}_{S_5}(0\text{x}a, \delta_3, 0\text{x}1)$$
$$\times \texttt{FBDT}_{S_4}(0\text{x}2, \alpha_u, \delta_3) \times DDT_{S_3}(0\text{x}2, \alpha_v) \times DDT_{S_2}(0\text{x}1, \alpha_w)$$

Referring to the DDT and FBDT we found that it is equal to $2^{-24} \times 2^{19} = 2^{-5}$, which closely matches what we found experimentally (we obtained a probability of $2^{-4.998}$ when doing 2^{20} tests corresponding to 2^{10} keys with 2^{10} messages each).

7.6 Generic Formula for a Feistel Boomerang Switch over Multiple Rounds

To obtain an accurate estimation of the probability of a boomerang distinguisher, an attacker has to correctly evaluate the size of E_m, that is the number of middle rounds for which there exists a dependency between the characteristic on E_0 and the one on E_1. Once this is done, the formula introduced with the sandwich attack theory [DKS10] can be applied and the value of p^2q^2r (where r is the probability of E_m, p the one of E_0 and q the one of E_1) gives a good estimate (under the usual assumptions).

The problem of evaluating the size of E_m has already been discussed in two papers in the case of SPN ciphers: by Song *et al.* in [SQH19] and by Wang and Peyrin in [WP19]. The algorithm proposed in [SQH19] (that we recall in Algorithm 19) is rather natural: additional rounds are added to E_m as long as the probability of the newly added round is higher than the probability that would have been obtained if they were no dependencies. Since this technique directly applies to boomerang distinguishers on Feistel constructions we do not elaborate more on this.

Algorithm 19 Song *et al.*'s [SQH19] algorithm to compute the size of E_m.

```
1. Extend both $E_0$ and $E_1$ with probability 1.
2. Initialize $E_m$ with the last round of $E_0$ and the first round of $E_1$.
3. Prepend one round to $E_m$
   (a) Check whether the lower crossing differences for the newly added round are distributed
       uniformly. If they are, peel off the first round of $E_m$ and go to step 4.
   (b) Go to step 3.
4. Append one more round to $E_m$
   (a) Check whether the upper crossing differences for the newly added round are distributed
       uniformly. If they are, peel off the last round of $E_m$ and go to step 5.
   (b) Go to step 4.
5. Compute the probability of $E_m$.
```

The remaining problem in the case of Feistel ciphers is to compute the probability of a boomerang switch over more than 2 rounds. We address this now, with a setting and notation given in Figure 7.12 and that is a direct generalization of the one in Figure 7.10.

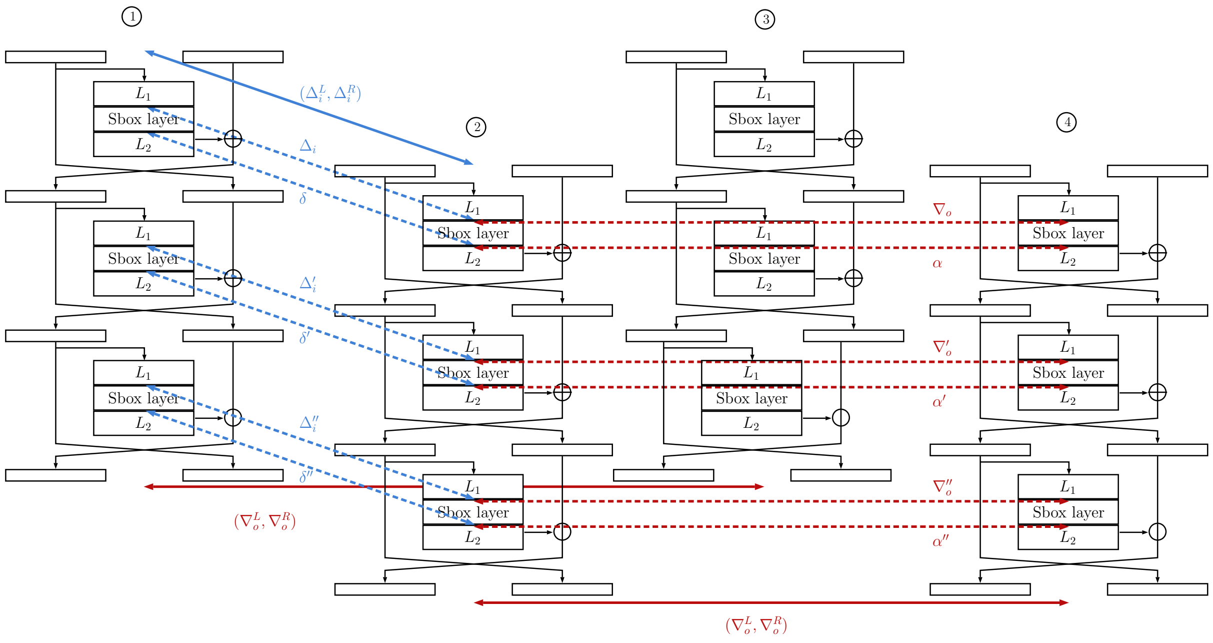


Figure 7.12: Setting for a boomerang Switch over more than two rounds of a balanced Feistel with two branches. The differences denoted with straight lines are imposed and fixed.

As depicted in the figure, we introduce new variables to represent all the intermediate differences. As we did when discussing the 2-round switch, the idea will be to iterate over all the possible values for these, to compute the probability of the obtained settings and finally to sum together the probabilities.

We introduce a coefficient that corresponds to the situation where an active S-box in E_0 is in front of an active S-box in E_1, and for which both S-box outputs (when looking at state ① and ② and state ② and ④) are fixed. We obtain the following formula:

Definition 7.19 (FBET). *Let S be a function from $\mathbb{F}_2^n$, and $(\Delta_i, \delta, \nabla_o, \alpha)$ be elements of $(\mathbb{F}_2^n)^4$. The Feistel boomerang extended table (FBET) of S is a four-dimensional table, in which the entry for $(\Delta_i, \delta, \nabla_o, \alpha)$ is computed by:*

$$\begin{aligned}\mathtt{FBET}(\Delta_i, \delta, \nabla_o, \alpha) = \#\{x \in \mathbb{F}_2^n | S(x) \oplus S(x \oplus \Delta_i) \oplus S(x \oplus \nabla_o) \oplus S(x \oplus \Delta_i \oplus \nabla_o) = 0,\\ S(x) \oplus S(x \oplus \Delta_i) = \delta,\\ S(x \oplus \Delta_i) \oplus S(x \oplus \Delta_i \oplus \nabla_o) = \alpha\}.\end{aligned}$$

The probability of a switch is then estimated to be[6] the sum over all the possible intermediate differences of the product of the FBET coefficient (divided by 2^n) of each S-box. For instance, the probability of the 3-round boomerang switch depicted in Figure 7.10 can be approximated by:

$$2^{-3tn} \sum_{0 \le \delta, \alpha, \delta', \alpha', \delta'', \alpha'' < 2^n} \mathtt{FBET}(\Delta_i, \delta, \nabla_o, \alpha) \times \mathtt{FBET}(\Delta_i', \delta', \nabla_o', \alpha') \times \mathtt{FBET}(\Delta_i'', \delta'', \nabla_o'', \alpha'')$$

[6]Note that this approximation considers that the same characteristic is followed between state ① and ② and between state ③ and ④. For 3 rounds and more it is not apparent that this is always the only possible case.

where again by abuse of notation the FBET coefficient is the one of the full S-layer, but should be replaced by the ones of the individual S-boxes. Note that Δ_i and ∇''_o are determined by (Δ_i^L, Δ_i^R) and (∇_o^L, ∇_o^R), the input and output differences of the switch. Also, the values of Δ'_i, Δ''_i, ∇_o and ∇'_o are deduced from the other parameters on which we iterate (for instance $\Delta'_i = L_1(L_2(\delta) \oplus \Delta_i^R)$).

As we show in the following property, the obtained formula can be simplified when we sum coefficients over all the possible values of some variables. Further simplifications are obtained with Property 7.17.

Property 7.20 (Relations between the FBET and the previous tables)**.**

$$\sum_{0 \le \delta < 2^n} \mathtt{FBET}(\Delta_i, \delta, \nabla_o, \alpha) = \mathtt{FBDT}(\nabla_o, \alpha, \Delta_i).$$

$$\sum_{0 \le \alpha < 2^n} \mathtt{FBET}(\Delta_i, \delta, \nabla_o, \alpha) = \mathtt{FBDT}(\Delta_i, \delta, \nabla_o).$$

$$\mathtt{FBET}(0, 0, \nabla_o, \alpha) = \mathtt{FBET}(\nabla_o, \alpha, 0, 0) = \mathtt{DDT}(\nabla_o, \alpha).$$

It is rather easy to show that the FBET view covers the previous formula for the 2-round switch (given in Theorem 7.18): we use the notation of Figure 7.10 and additionally denote by δ' the output difference between state ① and ② of the second-round S-layer, and by α' the output difference between state ② and ④ of the first-round S-layer. The sum we have to compute is:

$$2^{-2tn} \sum_{0 \le \delta, \alpha', \delta', \alpha < 2^n} \mathtt{FBET}(\Delta_i, \delta, \nabla'_o, \alpha') \times \mathtt{FBET}(\Delta'_i, \delta', \nabla_o, \alpha).$$

Since α' and δ' have no impact on the other values we can rewrite the previous sum as:

$$\begin{aligned}
& 2^{-2tn} \sum_{0 \le \delta, \alpha', \alpha < 2^n} (\mathtt{FBET}(\Delta_i, \delta, \nabla'_o, \alpha') \times \sum_{0 \le \delta' < 2^n} \mathtt{FBET}(\Delta'_i, \delta', \nabla_o, \alpha)) \\
= \; & 2^{-2tn} \sum_{0 \le \delta, \alpha', \alpha < 2^n} (\mathtt{FBET}(\Delta_i, \delta, \nabla'_o, \alpha') \times \mathtt{FBDT}(\nabla_o, \alpha, \Delta'_i)) \\
= \; & 2^{-2tn} \sum_{0 \le \delta, \alpha < 2^n} (\mathtt{FBDT}(\nabla_o, \alpha, \Delta'_i) \times \sum_{0 \le \alpha' < 2^n} \mathtt{FBET}(\Delta_i, \delta, \nabla'_o, \alpha')) \\
= \; & 2^{-2tn} \sum_{0 \le \delta, \alpha < 2^n} (\mathtt{FBDT}(\nabla_o, \alpha, \Delta'_i) \times \mathtt{FBDT}(\Delta_i, \delta, \nabla'_o)).
\end{aligned}$$

In a similar way, if we focus on one round only, we have to compute

$$2^{-tn} \sum_{0 \le \delta, \alpha < 2^n} \mathtt{FBET}(\Delta_i, \delta, \nabla_o, \alpha).$$

Since both δ and α have no impact on the other variables it can be rewritten as:

$$2^{-tn} \sum_{0 \le \delta < 2^n} \mathtt{FBDT}(\Delta_i, \delta, \nabla_o) = 2^{-tn} \mathtt{FBCT}(\Delta_i, \nabla_o).$$

So the FBET coefficient allows to recover our previous formulas.

Note that when looking at a switch covering many rounds the application of this formula may require too much time if many S-boxes are involved, so it might be preferable to evaluate the probability of E_m experimentally.

Short Discussion on the SPN Case. While we focused on the Feistel case, it seems that a similar technique can be used to get the probability of a multiple-round boomerang switch on an SPN cipher. In particular, the counterpart of the FBET would be:

$$\begin{aligned} BET(\Delta_i, \delta, \nabla_o, \alpha) = \#\{x \in \mathbb{F}_2^n | S^{-1}(S(x) \oplus \nabla_o) \oplus S^{-1}(S(x \oplus \Delta_i) \oplus \nabla_o) &= \Delta_i, \\ S(x) \oplus S(x \oplus \Delta_i) &= \delta, \\ x \oplus S^{-1}(S(x) \oplus \nabla_o) &= \alpha\}. \end{aligned}$$

and we have the following direct properties:

Property 7.21 (Relation between the BET and the previous tables)**.**

$$\begin{aligned} \sum_{0 \le \alpha < 2^n} BET(\Delta_i, \delta, \nabla_o, \alpha) &= BDT(\Delta_i, \delta, \nabla_o), \\ \sum_{0 \le \delta < 2^n} BET(\Delta_i, \delta, \nabla_o, \alpha) &= BDT'(\nabla_o, \alpha, \Delta_i) = \mathcal{D}_{BCT}(\Delta_i, \nabla_o, \alpha) \\ \sum_{0 \le \alpha, \delta < 2^n} BET(\Delta_i, \delta, \nabla_o, \alpha) &= BCT(\Delta_i, \nabla_o) \end{aligned}$$

Our bet is that it provides a generic formula covering the previous particular cases discussed in [SQH19] and [WP19].

7.7 Application to LBlock-s

We propose here to study the case of LBLOCK-S, the Feistel cipher used in LAC, in order to illustrate the way our formula can be used to estimate the probability of a boomerang distinguisher.

LAC was a first-round candidate to the CAESAR competition submitted by Lei Zhang *et al.* [ZWW⁺14]. It is a lightweight authenticated encryption scheme that relies on a modified version of LBLOCK called LBLOCK-S. In this version, the 10 different 4-bit S-boxes are replaced with one unique S-box, which corresponds to the one called S_0 in LBLOCK. The block cipher also includes a modified key schedule algorithm that we do not detail here since it plays no role in the following discussion. The LAC algorithm uses both full 32-round LBLOCK-S as well as a round-reduced LBLOCK-S iterating 16 rounds.

In this section, we evaluate with the p^2q^2r formula the probability of a 16-round boomerang distinguisher on LBLOCK-S when the size of E_m varies from 2 to 8 rounds. We found out that when E_m covers 8 rounds the expected probability of the resulting distinguisher is $2^{-56.14}$.

This value is higher than the probability of the distinguisher that was proposed by Leurent in [Leu16]. In this paper, the author showed the existence of collections of differential characteristics with probability as high as $2^{-61.52}$. Still, our distinguisher cannot be used for forgery contrary to what is done in [Leu16].

7.7.1 Finding the Best 7-round Differential Characteristics for E_0 and E_1

As a starting point, we look at the setting where E_m covers 2 rounds and search the best characteristics over 7 rounds for E_0 and E_1. To find these, we use the two-step strategy described in [GLMS18]:

- In the first step, we abstract all the nibble differences by Boolean variables (if a nibble is active then its associated Boolean value is 1, else it is 0) and we look for the truncated differentials with the minimum number of active S-boxes. We implement this step using a high-level modeling language called MiniZinc [NSB+07]. MiniZinc models are translated into a simple subset of MiniZinc called FlatZinc, using a compiler provided by MiniZinc. Most existing constraint programming solvers (including SAT solvers and MILP solvers) have developed FlatZinc interfaces (there are currently fifteen solvers with FlatZinc interfaces). Using the PICAT SAT solver we found 8 possible optimal truncated differential characteristics that are valid for both E_0 and E_1.

- In the second step, we look for the best differential characteristics (in terms of probability) that follow the previous truncated differential paths. To do so, we use the constraint programming language Choco [PFL16]. For each possible truncated differential characteristics on 7 rounds we obtain 2766 solutions with an optimal probability equal to 2^{-16}. We tried several combinations and picked the one that gave the best probability for the 2-round E_m. We present it in Table 7.4.

	Differential characteristic used in E_0		Differential characteristic used in E_1
Input	20400000 00001460	Output r9	00004020 41000006
Output r1	00006000 20400000	Output r10	00000600 00004020
Output r2	40000000 00006000	Output r11	00400000 00000600
Output r3	00000000 40000000	Output r12	00000000 00400000
Output r4	00000040 00000000	Output r13	40000000 00000000
Output r5	00000004 00000040	Output r14	00400000 40000000
Output r6	00004400 00000004	Output r15	00060040 00400000
Output r7	00004440 00004400	Output r16	42000004 00060040

Table 7.4: The two differential characteristics on 7 rounds of E_0 and of E_1 in hexadecimal notations.

7.7.2 Choosing a Switch E_m

To obtain an accurate evaluation of the boomerang distinguisher, we evaluate the size and probability of E_m with the algorithm recalled in Algorithm 19. When E_m covers few rounds we were able to apply our formulas to compute its probability but we then switched to experiments to avoid intricate expressions with many parameters. As detailed in Table 7.5, we were able to apply the algorithm for an E_m covering up to 8 rounds, thus obtaining an estimation of the probability of the distinguisher of $2^{-56.14}$. Our observation is that E_m covers more than 8 rounds, but we were limited by computational power to get its exact size.

Example of Instantiation of the Generic Formula

As an example of the application of the switch formulas proposed in this work, we detail here how to compute the probability of the 3-round switch on LBLOCK-S with the parameters provided in Table 7.5 and depicted in Figure 7.13:

E_m	α	δ	theoretical r	practical r	p^2q^2r
0 rounds	-	-	-	-	2^{-88}
2 rounds	$(0x00004440, 0x00004400)$	$(0x00004020, 0x41000006)$	$2^{-3.09}$	$2^{-3.09}$	$2^{-67.09}$
3 rounds	$(0x00004440, 0x00004400)$	$(0x00000600, 0x00004020)$	$2^{-6.80}$	$2^{-6.80}$	$2^{-62.8}$
4 rounds	$(0x00004400, 0x00000004)$	$(0x00000600, 0x00004020)$	n/a	$2^{-14.10}$	$2^{-62.1}$
6 rounds	$(0x00000004, 0x00000040)$	$(0x00400000, 0x00000600)$	n/a	$2^{-19.04}$	$2^{-59.04}$
8 rounds	$(0x00000040, 0x00000000)$	$(0x00000000, 0x00400000)$	n/a	$2^{-24.14}$	$2^{-56.14}$

Table 7.5: Theoretical and practical values of r for various sizes of E_m and corresponding probability of the 16-round distinguisher when applying the p^2q^2r formula. We detail the theoretical computation for 3 rounds in Section 7.7.2.

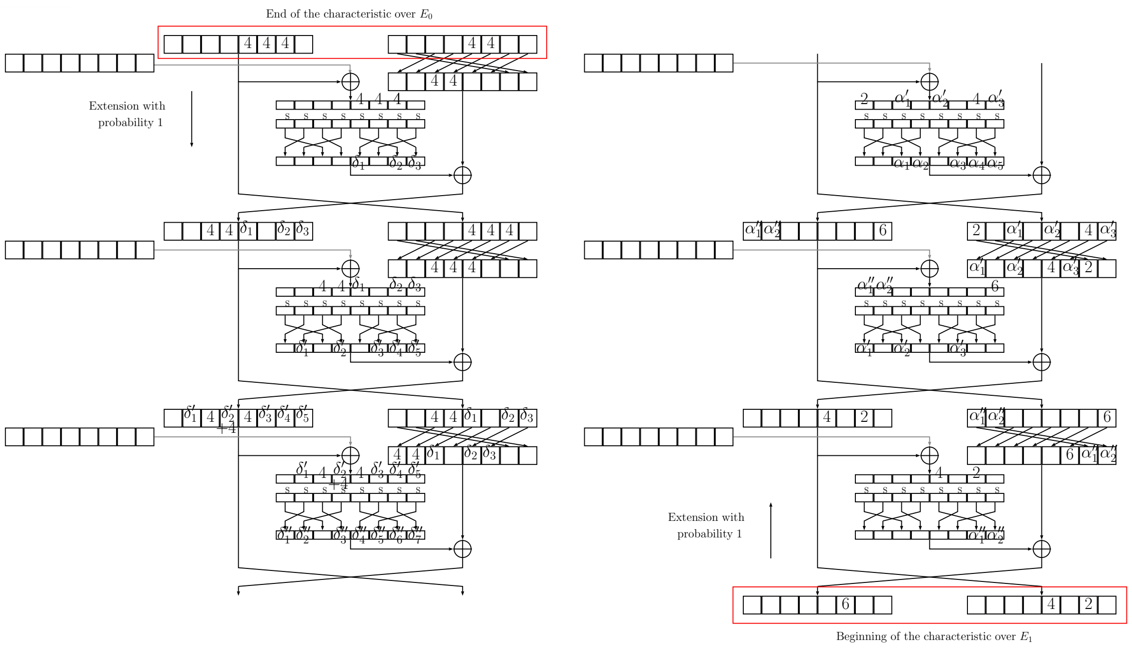

End of the characteristic over E_0
Extension with probability 1
Extension with probability 1
Beginning of the characteristic over E_1

Figure 7.13: Setting for the switch over three rounds of LBlock-s.

According to our theory, an approximation of the 3-round switch is given by the sum over all the possible intermediate differences (the δ and α in the figure) of the product of the FBET coefficients of each S-box, each divided by 2^n. Since $\texttt{FBET}(0,0,0,0) = 2^n$ and $\texttt{FBET}(0,0,\nabla_o,\alpha) = \texttt{FBET}(\nabla_o,\alpha,0,0) = \texttt{DDT}(\nabla_o,\alpha)$, we expect only 5 FBET coefficients corresponding to the S-boxes that are active on both sides, and 15 DDT coefficients corresponding to S-boxes that are only active in one side.

An additional simplification comes from the following fact: the active S-boxes in the first round of the right part of the figure and the ones in the last round of the left part of the figure have an output that is free of constraints, so they don't have an impact on the probability we are computing (this comes from the fact that $\frac{1}{2^n}\sum_\alpha \texttt{DDT}(\gamma,\alpha) = 1$).

Putting things together, we obtain the following expression, where the sum is over all the involved variables:

$$\begin{aligned} r &= 2^{-4\times 12} \sum \texttt{FBET}(4,\delta_2,\alpha_2',\alpha_4)\cdot\texttt{DDT}(4,\delta_1)\cdot\texttt{FBET}(4,\delta_3,4,\alpha_5)\\ &\cdot\texttt{DDT}(4,\delta_2')\cdot\texttt{DDT}(4,\delta_1')\cdot\texttt{DDT}(\delta_1,\delta_4')\cdot\texttt{DDT}(\delta_2,\delta_5')\cdot\texttt{FBET}(\delta_3,\delta_3',6,\alpha_3')\\ &\cdot\texttt{DDT}(\alpha_1'',\alpha_2')\cdot\texttt{DDT}(\alpha_2'',\alpha_1')\\ &\cdot\texttt{FBET}(4,\delta_6'',4,\alpha_1'')\cdot\texttt{FBET}(\delta_4',\delta_7'',2,\alpha_2''). \end{aligned}$$

We can further simplify this expression by using the relations between the tables discussed in this work. It gives:

$$\begin{aligned} r &= 2^{-4\times 8} \sum \texttt{FBCT}(4,\alpha_2')\cdot\texttt{DDT}(4,\delta_1)\cdot\texttt{DDT}(4,\delta_3)\\ &\cdot\texttt{DDT}(\delta_1,\delta_4')\cdot\texttt{FBCT}(\delta_3,6)\cdot\texttt{DDT}(\alpha_1'',\alpha_2')\\ &\cdot\texttt{DDT}(4,\alpha_1'')\cdot\texttt{FBCT}(2,\delta_4') \end{aligned}$$

We computed this sum and we obtained $r = \frac{38338560}{(2^4)^8} = 2^{-6.807}$, which confirms the experiment reported in Table 7.5.

7.7.3 Deriving a Boomerang Distinguisher

The previous discussion indicates that the 16-round boomerang distinguisher we are looking at has a probability higher than $2^{-56.14}$. It can be used as follows:

The attacker randomly chooses M_i^1 $(0 \le i < m)$ and compute $M_i^2 = M_i^1 \oplus \alpha$ with $\alpha = (20400000, 00001460)$. She encrypts these plaintexts over 16 rounds of LBlock-s to obtain the ciphertexts C_i^1 and C_i^2 from which she deduces $C_i^3 = C_i^1 \oplus \delta$ and $C_i^4 = C_i^2 \oplus \delta$ with $\delta = (0x42000004, 0x00060040)$ and asks for their corresponding plaintexts M_i^3 and M_i^4. Finally she checks if the boomerang comes back by testing if $M_i^3 \oplus M_i^4 = \alpha$.

Given our estimate, $m = 2^{56.14}$ quartets are sufficient to expect one boomerang to return (using $2^{58.14}$ ciphering/deciphering operations).

7.8 Conclusion

Starting from an observation similar to the one made by Murphy in 2011, we develop a new theory that explains the behavior of boomerang switches for Feistel ciphers. We introduce the adequate notion of FBCT and give its main properties and relations with other well-known cryptographic tables. Taking things further, we provide a rather simple expression of the probability of a boomerang switch over two rounds, and a (more intricate) general expression of the one over multiple rounds.

8

Looking for new Extended Generalized Feistel Networks structures

You can't see the whole complete act yet. But when this is done... when it's finished...it's gonna be... People will barely be able to comprehend.

John Doe

This final chapter presents a study on a particular family of Extended Generalized Feistel Networks (EGFNs), first introduced in [BMT14]. This work was initiated during the design of Lilliput-AE [ABC$^+$18], a Round 1 candidate the NIST LWC standardization process: starting from round function of the Lilliput lightweight block cipher, its aim was to determine whether it was possible to derive lighter structures providing good resistance to differential and integral cryptanalysis and optimal diffusion delay.

First, some properties of Generalized Feistel Networks (GFNs) and their extensions are recalled in Section 8.1. This section essentially summarizes the results from [BMT14, Tho15], in which the matrix representation of GFNs were introduced and used to propose an extension of such schemes, namely, Extended Generalized Feistel Networks (EGFNs). Then, Section 8.2 provides some comments on the experiments that were conducted to assess the resistance to integral and differential cryptanalysis of some EGFNs derived from the structures proposed in [BMT14]. Some early results seemed to point out that the Lilliput round function was among the best choices. Moreover, this primitive had already been subject to third-party analysis [ST16, ST17]. For these reasons, my co-authors and I ultimately chose the build our NIST proposal around the round function of Lilliput. While this work was carried out in the context of our NIST LWC submission, it covers a broader class of EGFNs and thus, we found it appropriate to place it as a separate chapter. Unfortunately, this work remains incomplete for now as we were not successful in identifying some general criteria for the family of EGFNs considered.

8.1 Background on GFNs and their Extensions

8.1.1 Generalized Feistel Networks

While Feistel networks (see Section 1.3.2) act on a state that is split into two branches (also called *blocks*), Zheng *et al.* introduced at CRYPTO 1989 [ZMI90] a construction that considered a k-block state, called the *Generalized Feistel Network* or GFN. GFNs mostly inherit the interesting features of the classical Feistel network: their structure remains simple and allows for high parallelism, they transform any set of noninvertible functions into a permutation and the inverse is the same up to a permutation of the blocks, which makes them suitable for low cost implementation. One GFN round is made of two layers: the *nonlinear layer* and the *permutation layer*. The nonlinear layer contains one or several Feistel functions F_i applied in parallel whose outputs are then XORed to a single or to several other branches. The permutation layer originally consisted of a cyclic block-wise shift to the left, however works such as the ones of [SM10, YI13, CGT19] or [DFLM19] among others have since shown that the diffusion delay of GFNs could be improved by using different shufflings.

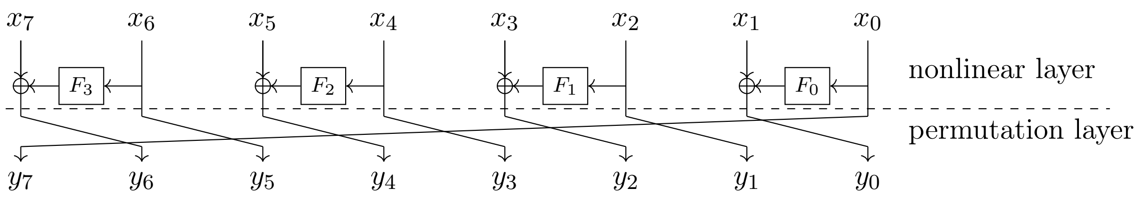


Figure 8.1: An example of a GFN round with $k = 8$ blocks.

Figure 8.1 depicts an example of what is called a *type-2* GFN with $k = 8$ branches. The type of a GFN is defined by the way the Feistel functions interact with the k input blocks during one round of the GFN. A brief description of each of the main types of GFNs is given in the rest of this section. For the sake of simplicity, the permutation layer considered will be the cyclic shift to the left.

Type-1. Akin to the classical Feistel network, the type-1 Feistel scheme only has one Feistel function that takes one branch as input, and XORs its output to another branch, as shown in Figure 8.2.

Type-2. The type-2 Feistel scheme is defined over an even number of branches k and operates using $k/2$ Feistel functions $F_0, \cdots, F_{k/2-1}$. Each function F_i takes as input branch x_{2i} and the output is XORed to branch x_{2i+1}, as shown in Figure 8.3. This construction is used in

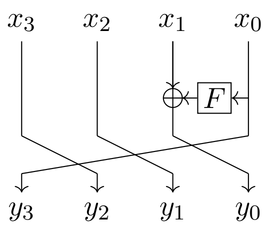


Figure 8.2: One round of a type-1 GFN with $k = 4$ blocks.

Clefia [SSA+07], and in lightweight ciphers such as LBlock [WZ11b], Piccolo [SIH+11] and Twine [SMMK13].

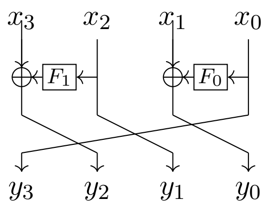


Figure 8.3: One round of a type-2 GFN with $k = 4$ blocks.

Type-3. The nonlinear layer of a type-3 Feistel network with k branches contains $k - 1$ Feistel functions. Each function F_i takes as input branch x_i and the output is XORed to branch x_{i+1}, for $0 \leq i \leq k - 2$. Because of its structure, one cannot have a parallel evaluation of the Feistel functions in decryption mode for the type-3 Feistel network. Indeed, function F_i requires $x_i = y_{i-1} \oplus F_{i-1}(x_{i-1})$, meaning that each Feistel function has to be treated sequentially. One example is given in Figure 8.4.

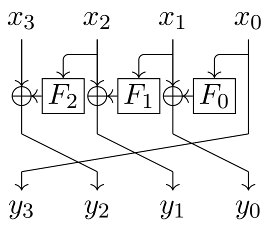


Figure 8.4: One round of a type-3 GFN with $k = 4$ blocks.

Source-heavy. A source-heavy Feistel network with k branches only leverages a single Feistel function defined from $\mathbb{F}_2^{(k-1)n}$ to $\mathbb{F}_2^n$ that takes $k - 1$ branches as inputs and the output is XORed to the remaining branch. Figure 8.5 depicts an example of a source-heavy GFN with 4 blocks. This type of construction is used in the Rc2 block cipher [Riv98], the Sha-1 hashing function [Nat95] or Sms4 [Dt08], which is now a Chinese standard for Wireless LANs.

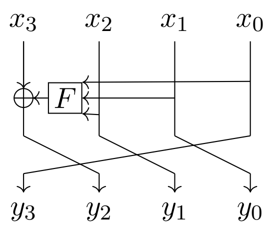


Figure 8.5: One round of a source-heavy GFN with $k = 4$ blocks.

Target-heavy. As depicted in Figure 8.6, a target-heavy Feistel scheme uses a single Feistel function from $\mathbb{F}_2^n$ to $\mathbb{F}_2^{(k-1)n}$. This construction was used in MARS [BCD+99], which was selected as a finalist in the AES competition.

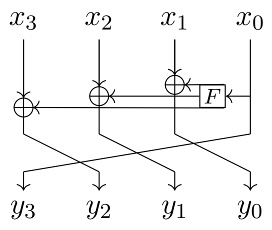


Figure 8.6: One round of a target-heavy GFN with $k = 4$ blocks.

Nyberg. The Nyberg Feistel [Nyb96]—depicted in in Figure 8.7—is similar to the type-2 Feistel in the way that it also operates using $k/2$ Feistel functions. However in this case, the first half of the branches are emitting branches—through a Feistel function—and the second half are receiving branches. More precisely, for $0 \leq i \leq k/2 - 1$, the Feistel function F_i takes as input branch x_i and its output is XORed to branch x_{k-i}.

8.1.2 Full diffusion delay

Once the number of branches of a GFN is chosen, one important matter that remains is the number of rounds required to ensure a secure scheme. One criterion for that choice is the *full diffusion delay* [Tho15], originally introduced as the *maximum diffusion round* in [SM10]. Informally, the full diffusion delay is the smallest number of rounds needed so that each of the final output blocks

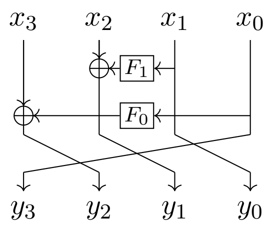


Figure 8.7: One round of a Nyberg GFN with $k = 4$ blocks.

$y_0, \cdots y_{k-1}$ depends on every input blocks $x_0, \cdots, x_{k-1}$. This notion gives a quantifiable measure of the diffusion inside a block cipher. A more formal definition can be given, using a graph point of view.

Definition 8.1 (Associated digraph of a GFN)**.** *The* associated digraph *of a k-block* GFN *is the graph with vertex set $\{0, \cdots, k-1\}$ and such that (i, j) is an edge if the output block y_j depends on the input block x_i. Edges of the graph either come from the nonlinear layer or the permutation layer. Whenever the dependency spans from a Feistel function, a symbol F is used to label the corresponding edge.*

The same F is used for all the different round-functions used throughout the cipher.

A block x_i is said to affect or influence an output block y_j^r at round r if the computation of y_j^r depends on x_i. Block x_i is said to have diffused at round r if all outputs y_j^r at round r depend on x_i, for $0 \leq j \leq k-1$. When all blocks x_i have diffused at round r, for $0 \leq i \leq k-1$, the GFN has reached *full diffusion*. The full diffusion delay, denoted $d+$, is the smallest number of round required to reach full diffusion.

Definition 8.2 (Diffusion delay $d+$ of a GFN)**.** *Input block x_i influences output block y_j^r if there exists a path of length exactly r going from vertex i to vertex j in the associated digraph of the* GFN*. The* full diffusion delay *$d+$ can thus be defined as the smallest integer r such that, for all ordered pair of vertices (i, j) there exists a path of length exactly r connecting i to j.*

Namely, the full diffusion delay $d+$ is the radius of the graph associated to the GFN. From these definitions, it should be noticed that the full diffusion delay of a GFN is a structural property of a GFN: it solely depends on the way the Feistel functions used in the GFN are placed, provided they are not the zero function. Additionally, if a GFN reaches a full diffusion state at round r then it will remain in such state for the remaining rounds.

In a similar manner, the full diffusion delay for decryption can be defined using the digraph associated to the GFN in decryption mode. It is denoted d^-.

Since encryption and decryption are equally important, we consider the full diffusion delay for both ways, $d = \max(d^+, d^-)$. Table 8.1 summarizes the full diffusion delay for both ways d for classical GFNs.

GFN Type	Source-Heavy	Target-Heavy	Type-1	Type-2	Type-3	Nyberg
d	k	k	$(k-1)^2+1$	k	k	k

Table 8.1: Both-way full diffusion delay d for the main instances of GFNs with k blocks.

8.1.3 Improvement of the diffusion delay

As stated before, the permutation layer of a GFN initially consisted in a cyclic shift of the branches. At FSE 2010, Suzaki and Minematsu [SM10] presented their analysis results on GFNs regarding noncyclic permutations. It was shown that the diffusion could be improved for type-2 GFNs by carefully choosing the permutation. It particular, a generic construction based on de Bruijn graphs was proposed for schemes with a number of branches k that is a power of 2, leading to a decrease of the diffusion delay from k to$2 \log_2 k$. Similar constructions were analyzed in [YI13] for type-1, type-3, source-heavy and target-heavy GFNs with noncyclic permutation. Regarding the source-heavy and the target-heavy GFNs, it was observed that these scheme can only reach full diffusion if the permutation is a single cycle of length k, leading to a diffusion in k rounds, otherwise $d = +\infty$. As a result, using different permutations cannot improve the diffusion delay. For Type-1 GFNs on the other hand, authors were able to give an optimum generic construction for any k. Finally, a necessary and sufficient condition for to improve the diffusion delay of type-3 GFNs with k branches, for $2 \leq k \leq 8$.

All the improved results are given in Table 8.2.

GFN Type	[YI13] Type-1	[SM10] Type-2
d	$k(k+2)/2-2$	$2\log_2 k$

Table 8.2: Improved both-way full diffusion delay d for type-1 and type-2 of GFNs with k blocks.

8.1.4 Matrix Representation of Feistel Networks

Another way to further study the diffusion properties of a GFN is through a more unified vision using a matrix representation, introduced by Berger *et al.* in [BMT14, Tho15]. In this section, we give the definition of a GFN matrix and how this representation ties in with the graph view introduced earlier.

Matrix of a GFN

We recall that one round of a GFN can be divided into two distinct transformations: first, the nonlinear layer $\mathcal{N}$ and second, the permutation layer π. The matrix of the permutation layer is straightforward: it is the $k \times k$ binary permutation matrix $\mathcal{P}$ such that

$$\mathcal{P}_{i,j} = \begin{cases} 1 & \text{if } i = \pi(j) \\ 0 & \text{otherwise.} \end{cases}$$

As for the nonlinear layer, the definition is given below:

Definition 8.3 (Matrix representation of the nonlinear layer of a GFN)**.** *The* matrix representation of the nonlinear layer *of a k-block* GFN *is the $k \times k$-matrix $\mathcal{F}$ over $\mathbb{Z}[F]$ with an all-one diagonal and with a formal parameter denoted F at position (i, j) if and only if there is a Feistel-function taking block x_i as input and whose output is* XOR*ed to block x_j.*

It follows that the matrix of the GFN as a whole can be defined as below:

Definition 8.4 (Matrix representation of a GFN)**.** *Consider a* GFN *whose nonlinear layer and permutation layer are represented by a matrix $\mathcal{F}$ and a binary permutation matrix $\mathcal{P}$ respectively, then the matrix of the* GFN *is $\mathcal{M} = \mathcal{P} \times \mathcal{F}$.*

For a GFN with k blocks, $\mathcal{M}$ is the $k \times k$ matrix over $\mathbb{Z}[F]$ such that for $0 \leq i, j \leq k-1$,

$$\mathcal{M}_{i,j} = \begin{cases} 1 & \text{if } y_i \text{ directly depends on } x_j \\ F & \text{if } y_i \text{ depends on } x_j \text{ via a Feistel function} \\ 0 & \text{otherwise.} \end{cases}$$

The matrix of the GFN in decryption mode is $\mathcal{M}^{-1}$. The corresponding matrices for the GFN depicted in Figure 8.1 are given in Figure 8.8.

The following theorem ties the graph representation and the matrix representation of a GFN.

$$\mathcal{M} = \begin{pmatrix} F & 1 & & & & & & \\ & & 1 & & & & & \\ & & F & 1 & & & & \\ & & & & 1 & & & \\ & & & & F & 1 & & \\ & & & & & & 1 & \\ & & & & & & F & 1 \\ 1 & & & & & & & \end{pmatrix} \quad \mathcal{P} = \begin{pmatrix} & 1 & & & & & & \\ & & 1 & & & & & \\ & & & 1 & & & & \\ & & & & 1 & & & \\ & & & & & 1 & & \\ & & & & & & 1 & \\ & & & & & & & 1 \\ 1 & & & & & & & \end{pmatrix} \quad \mathcal{F} = \begin{pmatrix} 1 & & & & & & & \\ F & 1 & & & & & & \\ & & 1 & & & & & \\ & & F & 1 & & & & \\ & & & & 1 & & & \\ & & & & F & 1 & & \\ & & & & & & 1 & \\ & & & & & & F & 1 \end{pmatrix}$$

Figure 8.8: Decomposition of the transition matrix of the GFN given in Figure 8.1.

Theorem 8.5. *The matrix $\mathcal{M}$ of a* GFN *is the transpose of the adjacency matrix of the associated digraph of the* GFN*.*

The graph representation and the matrix representations are thus equivalent. Powers of the matrix $\mathcal{M}$ provide information on how each block diffuses throughout the scheme. Indeed, the element $\mathcal{M}^r_{i,j}$ gives the number of walks of length r from vertex i to vertex j. Put another way, $\mathcal{M}^r_{i,j}$ indicates wether input block x_i influences output block y^r_j after r rounds. As a consequence the diffusion delay d^+ can be defined from the matrix point of view as follows:

Theorem 8.6. *Given a* GFN *with diffusion delay d^+ and associated matrix $\mathcal{M}$, then d^+ is the smallest nonzero integer such that $\mathcal{M}^{d^+}$ has only nonzero coefficients.*

The diffusion delay in decryption mode d^- can naturally be defined in an analogous way using $\mathcal{M}^{-1}$. One advantage of the matrix representation over the graph representation is that one can multiply different matrices together instead of just raising one matrix to a power, which means that one can use round functions that are different. Moreover, the notion of inversion comes more naturally for matrices than it does for graphs.

Matrix Equivalences

The matrix representation now allows us to properly define an equivalence relation on GFNs.

Definition 8.7. *Two* GFNs *are equivalent if their associated graphs are isomorphic. From a matrix point of view, two* GFN *matrices $\mathcal{M}$ and $\mathcal{M}'$ are equivalent if there exists a permutation matrix π such that $\pi\mathcal{M}\pi^{-1} = \mathcal{M}'$.*

In other words, two GFNs are equivalent if they are the same up to block reindexation and thus share the same properties, such as a common full diffusion delay. Moreover, the following property describes how the corresponding inner layers of two equivalent GFNs are tied one to another.

Theorem 8.8. *Let $\mathcal{M} = \mathcal{P}\mathcal{F}$ and $\mathcal{M}' = \mathcal{P}'\mathcal{F}'$ be two* GFNs *according to Definition 8.4 and equivalent as defined in Definition 8.7. Let also π be such that $\pi\mathcal{M}\pi^{-1} = \mathcal{M}'$. Then $\pi\mathcal{P}\pi^{-1} = \mathcal{P}'$ and $\pi\mathcal{F}\pi^{-1} = \mathcal{F}'$.*

Two GFNs are thus equivalent if and only if both of their layers are equivalent with the same conjugating element π.

8.1.5 Characterizing Quasi-involutive GFNs

In this section, we look at the characteristics of a GFN matrix. Clearly, not every matrix with coefficient in $\{0, 1, F\}$ corresponds to a GFN.

We recall that an important and interesting feature of GFNs is that they are invertible permutations, regardless of the invertible nature of the inner Feistel functions that are used in each round: the decryption process uses the exact same functions. Consequently, when looking at the matrix of the GFN in decryption mode $\mathcal{M}^{-1}$, no element should be computed as inverses of expressions containing an F. This yields that $\det(\mathcal{M})$ is independent of F, or equivalently, that $\det(\mathcal{M}) = \pm 1$, hence $\det(\mathcal{F}) = \pm 1$ since $\mathcal{P}$ is a permutation matrix.

Another feature of many GFNs is that the decryption process is roughly the same as the encryption process, up to inverting the permutation layer $\mathcal{P}$. Such property is called *quasi-involutiveness*. The remaining of this section focuses of these particular schemes.

Definition 8.9. *A* GFN *is* quasi-involutive *if its nonlinear layer is an involution.*

All classical GFNs described in Section 8.1.1 are quasi-involutive, except for the type-3 GFN, in which case the round-functions must be evaluated sequentially while decrypting.

The definition of a matrix of a quasi-involutive GFN is given below:

Definition 8.10. *A $k \times k$-matrix $\mathcal{M}$ with coefficients in $\{0, 1, F\} \subset \mathbb{Z}[F]$ is the* matrix of a quasi-involutive GFN *if it can be written as $\mathcal{M} = \mathcal{P}\mathcal{F}$, where $\mathcal{P}$ is a permutation matrix and $\mathcal{F}$ satisfies the following conditions:*

1. *the main diagonal is filled with 1,*
2. *the off-diagonal coefficients are either 0 or F,*
3. *for each index i, row i and column i cannot both have an F coefficient.*

Condition 3 means that the blocks of the GFN can be partitioned into three categories:

- *emitting* blocks, that go through a Feistel function,
- *receiving* blocks, that are XORed to the output of a Feistel function,
- blocks that do not emit nor receive.

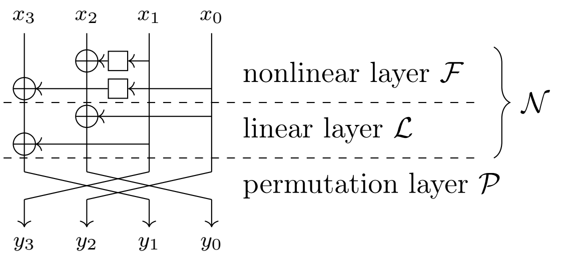
Theorem 8.11. *Let $\mathcal{M} = \mathcal{P}\mathcal{F}$ be a* GFN *according to Definition 8.10. Then $\mathcal{F}$ is invertible and $\mathcal{F}^{-1} = 2\mathcal{I} - \mathcal{F}$, where $\mathcal{I}$ stands for the identity matrix.*

Theorem 8.11 shows the property of quasi-involutiveness of GFNs defined as in Definition 8.10. The matrices are indeed with coefficients in a ring of characteristic 0 so that distances can be computed/measured in the associated Feistel graph. In this context, inverting the operation of adding the output of the Feistel functions means performing the corresponding subtraction, hence the term $-\mathcal{F}$. The other term, $2\mathcal{I}$, ensures that the coefficients on the diagonal equal 1. In characteristic 2, which is the case where the outputs of the Feistel functions are simply XORed with other blocks, this theorem yields $\mathcal{F}^{-1} = \mathcal{F}$.

Conversely, a characterization can be given for matrices $\mathcal{F}$ such that $\mathcal{F}^{-1} = 2\mathcal{I} - \mathcal{F}$:

Theorem 8.12. *Let $\mathcal{F}$ be a matrix that verifies Conditions 1 and 2 of Definition 8.10. If $(\mathcal{F} - \mathcal{I})^2 = 0$ then $\mathcal{F}$ also verifies Condition 3.*

In other words, all quasi-involutive nonlinear layer matrices are exactly those for which Condition 3 of Definition 8.10 holds.


x_3
x_2
x_1
x_0
nonlinear layer $\mathcal{F}$
linear layer $\mathcal{L}$
permutation layer $\mathcal{P}$
$\mathcal{N}$
y_3
y_2
y_1
y_0

$$\mathcal{M} = \begin{pmatrix} I & F & 1 & \\ F & I & & 1 \\ 1 & & & \\ & 1 & & \end{pmatrix} \quad \mathcal{P} = \begin{pmatrix} & & 1 & \\ & & & 1 \\ 1 & & & \\ & 1 & & \end{pmatrix}$$

$$\mathcal{L} = \begin{pmatrix} 1 & & & \\ & 1 & & \\ I & & 1 & \\ & I & & 1 \end{pmatrix} \quad \mathcal{F} = \begin{pmatrix} 1 & & & \\ & 1 & & \\ & F & 1 & \\ F & & & 1 \end{pmatrix}$$

Figure 8.9: An example of an EGFN with its three layers and the corresponding matrices.

8.1.6 Extended Generalized Feistel Networks

In this section, we remind the results from [BMT14, Tho15] in which *Extended Generalized Feistel Networks* (EGFNs) were introduced. Using permutations other than the cyclic shift in the permutation layer of GFNs has proven to an effective way to improve diffusion, as shown by Suzaki *et al.* in [SM10] and more recently in [CGT19] and [DFLM19]. The idea behind EGFNs is to go even further and to replace the block permutation with a linear mapping, which is equivalent to adding a third layer, called the *linear layer* between the nonlinear layer and the permutation layer.

Definition 8.13. *A matrix $\mathcal{M}$ with coefficients in $\{0, 1, F, I\} \subset \mathbb{Z}[F, I]$ is an Extended Generalized Feistel Network (*EGFN*) matrix if it can be written in the form $\mathcal{M} = \mathcal{P}\mathcal{N}$ such that $\mathcal{P}$ is a permutation matrix and the matrix $\mathcal{N}$ satisfies the following properties:*

1. *the main diagonal is filled with* 1*,*
2. *the off-diagonal coefficients are either* 0*, F or I,*
3. *for each index i, row i and column i cannot both contain a nonzero coefficient other than on the diagonal,*
4. *for each index i, if row i contains an I then it also contains an F.*

An example of an EGFN with $k = 4$ branches and its associated matrices is depicted in Figure 8.9.

Definition 8.14. *Conside an* EGFN *and its associated matrix $\mathcal{M}$ as defined in Definition 8.13, the diffusion delay in encryption mode of the* EGFN *is the smallest integer d^+ such that $\mathcal{M}^{d^+}$ does not have any zero coefficients. The diffusion delay in decryption mode is defined in a similar way and is denoted d^-. Finally, we define the full diffusion delay of the* EGFN *as $d = max(d^+, d^-)$.*

More generally, we have the following definition.

Definition 8.15. *Consider an* EGFN *and its associated matrix $\mathcal{M}$ as defined in Definition 8.13, for an integer $\ell \in \mathbb{N}$, we define the ℓ^{th} diffusion delay of the scheme as the smallest integer d_ℓ^+ such that:*

$$\min_{0 \le i,j \le k-1} deg(\mathcal{M}'^{d_\ell^+}_{i,j}) \ge \ell,$$

where $\mathcal{M}'$ is the matrix with coefficients in $\{0, 1, F\} \subset \mathbb{Z}[F]$ such that:

$$\mathcal{M}'_{i,j} = \begin{cases} 1 & \text{if } \mathcal{M}_{i,j} = I \\ \mathcal{M}_{i,j} & \text{otherwise.} \end{cases}$$

Similarly, we define d_ℓ^- using $\mathcal{M}^{-1}$ instead of $\mathcal{M}$.
$d_\ell = max(d_\ell^+, d_\ell^-)$ is then called the ℓ^{th} diffusion delay in both ways.

Theorem 8.16. *Consider an* EGFN *as defined in Definition 8.13 with $k \geq 4$ blocks, k even, and its associated matrices $\mathcal{M}$, $\mathcal{N}$ and $\mathcal{P}$ with $\mathcal{N}$ being the matrix representing the both nonlinear and linear layers of the* EGFN *and with $\mathcal{P}$ being the matrix of the permutation layer defined by a permutation π then,*

1. *if the matrix $\mathcal{N}$ is equal to $\mathcal{N} = \left(\begin{smallmatrix} \mathcal{I} & 0 \\ \mathcal{A} & \mathcal{I} \end{smallmatrix}\right) \in \mathbb{Z}[F, I]$ where $\mathcal{I}$ is the $\frac{k}{2} \times \frac{k}{2}$ identity matrix and where $\mathcal{A}$, the lower left quarter of $\mathcal{N}$, is such that* $\mathcal{A} = \begin{pmatrix} & (0) & & F & \substack{F \\ I} \\ & & \iddots & & I \\ & F & & (0) & \vdots \\ F & I & I & \cdots & I \end{pmatrix}$
2. *and if $\mathcal{P}$ globally exchanges emitters (blocks x_0 to $x_{k/2-1}$) with receivers (blocks $x_{k/2}$ to x_{k-1}) with $\pi(k/2 - 1) = k - 1$ and $\pi(k - 1) = k/2 - 1$,*

then its diffusion delay d is equal to 4.

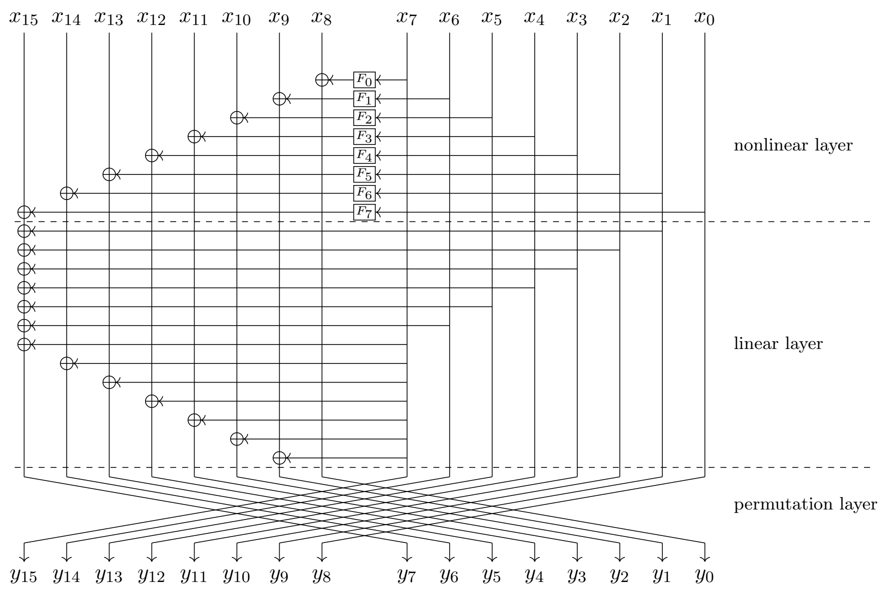

x_{15} x_{14} x_{13} x_{12} x_{11} x_{10} x_9 x_8 x_7 x_6 x_5 x_4 x_3 x_2 x_1 x_0
F_0 F_1 F_2 F_3 F_4 F_5 F_6 F_7
nonlinear layer
linear layer
permutation layer
y_{15} y_{14} y_{13} y_{12} y_{11} y_{10} y_9 y_8 y_7 y_6 y_5 y_4 y_3 y_2 y_1 y_0

Figure 8.10: One round of an EGFN that reaches full diffusion in 4 rounds.

Figure 8.10 depicts one example of an EGFN round that fulfills all the conditions stated in Theorem 8.16 and reaches full diffusion in 4 rounds as a result.

8.1.7 Lilliput

The LILLIPUT round function is a particular case of EGFN deduced from Theorem 8.16, introduced in [BFMT16]. This EGFN with $k = 16$ blocks is depicted in Figure 8.11.

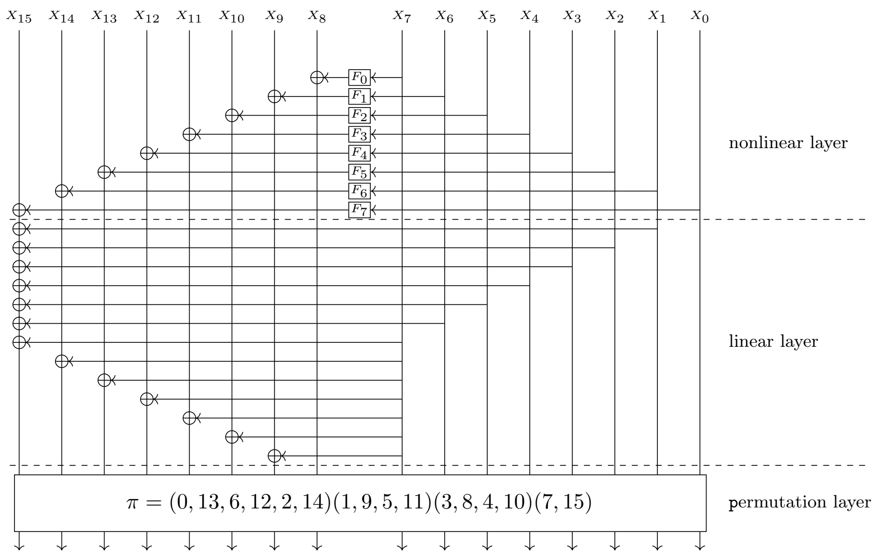

x_{15} x_{14} x_{13} x_{12} x_{11} x_{10} x_9 x_8 x_7 x_6 x_5 x_4 x_3 x_2 x_1 x_0
F_0 F_1 F_2 F_3 F_4 F_5 F_6 F_7
nonlinear layer
linear layer
$\pi = (0, 13, 6, 12, 2, 14)(1, 9, 5, 11)(3, 8, 4, 10)(7, 15)$
permutation layer

Figure 8.11: The round function of LILLIPUT, that reaches full diffusion in 4 rounds. The permutation π is given as a product of cycles.

The permutation π has been chosen so that the number of active S-boxes on 18, 19 and 20 rounds is maximized among the 37108 possible permutations (up to block reindexing equivalence) that fulfill the conditions of Theorem 8.16. A more complete description of LILLIPUT can be found in Section 2.4.4 of this thesis.

8.2 Towards Lighter EGFN Constructions

The goal of this work was to design an even lighter version of LILLIPUT with similar properties—same diffusion delay and good resistance to differential and integral cryptanalysis.

For the permutation layer, we wanted to focus on involutions to minimize the overhead of decryption. For the upper layers, our intentions was to reduce the number of XORs, while keeping the same amount nonlinear components. From a matrix perspective, this essentially meant removing some of coefficients equal to I.

A relaxed definition of EGFNs. Regarding Theorem 8.16, one interesting observation is that the result stays true, as long as the last row and the last column of the $\mathcal{A}$ matrix—the lower left quarter of the $\mathcal{N}$ matrix—are both entirely filled with nonzero coefficients. In other words, all the coefficients of the antidiagonal except the two ends can be removed, potentially allowing for lighter constructions of EGFNs with a diffusion delay d equal to 4. However, the four conditions stated in Definition 8.13 were too restrictive and did not allow for much reduction. Conditions 1 and 2 were essential to the definition of EGFNs and Condition 3 simply implied that branches of the EGFN were either *emitters*[1] or *receivers*[2], but not both. Condition 4, on the other hand,

[1]We recall that an emitting branch influences other branch in a linear or nonlinear way.
[2]A receiving branch is influenced by at least another branch in a linear or nonlinear way.

seemed too strong to us and thus, we decided to investigate whether finding EGFNs with good properties that did not satisfy this condition was possible.

As a result, in the following, we consider $\mathcal{M}$ matrices with coefficients in $\{0, 1, F, I\} \subset \mathbb{Z}[F, I]$ that can be written in the form $\mathcal{M} = \mathcal{P}\mathcal{N}$ such that $\mathcal{P}$ is a involutory permutation matrix and the matrix $\mathcal{N}$ satisfies the following properties:

1. the main diagonal is filled with 1,
2. the off-diagonal coefficients are either 0, F or I,
3. for each index i, row i and column i cannot both contain a nonzero coefficient other than on the diagonal.

In the case of the 8×8 $\mathcal{A}$ matrix used in LILLIPUT, shown in Figure 8.12, the antidiagonal is filled with Fs and the rest of the last column and of the last row is made of 13 coefficients all equal to I. However, with this relaxed definition, other configurations with the same number of Feistel functions and the same diffusion delay are possible: instead of being on the antidiagonal, the F coefficients can be moved to the last row or to the last column, thus removing some I coefficients, which is equivalent to removing some XORs from the scheme.

$$\begin{pmatrix} & (0) & & F & \begin{smallmatrix}F\\ I\end{smallmatrix} \\ & & \iddots & & I \\ & \begin{smallmatrix}F\\ I\end{smallmatrix} & & (0) & \vdots \\ F & & I & \cdots & I \end{pmatrix}$$

Figure 8.12: The $\mathcal{A}$ matrix of LILLIPUT.

We therefore examined several sets of 8×8 matrices made of 8 coefficients equal to F following a similar triangular structure as the one described in Theorem 8.16 but with a reduced number of nonzero coefficients on the antidiagonal. For each matrices, we wanted to assess the number of permutations in P_{invol} that resulted in schemes with good resistance to integral and differential cryptanalysis. The idea was then to keep the matrices that could be combined with the highest number of permutations for more in-depth analysis. The results of our experiment are reported in the following parts. We also try to understand how the permutations and the matrices interact together to possibly exhibit some criteria that could help with the choice of the permutation layer.

8.2.1 Resistance to integral attacks

Integral cryptanalysis [DKR97] (see Section 3.3.1) uses sets of chosen plaintexts with a fixed part and another that varies through all possibilities. Typically, for an n-bit block cipher, these sets contain 2^n messages that only differ in one block. The XOR of these 2^n values necessarily sums up to zero and the sum of the corresponding ciphertexts can leak information about the cipher's operation.

The examined matrices were split into three sets (with some redundancy), described below.

1. The first set consists of matrices $\mathcal{A}$ such that each *column* of $\mathcal{A}$ has *exactly one* F coefficient. We recall that this coefficient can either be on the antidiagonal, on the last row or on the last column of $\mathcal{A}$.

2. In the second set, we consider matrices $\mathcal{A}$ such that each *row* has *exactly one* F coefficient. As for the first set, this coefficient can either be on the antidiagonal, on the last row or on the last column. This entire set actually verifies Condition 4 of Definition 8.13.

3. Finally, the last set consists of matrices with *no coefficients on the antidiagonal* at all, and 8 F coefficients.

For each matrix, we also ensure that the last row and the last column of $\mathcal{A}$ contain only nonzero coefficients by filling the gaps with I coefficients in order to guarantee a diffusion delay equal to 4. This entails that in any case, branch x_7 influences every receiving branch x_i, for $8 \leq i \leq 15$—linearly or via a Feistel function—and x_{15} receives from every emitting branch x_j, for $0 \leq j \leq 7$.

For the rest of this section, we only focus on the set of $7! = 5040$ involutory permutations π_i that satisfy Condition 2 of Theorem 8.16. We denote the set by $\mathcal{P}_{invol}$.

Set 1: every emitting branch is taken as input of a Feistel function

The set considered here contains matrices for which *each column has exactly one F coefficient.* This coefficient can either be on the antidiagonal, on the last row or on the last column of $\mathcal{A}$.

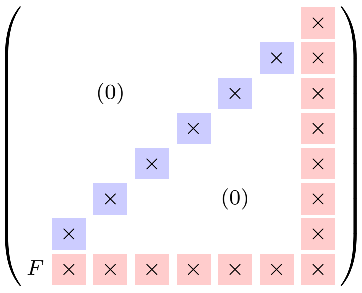


Figure 8.13: Layout of a matrix from set 1. Each column can only contain one F coefficient. Red coefficients are either I or F, blue coefficients equal F if the red coefficient on the same column equals I and 0 otherwise.

In other words, for $0 \leq i \leq 7$, branch x_i emits through exactly one Feistel function. Moreover, if $0 < i < 7$, there are two possibilities: the output of the Feistel function can either be XORed to branch x_{15-i} (if the F is on the antidiagonal) or to branch x_{15} (if the F is on the last row). For x_7, the output of the Feistel function can be XORed to any branch x_j, with $8 \leq j \leq 15$ and x_{15} will always be influenced at least by x_0 in a nonlinear manner. Figure 8.13 shows the general layout of the matrix considered; an example and the resulting scheme are depicted in Figure 8.14.

The set contains $8 \times 2^6 = 512$ matrices. Indeed, the F on the first column of $\mathcal{A}$ is fixed and for each of the 8 possible positions for the F on the last column, the remaining 6 F coefficients are either on the antidiagonal or on the last row.

All matrices result in the same d_1 and d_2 values (as defined by Definition 8.15) as LILLIPUT, which are 5 and 6 respectively, for all permutations in P_{invol}.

When considering 8 rounds, we found 20 matrices with no integral property, no matter which permutation was used. These are the 14 matrices with at least 7 F coefficients on the antidiagonal, and the 6 matrices that have 6 Fs on the diagonal with the two remaing Fs at $(7, i)$ and $(i, 7)$ for $i \in [\![1, 6]\!]$.

From the EGFN point of view, this means that the three cases described below can occur.

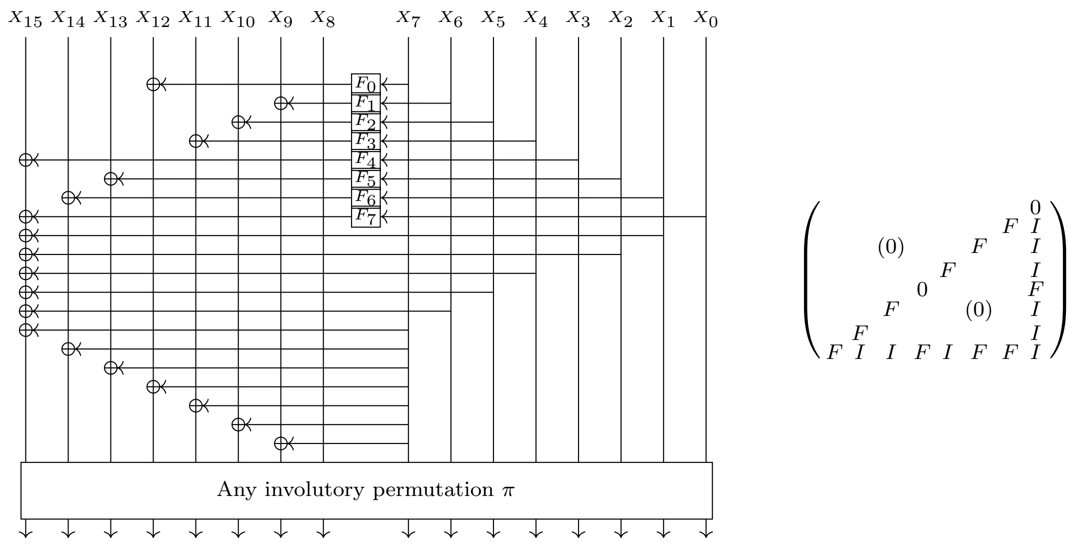

x_{15} x_{14} x_{13} x_{12} x_{11} x_{10} x_9 x_8 x_7 x_6 x_5 x_4 x_3 x_2 x_1 x_0
F_0 F_1 F_2 F_3 F_4 F_5 F_6 F_7
Any involutory permutation π
(0)
(0)
0
F
I

Figure 8.14: A matrix from set 1 with no integral property for 8 rounds (3rd case) and the associated EGFN with 16 blocks.

1. Branch x_i emits via a Feistel function to branch x_{15-i}, for $i \in [\![0, 7]\!]$. That is the LILLIPUT matrix.

2. There is a branch x_e, with $1 \leq e \leq 7$, that emits via a Feistel function to x_{15}, and for all $i \in [\![0, 7]\!] \setminus \{e\}$, branch x_i emits via a Feistel function to branch x_{15-i}.

3. There is a branch x_e, with $1 \leq e \leq 6$, that emits via a Feistel function to x_{15}, and for all $i \in [\![0, 6]\!] \setminus \{e\}$, branch x_i emits via a Feistel function to branch x_{15-i}. As for x_7, the branch goes through a Feistel function and the result is XORed to x_{15-e}. One example is provided in Figure 8.14.

In essence, whenever one considers schemes for which every emitting branch goes through a Feistel function, there exist schemes with one less XOR operation than the original LILLIPUT that ensure an absence of integral property after 8 rounds, no matter which permutation is used. For 6 rounds however, only the LILLIPUT matrix seems to be resistant to integral cryptanalysis for all permutations[a].

[a]Combining the LILLIPUT matrix with involutory permutations, leads to schemes that are all equivalent, thus, the outcome is the same for all involutory permutations in $\mathcal{P}_{invol}$.

Set 2: every receiving branch is XORed to the output of a Feistel function

For this set, each row of $\mathcal{A}$ has exactly one F coefficient. We consider two cases:

1. a case where both ends of the antidiagonal have F coefficients, which is simpler to study, leaving only 6 other coefficients to position, *i.e.* for $8 \leq i \leq 15$, every receiving branch x_i is influenced via a Feistel function, either by x_{15-i} or by x_7.

2. the full case, where the F coefficients can either be anywhere on the last row, the last column or the antidiagonal as long as there are *exactly one F coefficient per row*, *i.e.* for $8 \leq i \leq 14$, every receiving branch x_i is influenced via a Feistel function, either by x_{15-i} or by x_7. As for x_{15}, the branch is influenced in a nonlinear way by one of the emitters. This means that besides x_7, another the emitter might influence two branches at the same time.

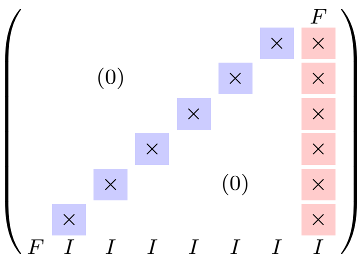


Figure 8.15: Possible placements for F coefficients for a matrix from set 2 in the simple case. Each row can only contain one F coefficient. Red coefficients are either I or F, blue coefficients equal F if the red coefficient on the same row equals I, 0 otherwise.

Simple case. The set consists of 64 matrices: the first row and the last row are already fixed, as shown in Figure 8.15, and there are 2 possibilities for each of the 6 remaining rows (the F coefficient is either on the antidiagonal or on the last column).

All 64 matrices result in the same d_1 and d_2 values as LILLIPUT, for all permutations in $\mathcal{P}_{invol}$. No matrix appears to show any integral property after 8 rounds, for all 5040 permutations. After 6 rounds, however, for some matrices, only a portion of the permutations yields no integral property. We will call these *valid permutations* for the rest of the section.

Proposition 8.17. *The number p_{n_F} of valid permutations depends on the number n_F of F coefficients on the diagonal, excluding the ones on both ends:*

$$\forall n_F \in [\![0,5]\!], p_{n_F} = \begin{cases} (7-n_F)! \cdot n_F! & \textit{if } n_F \textit{ is even} \\ (7-n_F)! \cdot (n_F+1)! & \textit{otherwise.} \end{cases}$$

To explain this proposition, we denote the n_F corresponding emitting branches by $\mathcal{E} = \{e_0, \cdots, e_{n_F-1}\} \subset \{x_1, \cdots, x_6\}$ and the n_F corresponding receiving branches by $\mathcal{R} = \{r_0, \cdots, r_{n_F-1}\} \subset \{x_8, \cdots, x_{14}\}$. Figure 8.16 shows a matrix $\mathcal{A}$ with $n_F = 1$ F coefficient and the corresponding EGFN scheme. Here, $\mathcal{E} = \{x_4\}$ and $\mathcal{R} = \{x_{15-4}\} = \{x_{11}\}$.

1. When n_F is even, then every permutation π such that $\pi(\mathcal{E}) = \mathcal{R}$ leads to a scheme that can resist integral attacks in 6 rounds. There are $n_F!$ such possible swaps and $(7-n_F)!$ possible swaps for the remaining emitting branches (that is, $\{x_0, \cdots, x_6\} \setminus \mathcal{E}$), hence in total:

$$(7-n_F)! \cdot n_F!$$

2. When n_F is odd, the valid permutations include permutations π such that globally exchange $\mathcal{E}$ and $\mathcal{R}$ like in the even case, as well as permutations satisfying the three following conditions:

   (a) x_0 is swapped with a branch $r_{x_0} \in \mathcal{R}$ ($\binom{n_F}{1}$ choices),

(b) $n_F - 1$ emitters among $\mathcal{E}$ are swapped with the $n_F - 1$ receivers left available from $\mathcal{R} \setminus \{r_{x_0}\}$ ($\binom{n_F}{n_F-1}$ choices to pick the emitters, then $(n_F - 1)!$ permutations),

(c) the $(6 - (n_F - 1))$ remaining emitters are swapped with any other receiver that is not in $\mathcal{R}$.

Put another way, when n_F is odd, one of the branches in $\mathcal{E}$ can switch its role with x_0, hence $(7 - n_F)! \cdot n_F!$ possibilities for each of the n_F branches. The reason why this occurs in the odd case remains to be determined. As a result, the total number of permutations is:

$$(7 - n_F)! \cdot n_F! + n_F \cdot (7 - n_F)! \cdot n_F! = (7 - n_F)! \cdot (n_F + 1)!$$

The case where $n_F = 6$ is more particular as $p_6 = 5040$; that is to say, any permutation of $\mathcal{P}_{invol}$ can be used. This case correspond to the LILLIPUT matrix and is simply explained by the fact that for all permutations in $\mathcal{P}_{invol}$, the resulting schemes are equivalent, as mentioned earlier. Actually, when $n_F = 0$, every receiving branch is influenced in a nonlinear way, either by x_0 for x_{15}, or by x_7 for all the other branches. That is, every branch x_i is interchangeable, for $i \in [\![8, 14]\!]$ and, as for $n_F = 6$, any permutation in $\mathcal{P}_{invol}$ produces an equivalent scheme. All the results are summarized in Table 8.3.

n_F	0	1	2	3	4	5	6
p_{n_F}	5040	1440	240	576	144	1440	5040

Table 8.3: Number of permutations p_{n_F} for which no integral property exists for 6 rounds given a matrix $\mathcal{A}$ with n_F F coefficients on the antidiagonal, both ends excluded.

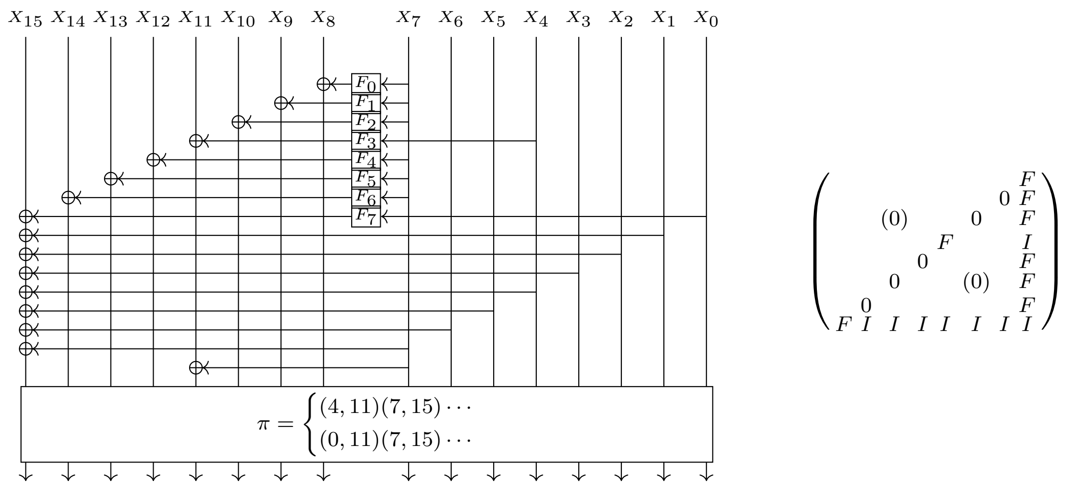

x_{15} x_{14} x_{13} x_{12} x_{11} x_{10} x_9 x_8 x_7 x_6 x_5 x_4 x_3 x_2 x_1 x_0
F_0 F_1 F_2 F_3 F_4 F_5 F_6 F_7
$\pi = \begin{cases}(4,11)(7,15)\cdots \\ (0,11)(7,15)\cdots\end{cases}$
F
0 F
(0) 0 F
F I
0 F
0 (0) F
0 F
F I I I I I I I

Figure 8.16: One example of a matrix for the $n_F = 1$ case of set 2 and its associated EGFN. The permutations are given as a product of cycles.

For the simple case of this set, in which every receiving is influenced via a nonlinear function, there exist schemes with a total of 15 XOR operations for which there is no integral property after 6 rounds. This is 6 less than for the LILLIPUT matrix. The corresponding matrix is the following:

$$\mathcal{A} = \begin{pmatrix} & & & & & & & F \\ & & & & & & 0 & F \\ & & (0) & & & 0 & & F \\ & & & & 0 & & & F \\ & & & 0 & & & & F \\ & & 0 & & & (0) & & F \\ & 0 & & & & & & F \\ F & I & I & I & I & I & I & I \end{pmatrix}$$

All permutations are valid.

Full case. In the full case, we have 512 matrices (64 possible combinations for each position on the last row) and all matrices result in $d_1 = 5$ and $d_2 = 6$, for all permutations. The placement choices for the F coefficients are depicted in Figure 8.17.

After 20 rounds, none of the 512 matrices has any integral property, for all permutations in $\mathcal{P}_{invol}$. This holds down to 8 rounds. After 6 rounds however, 132 matrices have undesirable properties, no matter which permutation is used and for the 380 matrices that remain, not all permutations are valid.

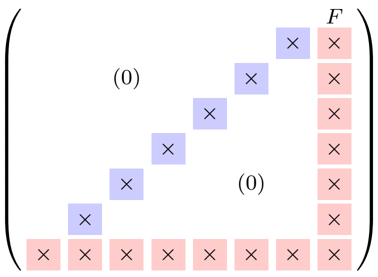


Figure 8.17: Layout of a matrix from set 2, full case. Each row can only contain one F coefficient. Blue coefficients are either 0 or F. Red coefficients are either I or F.

While in the simpler case, the number of F coefficients on the antidiagonal (with both ends excluded) seemed enough to determine the number of valid permutations, other parameters come into play here, such as the number of F coefficients on a single column that is not the last one. Indeed, in the simpler case, the F coefficient on the last row of $\mathcal{A}$ was fixed at position (0,7). Here, it can be anywhere on the last row, at a position that we denote by $(0, lr)$. If another F coefficient is placed on the diagonal at position (lr, lr), then branch x_{lr} has a nonlinear influence on two receiving branches. This leads to many subcases, as shown in Table 8.4, and as a result, the formula derived in the simpler case does not apply anymore.

n_F		0	1	2	3	4	5	6
p_{n_F}	$x_{lr} \notin \mathcal{E}$ and $x_{lr} \neq x_7$	5040	1440	240	576	144	1440	5040
	$x_{lr} \in \mathcal{E}$	-	720	0	528	0	624	720
	$x_{lr} = x_7$	5040	5040	1200	0	0	0	0

Table 8.4: Number of permutations p_{n_F} for which no integral property exists for 6 rounds given a matrix $\mathcal{A}$. n_F is the number of F coefficients on the antidiagonal, both ends excluded, $\mathcal{E}$ is the set of n_F corresponding emitters and x_{lr} denotes the branch that emits to x_{15}. The "-" symbol is used for settings that never occur.

Let us have a closer look at Table 8.4. Consider a matrix with n_F coefficients on the diagonal, excluding both ends. As done in the simple case, let us denote the n_F corresponding emitting branches by $\mathcal{E} = \{x_{e_0}, \cdots, x_{e_{n_F-1}}\} \subset \{x_1, \cdots, x_6\}$, and the n_F corresponding receiving branches by $\mathcal{R} = \{x_{15-e_0}, \cdots, x_{15-e_{n_F-1}}\} \subset \{x_8, \cdots, x_{14}\}$.

- The first row shows that when the only column of $\mathcal{A}$ with more than one F coefficient is the last column ($x_{lr} \notin \mathcal{E}$), and that the F coefficient of the last row is not on the last column either ($x_{lr} \neq x_7$), then the behavior observed in the simple case can be seen again. The only difference here is that x_{lr} might not be x_0.

- On the second row, when one emitter—that is not x_7—nonlinearly influences two other branches ($x_{lr} \in \mathcal{E}$, and for simplicity, let us consider that $x_{lr} = x_{e_0}$), a valid permutation π has to fulfill one the following conditions:

  – if n_F is odd,

    1. $\pi(x_{e_0}) \in \mathcal{R} \setminus \{x_{15-e_0}\}$ and there is one subset $\mathcal{E}_{n_F-2}$ of $\mathcal{E} \setminus \{x_{e_0}\}$ of size $n_F - 2$, such that $\pi(\mathcal{E}_{n_F-2}) = \mathcal{R} \setminus \{x_{15-e_0}, \pi(x_{e_0})\}$ and $\pi(\mathcal{E} \setminus \{x_{e_0}, \mathcal{E}_{n_F-2}\}) \neq x_{15-e_0}$.

    2. $\pi(\mathcal{E}) = \mathcal{R}$

    Thus, the total number of permutations is:

$$(n_F - 1) \cdot \binom{n_F - 1}{n_F - 2} \cdot (n_F - 2)! \cdot \big(6 - (n_F - 1)\big) \cdot (7 - n_F)! \;+\; n_F! \cdot (7 - n_F)!$$

    that is,

$$(7 - n_F)! \cdot \big((n_F - 1)^2 \cdot (n_F - 2)! \cdot (7 - n_F) \;+\; n_F!\big).$$

  – if n_F is even, it is still unclear why for $n_F = 2$ or $n_F = 4$, none of the permutations are valid. When $n_F = 6$, the only valid permutations are the ones such that $\pi(\mathcal{E}) = \mathcal{R}$.

- Regarding the third row, when $n_F = 2$, there are two emitting branches x_{e_0} and x_{e_1}, with $1 \leq e_0, e_1 \leq 6$ that respectively emit to x_{15-e_0} and x_{15-e_1} and x_7 also emits to x_{15}. A valid permutation π has to fulfill one of the following conditions:

  1. $\pi(x_{e_0}) = x_{15-e_1}$ and $\pi(x_{e_1}) \neq x_{15-e_0}$
  2. $\pi(x_{e_1}) = x_{15-e_0}$ and $\pi(x_{e_0}) \neq x_{15-e_1}$

  Since for the other values, $p_{n_F} = 5040$ or 0, we could not

Once again, for the full case of this set, there are 8 schemes with a total of 15 XOR operations for which there is no integral property after 6 rounds, no matter which permutation in $\mathcal{P}_{invol}$ is used. These are the schemes in which x_7 goes through a Feistel function and is XORed to $x_8, \ldots, x_{14}$.

Set 3: no coefficients on the antidiagonal

For this last configuration, we considered matrices with no coefficients on the antidiagonal at all, that is, matrices with only 15 nonzero coefficients. The 8 F coefficients are thus either on the last row or on the last column of matrix $\mathcal{A}$. Consequently, branch x_7 is the only branch that can have more than one receiver (via a Feistel function), and branch x_{15} is the only branch than can have more than one nonlinear emitter.

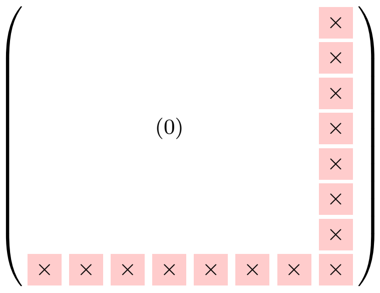


Figure 8.18: Layout of a matrix from set 3. Red coefficients are either equal to I or F.

Out of the $\binom{15}{8}$ combinations, 3446 $\mathcal{N}$ matrices resulted in the same d_1 and d_2 values as LILLIPUT.

If we denote by $a_{i,j}$ the coefficients at row i and column j of $\mathcal{A}$, then these remaining matrices are:

- the $\binom{12}{5}$ possible matrices such that $a_{\frac{k}{2}-1,\frac{k}{2}-1} = F$,
- the 7 matrices with exactly one F coefficient per row such that $a_{7,7} = I$,
- the 7 matrices with exactly one F coefficient per column such that $a_{7,7} = I$.

That is, if x_7 does not emit to branch x_{15}, then either every other emitting branch x_i influences x_{15} in a nonlinear way, for $0 \leq i \leq 6$, or every other receiving branch x_j receives from x_7 in a nonlinear way, for $8 \leq j \leq 14$. We then studied the integral property of these matrices when combined with the permutations from $\mathcal{P}_{invol}$:

- After 20 rounds, 204 matrices showed no integral property, for all permutations. These are the matrices with 6 F coefficients or more on the last column.
- After 8 rounds, only 57 matrices showed no integral property. This result does not depend on the chosen permutation. These are the matrices with exactly 7 F coefficients on the last column (8 possibilities to pick the row without an F on the last column $\times$ 7 possibilities for the F on the last row) and the target-heavy scheme in which x_7 emits to all 8 receiving branches.
- After 6 rounds, only 8 matrices had no integral property. These are the matrices already exhibited with the previous set, in which the first 7 positions of the last column are filled with F coefficients and the remaining F is on the last row.

Conclusion of these expriments

Regarding the interactions between the permutations and the matrices, in some cases we were able to identify the conditions required for a valid permutation. However, these observations still need to be interpreted before any meaningful criteria can be exhibited. From these results it appears that the most promising schemes are the 8 possible constructions in which the first 7 positions of the last column are filled with F coefficients and the remaining F is on the last row, as they display no integral property for 6 rounds, for all permutations in $\mathcal{P}_{invol}$, and they have the minimum number of nonzero coefficients to ensure a diffusion delay equal to 4. The corresponding matrices are depicted in Figure 8.19. It should be noted that these matrices fulfill Condition 4 of Definition 8.13, which we initially deemed too restrictive.

$$\begin{pmatrix} & & & & & & & F \\ & & & & & & & F \\ & & & & & & & F \\ & & & (0) & & & & F \\ & & & & & & & F \\ & & & & & & & F \\ & & & & & & & F \\ \times & \times & \times & \times & \times & \times & \times & \times \end{pmatrix}$$

Figure 8.19: An interesting family of matrices $\mathcal{A}$ with 8 F coefficients. The last coefficient can be placed at any red-marked position.

Theorem 8.18. *Consider an* EGFN *as defined in Def. 8.13 with $k \geq 4$ blocks, k even, and its associated matrices $\mathcal{M}$, $\mathcal{N}$ and $\mathcal{P}$ with $\mathcal{N}$ the matrix representing the nonlinear and linear layers of the* EGFN *and with $\mathcal{P}$ the matrix of the permutation layer defined by a permutation π,*

1. *if the matrix $\mathcal{N}$ is equal to $\mathcal{N} = \left(\begin{smallmatrix} \mathcal{I} & 0 \\ \mathcal{A} & \mathcal{I} \end{smallmatrix}\right) \in \mathbb{Z}[F, I]$ where $\mathcal{I}$ is the $\frac{k}{2} \times \frac{k}{2}$ identity matrix and where $\mathcal{A}$, the lower left quarter of $\mathcal{N}$, is such that $\mathcal{A} = \begin{pmatrix} & & & F \\ & (0) & & \vdots \\ I & \cdots & I & F \end{pmatrix}$;*
2. *and if $\mathcal{P}$ globally exchanges emitters (blocks x_0 to $x_{k/2-1}$) with receivers (blocks $x_{k/2}$ to x_{k-1}) with $\pi(k/2-1) = k-1$ and $\pi(k-1) = k/2-1$;*

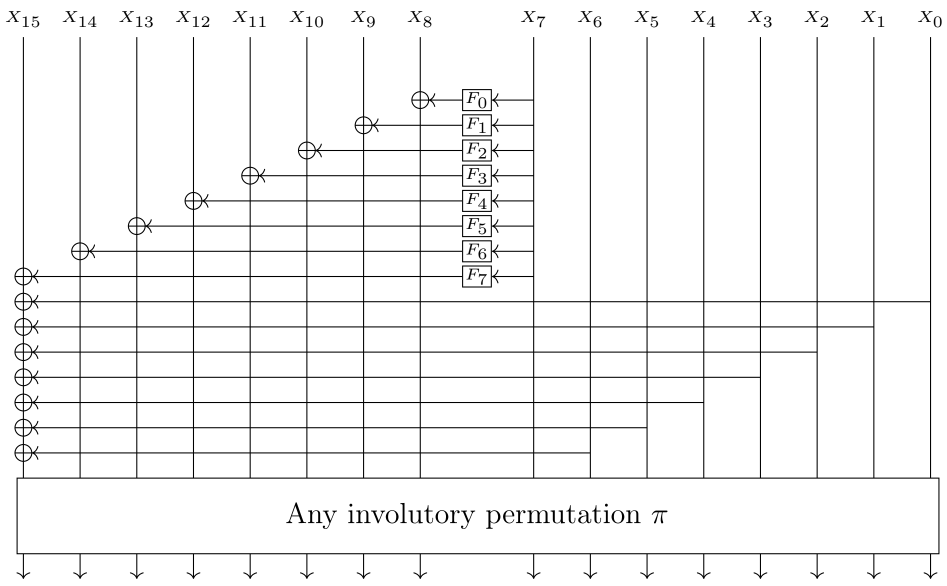
then its diffusion delay d is equal to 4.

Proof. $\mathcal{A}^2 = \begin{pmatrix} FI & \cdots & FI & F^2 \\ \vdots & & \vdots & \vdots \\ FI & \cdots & FI & F^2 \\ FI & \cdots & FI & F^2+(\frac{k}{2}-1)FI \end{pmatrix}$ has no zero coefficients. Writing the permutation layer $\mathcal{P}$ as $\mathcal{P} = \left(\begin{smallmatrix} 0 & \mathcal{P}_1 \\ \mathcal{P}_2 & 0 \end{smallmatrix}\right)$ by definition of π, the matrix of the EGFN becomes $\mathcal{M} = \mathcal{P}\mathcal{N} = \left(\begin{smallmatrix} \mathcal{P}_1\mathcal{A} & \mathcal{P}_1 \\ \mathcal{P}_2 & 0 \end{smallmatrix}\right)$. Computing $\mathcal{M}^3 = \left(\begin{smallmatrix} (\mathcal{P}_1\mathcal{A})^3+\mathcal{P}_1\mathcal{A}\mathcal{P}_1\mathcal{P}_2+\mathcal{P}_1\mathcal{P}_2\mathcal{P}_1\mathcal{A} & (\mathcal{P}_1\mathcal{A})^2\mathcal{P}_1+\mathcal{P}_1\mathcal{P}_2\mathcal{P}_1 \\ \mathcal{P}_2(\mathcal{P}_1\mathcal{A})^2+\mathcal{P}_2\mathcal{P}_1\mathcal{P}_1\mathcal{P}_2 & \mathcal{P}_2\mathcal{P}_1\mathcal{A}\mathcal{P}_1 \end{smallmatrix}\right)$ shows it still has zero coefficients, as $\mathcal{P}_2\mathcal{P}_1\mathcal{A}\mathcal{P}_1$ does. Thus, $d^+ \geq 4$.

Compute then $\mathcal{M}^4 = \left(\begin{smallmatrix} a & (\mathcal{P}_1\mathcal{A})^3+\mathcal{P}_1\mathcal{A}\mathcal{P}_1\mathcal{P}_2\mathcal{P}_1+\mathcal{P}_1\mathcal{P}_2\mathcal{P}_1\mathcal{A}\mathcal{P}_1 \\ b & \mathcal{P}_2(\mathcal{P}_1\mathcal{A})^2\mathcal{P}_1+(\mathcal{P}_2\mathcal{P}_1)^2 \end{smallmatrix}\right)$ with $a = (\mathcal{P}_1\mathcal{A})^4 + (\mathcal{P}_1\mathcal{A})^2\mathcal{P}_1\mathcal{P}_2 + \mathcal{P}_1\mathcal{A}\mathcal{P}_1\mathcal{P}_2\mathcal{P}_1\mathcal{A} + \mathcal{P}_2^2(\mathcal{P}_1\mathcal{A})^2 + \mathcal{P}_2^2\mathcal{P}_1\mathcal{P}_2$ and $b = \mathcal{P}_2(\mathcal{P}_1\mathcal{A})^3 + \mathcal{P}_2\mathcal{P}_1\mathcal{A}\mathcal{P}_1\mathcal{P}_2 + (\mathcal{P}_2\mathcal{P}_1)^2\mathcal{A}$.

Thus $\mathcal{M}^4$ has no zero coefficient because by definition $\mathcal{P}_1$ is of the form: $\mathcal{P}_1 = \begin{pmatrix} & & & 0 \\ & \mathcal{P}'_1 & & \vdots \\ & & & 0 \\ 0 & \cdots & 0 & 1 \end{pmatrix}$.

Thus, as the last row and the last column of $\mathcal{A}$ has no zero, the product $(\mathcal{P}_1 A)^2$ has only nonzero coefficients. Thus, the condition $\pi(k-1) = k/2-1$ implies that $d^+ = 4$. The reasoning is the same on $\mathcal{M}^{-1}$ to prove that $d^- = 4$ and finally that $d = 4$. □



x_{15} x_{14} x_{13} x_{12} x_{11} x_{10} x_9 x_8 x_7 x_6 x_5 x_4 x_3 x_2 x_1 x_0
F_0
F_1
F_2
F_3
F_4
F_5
F_6
F_7
Any involutory permutation π

Figure 8.20: A target-heavy construction that reaches full diffusion in $d = 4$ rounds and has no integral property for 6 rounds

8.2.2 Resistance to differential cryptanalysis

While the constructions shown in Section 8.2.1 displayed attractive features with regards to lightweightness and integral property, we found out that they had very undesirable behaviors regarding differential cryptanalysis. More precisely, when combined with an involutory permutation layer, they allowed for the cancelation of differences.

As an example, let us consider a very simple case, using the $\mathcal{A}$ matrix exhibited in the previous section

$$\mathcal{A} = \begin{pmatrix} & & & F \\ & (0) & & \vdots \\ I & \cdots & I & F \end{pmatrix}$$

and the $\frac{k}{2}$-blocks shift to the left as the permutation layer.

In this target-heavy configuration, the only emitting block that goes through a nonlinear function is x_7 and every other emitting block is simply XORed to x_{15} in the linear layer. Consequently, two active differences set on any pair of emitting blocks that does not include x_7 will be canceled in the linear layer if they are equal. After the permutation layer, these two active differences will then be moved to two receiving blocks, before being switched back to their original location at the end of the second round, since the permutation is only made of transpositions. Figure 8.21 shows this behavior.

To avoid such cases, it is required that one of the active differences eventually goes through a nonlinear function. In the scheme shown above, this is only possible if the difference propagates to x_7 at a given round r, which never can never happen because of the permutation that was chosen. This implies some constraints on the nonlinear layer and the permutation layer:

Proposition 8.19. *Let us consider an* EGFN *with k blocks as defined in Definition 8.13, satisfying Condition 2 of Theorem 8.16:*

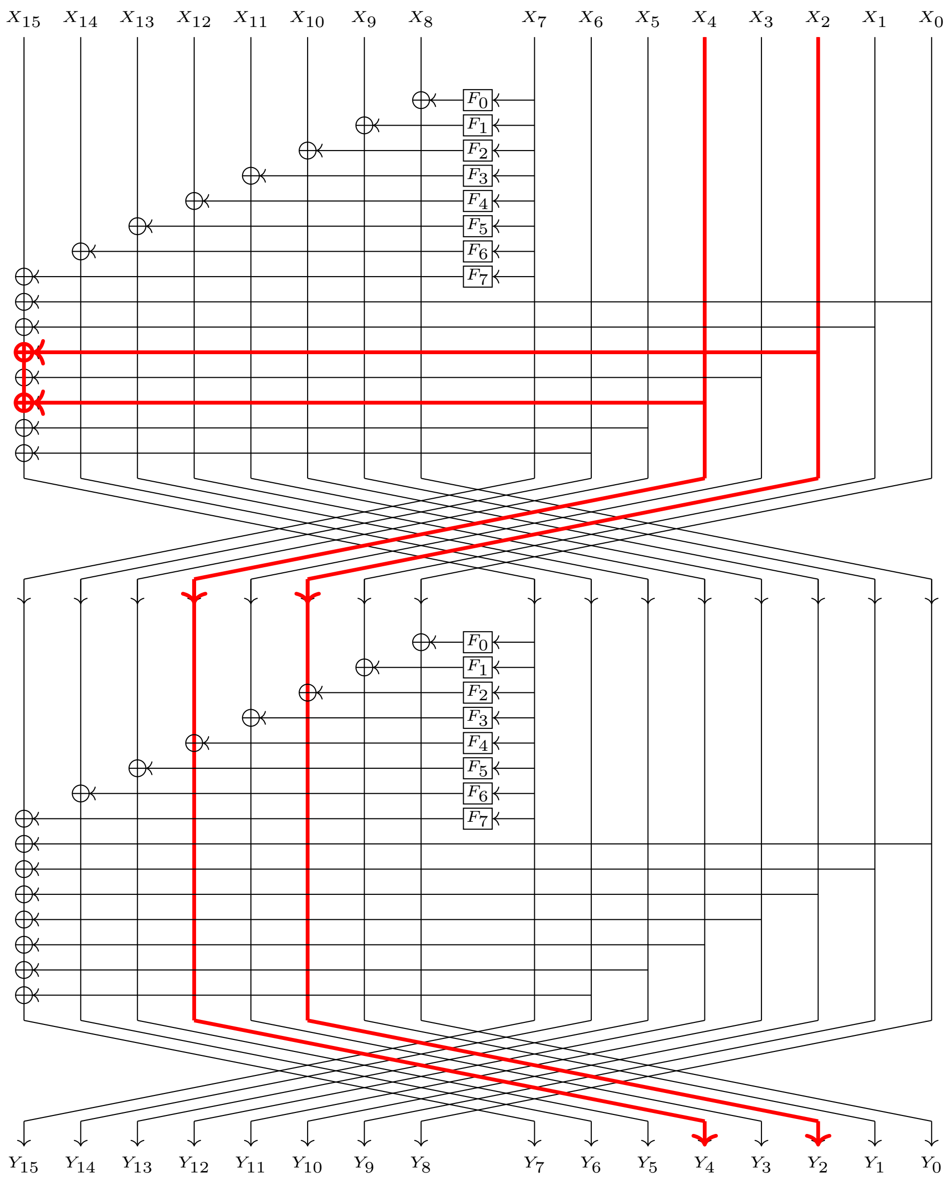


Figure 8.21: Cancelation of two differentials in the linear layer, leading to an iterative probability-1 differential characteristic for the whole cipher.

1. *there must be at least two emitters in the nonlinear layer,*
2. *the permutation used can have at most one cycle that does not contain any of the nonlinear emitters in its decomposition, and in that case, that cycle is a transposition.*

One of the conditions to reach an optimal diffusion delay, described in [BFMT16] and recalled in Theorem 8.16, is that the permutation π globally exchanges emitters (blocks x_0 to $x_{\frac{k}{2}-1}$) with receivers (blocks $x_{\frac{k}{2}}$ to x_{k-1}) with $\pi(\frac{k}{2}-1) = k-1$ and $\pi(k-1) = \frac{k}{2}-1$.

As a consequence, π can be written as a product of cycles of even length, alternating between an emitter and a receiver, with one of the cycles being the transposition $(\frac{k}{2}-1, k-1)$. If π has a cycle of length at least 4 that contains none of the nonlinear emitters, then two differences located on the two linear emitters involved in this cycle will never go through a nonlinear function, thus canceling each other if their values are equal. Therefore, involutions cannot be used unless there are at least $\frac{k}{2} - 1$ emitters in the nonlinear layer.

We used SageMath to study the cancelation of differences for several $\mathcal{A}$ matrices at a symbolic level. Considering a k-block EGFN represented by a symbolic matrix `M=P×N` defined over the polynomial ring in `F`, `d` over $\mathbb{F}_2$—where `F` represents the Feistel functions and `d` a block difference—an input difference Δ can be seen as a vector of k components of the form Δ`=(0,...,0,d,0,...,0)`.

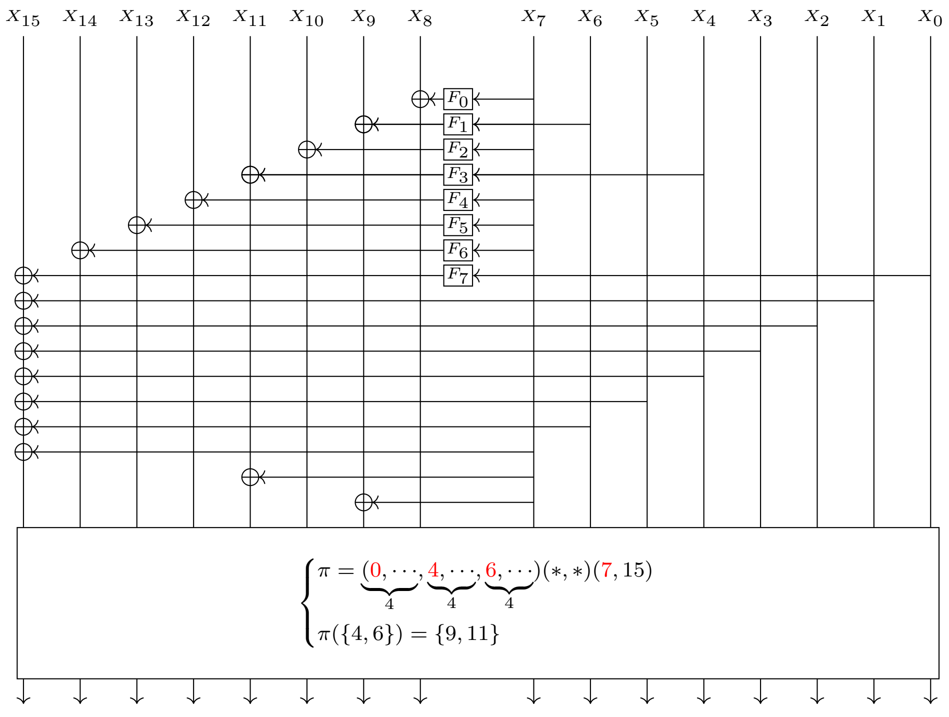
The propagation of a difference Δ after r rounds of encryption can then be tracked by computing $\mathtt{M}^r \times \Delta$. In our approach, we considered every vector Δ with a Hamming weight equal to 2 or 4, and for each scheme, we computed the smallest r value such that $(\mathtt{P}\times\mathtt{N})^r \times \Delta$ has an `F` coefficient, which corresponds to the minimum number of rounds required for a difference to go through a nonlinear function.

The target-heavy construction shown in Section 8.2.1 did not fit the constraints stated in Proposition 8.19, so we tried to investigate alternative structures such as the other matrices depicted in Figure 8.19, with two emitters in the nonlinear layer, namely, x_7 and a second branch that influences x_{15}. Unfortunately, for the set of permutations considered, at least $r = 12$ rounds were required before a difference could actually go through a Feistel function. We also studied EGFNs with more nonlinear emitters. For instance, the one depicted in Figure 8.22 seemed promising, as for some permutations[3], we obtained an `r` value equal to 4. However, it was quite similar to the original EGFN of Lilliput. Since Lilliput had already been studied by other cryptanalysts [ST16, ST17] and only had a slightly higher XOR count, we decided to base our NIST proposal on this primitive instead and thus did not pursue the search for the best permutations for other $\mathcal{A}$ matrices.

8.3 Conclusion

This chapter has investigated of a family of EGFNs identified by Berger *et al.* in [BMT14] with interesting diffusion properties. It was initially started to find lightweight constructions that could be used as round functions for a new authenticated encryption scheme that would later become a Round 1 NIST LWC candidate, Lilliput-AE [ABC+18]. We have tried to summarize the results of our experiments in hopes to exhibit some criteria to design EGFNs with good properties regarding integral and differential cryptanalysis. However, this work is still in its early stages and many observations remain to be explained.

[3] These permutations were the ones that globally exchanged x_4 and x_6 with x_9 and x_{11} as shown in Figure 8.22.



x_{15} x_{14} x_{13} x_{12} x_{11} x_{10} x_9 x_8 x_7 x_6 x_5 x_4 x_3 x_2 x_1 x_0
F_0 F_1 F_2 F_3 F_4 F_5 F_6 F_7
$\begin{cases}\pi = (\underbrace{0,\cdots,}_{4}\underbrace{4,\cdots,}_{4}\underbrace{6,\cdots}_{4})(*,*)(7,15)\\ \pi(\{4,6\}) = \{9,11\}\end{cases}$

Figure 8.22: An EGFN structure with a permutation π that helps avoid the cancelation of equal differences in the internal state.

Conclusion

Derek says it's always good to end a paper with a quote.

Danny Vinyard

In this thesis, the recent challenges of lightweight cryptography have been discussed and a new lightweight proposal has been presented along with some new cryptanalysis results. A large portion of these works, described in Part II, have been carried out in the context of the ongoing Nist Lightweight Cryptography Standardization Process. After designing a proposal, Lilliput-AE, introduced in Chapter 4, some components of other candidates have been studied, namely, Spook and Skinny, in Chapter 5 and Chapter 6, respectively.

Chapter 4. This chapter presented Lilliput-AE, a new Authenticated Encryption with Associated Data (aead) scheme that uses the round function of Lilliput [BFMT16], an algorithm based on an Extended Generalized Feistel Network. Lilliput-AE was part of the 56 Round 1 candidates to the nist lwc standardization process [ABC+18], however it was not selected for Round 2.

Chapter 5. For Spook, a Round 2 candidate in the nist lwc process, my coauthors and I found distinguishers on the underlying permutations, `Shadow-384` and `Shadow-512`, using some structural observations on the preservation of identical states. More precisely, by exploiting the cancellation of round constants, we exhibited distinguishers for the full (6-step) `Shadow-512` and for a round-reduced 5-step version of `Shadow-384`. We could even efficiently distinguish the 6-step permutation if its round constants were shifted. The properties exhibited were then used to build some forgeries in the nonce-misuse setting with a handful of queries, for a 4-step version of the permutation. Extending this attack to another step using similar techniques might be possible and remains to be investigated.

All results have very low, practical complexity and were experimentally verified. Our analysis shows that even when LS-designs seem to avoid the main problem related to round constants, namely invariant subspaces attacks [BCLR17], some unwanted properties may remain, such as the cancellation of their effect in the internal symmetries.

Chapter 6. This chapter focused on the automated search for optimal differential characteristics on the Skinny block cipher, on which several nist lwc proposals are based. To do so, a common approach is to divide the search procedure into two steps: the first step (Step 1) abstracts the difference values into Boolean variables and finds the truncated characteristics with the smallest number of active S-boxes; and the second step (Step 2) tries to instantiate the solutions output by Step 1 to find the best possible probability. Several automatic tools were used to generate the models for Step 1: MILP, SAT, CP and an Ad-Hoc method. Step 2 was modeled solely using CP as it seemed much more efficient than the other methods. With these models, we were able to retrieve some previous results in much less time and we also went further than the highest number of rounds previously reached in other papers. This also provided useful knowledge on the suitable tools for differential cryptanalysis. Overall, the Ad-Hoc method gave the best running times for Step 1 in most cases and thus, became the most efficient tool among the four. However, this method may require a lot of memory. Milp is a good alternative as it is a tool with easier access. For Step 2, the CP-based method was much faster, mainly due to the existing implementation of Table constraints that allowed for an efficient modeling of the S-boxes and their DDT.

Main conclusion of Part II and further research directions. Our results on `Shadow` exhibited in Chapter 5, or the attack on Lilliput-AE described in Section 4.5.1 are testimony to the complex task faced by designers, who must find the right trade-off between security, performance and cost, which is all the more important in the context of lightweight cryptography.

- Regarding Spook, the weakness came from a bad interaction between the round constant addition and the diffusion layer mixing the rows of the state bundles. For `Shadow-512`, the designers opted for the diffusion layer of the low-energy cipher Midori [BBI+15]. This involutory near-MDS matrix[4] was chosen for its efficiency in terms of area and signal-delay, which in turn resulted in a slower diffusion speed and a lower branch number—thus, a smaller minimum number of active S-boxes in each round. Our findings have led the Spook team to switch to an MDS matrix in order to counter our distinguishers and improve the security margins [BBB+20].

- Similarly, a weakness was introduced in Lilliput-AE because of some design choices meant to improve the efficiency of the algorithm and decrease its implementation cost. More specifically, the forgeries presented by Dunkelman *et al.* [DKLS19] exploited the fact that one of transformations used to update the tweakey schedule was the identity function, resulting in the existence of an iterative differential characteristic of probability 1.

These results are also reminders that even though differential cryptanalysis is one of the earliest and most studied technique, assessing the resistance of a primitive against this type of attack remains difficult when designing a new one. Automatic tools such as the ones investigated in Chapter 6 can be helpful in such problems.

The study has shown that finding the best differential characteristic for a given number of rounds is a more intricate task than expected, even with automatic tools. Indeed, while Step 1 usually returns a minimum number of active S-boxes v^* that is quite low, the optimal solution for Step 2 in terms of differential characteristic probability can sometimes be obtained for a number of active S-boxes v that is higher than v^*. As such, computing the optimal bounds is often not enough and we need to go further to be sure that the characteristics found have the best probabilities. However, the possible number of Step 1 solutions tends to grow exponentially when v^* increases. In such cases, the time required to complete Step 2 becomes the bottleneck. Therefore, our approach would greatly benefit from a better filtering for Step 1 in order to decrease the number of solutions as one deviates from the optimal bound. Below are listed other possible extensions of our work.

[4]This type of matrices has been adopted in several lightweight proposals [BCG+12, BBK+13, ADK+14, IMGM15].

- For SAT we chose to use a high level modeling language which did not perform as efficiently as expected since its interface did not implement the `SolveAll` constraint for the chosen SAT solver. As a result, for each Step 1 solution found, the program had to be run again with an added constraint to discard the previously found solution. Using a SAT solver without any other interfaces to be closer to the clauses, instead of using generic translated models as we did, may lead to better performances.

- The chosen tools may work differently for different ciphers, depending on the underlying structure. Performing similar experiments on other ciphers might be valuable.

In the third part of this dissertation, some more general works related to Feistel schemes have been described. In Chapter 7, we have introduced the Feistel Boomerang Connectivity Table (FBCT), which extends the Boomerang Connectivity Table (BCT) to the Feistel case. Then, Chapter 8 has presented a study of EGFNs inspired by the one used in LILLIPUT in an attempt to find lighter, yet more secure, constructions.

Chapter 7. This chapter investigated the computation of the probability of a boomerang switch for generic Feistel ciphers—a problem that had not been addressed so far in spite of the importance of such constructions. While boomerang switches of Feistel ciphers had been studied before, the role of the S-box in a Feistel boomerang switch had previously never been analyzed in a systematic way, as done in the SPN case. As such, my co-authors and I introduced the counterpart of the BCT, namely the FBCT. Our work laid the foundations for this tool by studying its properties. Interestingly, given an input difference Δ_i and an output difference ∇_o, we showed that the coefficient at row Δ_i, ∇_o of the FBCT corresponds to the number of times the second order derivative of the S-box at points (Δ_i, ∇_o) cancels out.

We also showed how to extend a boomerang switch to an arbitrary number of rounds and provided a generic formula to compute the corresponding probability. However, applying this formula over many rounds quickly requires too much computational power. As a result, when evaluating the probability of a switch covering many rounds, the experimental approach is more preferable to the theoretical evaluation of the FBCT—or the BCT for that matter [SQH19]). Still, the FBCT remains a useful tool that can help find S-boxes with the best properties or that are the most likely to lead to incompatibilities for a Feistel cipher.

Chapter 8. The analysis presented in this chapter was initiated during the design of LILLIPUT-AE. The goal was to find new structures inspired by the EGFN used in LILLIPUT [BFMT16], with the same diffusion delay and good resistance against structural attacks but a smaller XOR count. Several interesting structures were found but ultimately, it seemed that the EGFN of LILLIPUT was the most secure scheme. I attempted to derive some general properties for this class of EGFN but could not formalize any.

Main conclusion of Part III and further research directions. Regarding our analysis of the FBCT, some possible work directions include:

- Automating the search of the best parameters for a Boomerang attack using the FBCT. As shown in Chapter 6, the search for the best differential characteristics is a process that can easily be fully automated. Boomerang attacks would greatly benefit from similar tools but as we have seen in Section 7.6, the computation of the size of the middle part of a switch and the corresponding probability can be tricky due to the large number of parameters, which begs to question whether automating this part is feasible or not.

- Applying our formula to the DES and other Feistel ciphers such as CLEFIA, TWINE to possibly find new attacks. In Section 7.7, the FBCT was used to find a 16-round boomerang distinguisher in the single key setting, the related-key case should be investigated as well.
- Extending the FBCT to different types of boomerang distinguishers: in some cases, it might be preferable to consider a boomerang distinguisher that starts from a difference but does not come back to the same difference it has been sent with. For instance, when looking at the FBCT of an Almost Perfect Nonlinear (APN) function, all coefficients are zero apart from those of the first line, the first column and the diagonal, leaving very few possibilities for an attacker to have a switch that comes back. The definition of the FBCT could be extended to cover this scenario.

The study started in Chapter 8 on EGFNs is incomplete; some structures with optimal diffusion delay providing good resistance to integral cryptanalysis have been exhibited but the results obtained so far are still preliminary and mostly focused on involutory permutations. Having some criteria to choose an optimal permutation according to the layout of the nonlinear layer would be helpful. Regarding differential cryptanalysis, some early CP models were made in an attempt to derive differential bounds for some EGFN-based constructions but those were too slow to be exploited. Using an approach similar to the one described in Chapter 6 might lead to better results, although the structures are quite different here.

A

On Non-Triangular Self-Synchronizing Stream Ciphers

This Appendix presents a collaborative work with Loïc Besson, Julien Francq, Phillipe Guillot, Gilles Millérioux and Marine Minier on Self-Synchronizing Stream Ciphers (SSSCs) that was initiated before the Ph.D. but then was refined and submitted during those three years. We propose an instantiation of a dedicated Self-Synchronizing Stream Cipher, called Stanislas, involving an automaton with finite input memory using non-triangular state transition functions. This peculiarity makes our proposal different from existing ones, since the construction of previous dedicated SSSCs was exclusively based on automata with shifts or triangular functions (T–functions) as state transition functions. The construction is based on a general and systematic methodology based on the use of automata (called Linear Parameter Varying, LPV) admitting a matrix representation and a special property of dynamical systems called flatness. Hardware implementations and comparisons with some state-of-the-art stream ciphers on Xilinx FPGAs are presented. It turns out that Stanislas provides bigger throughput than the considered stream ciphers (synchronous and self-synchronizing) when straightforward implementations are considered. Moreover, its synchronization delay is much smaller than the SSSC Moustique (40 clock cycles instead of 105) and the standard approach CFB1-AES128 (40 clock cycles instead of 128). In contrast, it has a rather large hardware footprint.

A.1 Introduction

Self-Synchronizing Stream Ciphers (SSSC) were patented in 1946. The basic principle of such ciphers is to encrypt every plaintext symbol with a transformation that only involves a fixed number of previous ciphertext symbols. Therefore, every ciphertext symbol is correctly deciphered provided that previous symbols have been properly received. This self-synchronization property has many advantages and is especially relevant to group communications. In this respect, since 1960, specific SSSC have been designed and are still used to provide bulk encryption (for Hertzian line, RNIS link, etc.) in military applications or governmental radio mobile networks.

The canonical form of the SSSC combines a shift register, which acts as a state register with the ciphertext as input, together with a filtering function that provides the running key stream. The cryptographic complexity of the canonical form of the SSSC lies in the filtering function. In the early 90s, studies have been performed [Mau91, DGV92] to propose secure designs of SSSCs. These works have been followed by effective constructions ([HPRM04, Sar03, DK08]). What motivates the present proposal for a new SSSC is that, till now, all of these SSSC schemes have been broken ([JM03, JM05, JM06, KRB$^+$08, Klí05]). And up to our knowledge, since 10 years, no other proposals of SSSCs has been made. Clearly, SSSC can be naturally built using a block cipher by applying the Cipher Feedback (CFB) mode. However, the computational cost of CFB is one full block cipher operation per digit. So for single-bit digits, it is n times less efficient than synchronous stream encryption modes such as Output Feedback (OFB) or Counter (CTR) Mode, with n the block length. For example, AES in single-bit CFB mode (as defined as "CFB1-AES128" in NIST SP 800-38a [Dwo01]) is 128 times less efficient than AES in CTR mode. As a consequence, it seems interesting to propose dedicated SSSCs and consider them as a category of primitives on their own.

The aim of the present document is to propose a new framework, along with an instantiation called Stanislas, to design SSSC. The methodology of construction allows to obtain more general automata. Indeed, their state transition functions are not necessarily T–functions as was the case over the past years (see [DK05, DK08] as examples). Due to this peculiarity, the class of admissible automata to act as SSSC is thereby enlarged. It is recalled that a finite automaton with finite input memory M (see [Mau91]), is an automaton which can be represented in a canonical form that consists of an input shift-register of length at least M with an attached memoryless function whose input is the shift-register state. This memoryless function allows to introduce nonlinearities with proved properties, what a shift register does not always permit. The benefit of implementing an SSSC based on a finite automaton rather than the canonical form is that it is more relevant from a computational point of view. The design approach is based on both the special feature of Finite State Machines admitting a matrix representation, called LPV (Linear Parameter Varying) automata and on a property of dynamical systems named flatness. The matrix representation is a generalization of Rational Linear Finite State Machines or Feedback with Carry Shift Registers proposed in [ABL$^+$09, ABMP11]. The use of flatness for the sake of cryptography has been first proposed in [DGM17a]. It has been shown that flatness characterizes the self-synchronization property and that it allows to design automata which can be more general than T–functions.

In the present document, a complete cipher, called Stanislas (for Secure Transmission Algorithm with Non triangular Iterative Structure Looking After Self-synchronization) and designed from the LPV framework, is described. The Key Schedule, the design rationale and the security analysis are provided. Next, hardware implementation results on Xilinx FPGA platforms of Stanislas are performed and compared with some state-of-the art competitors: some Synchronous Stream Ciphers coming from eSTREAM portfolio (TRIVIUM [CP08] and Grain [HJMM08]), one SSSC (Moustique) and the only known feedback mode (the NIST-standardized CFB1-AES128). Interestingly, Stanislas provides the highest throughput on Xilinx Spartan-6 XC6SLX75T FPGAs compared to its stream ciphers and SSSC competitors, implemented in a straightforward manner. Moreover, with the same comparison conditions, its synchronization delay is much smaller than the SSSC Moustique (40 clock cycles instead of 105) and the standard approach CFB1-AES128 (40 clock cycles instead of 128), which provides a decisive advantage for applications when low-latency synchronization is required (e.g., Telecom).

The document is organized as follows. Section A.2 recalls the theoretical results linking together flatness and SSSC. Section A.3 presents the new SSSC Stanislas. The design rationale and the security analysis are detailed in Section A.4. Finally, Section A.5 provides hardware implementation results and comparisons. Section A.6 concludes this document.

A.2 Theoretical Foundations and Flatness

After few generalities on stream ciphers, it is recalled in this section the main results of [DGM17a] concerning the design of SSSC based on the property of flatness.

A.2.1 Generalities on Stream Ciphers

For a stream cipher, it must be given an alphabet A, that is, a finite set of basic elements named symbols. The set A stands in this paragraph as a general notation without any specific alphabet. Typically, A could be composed of 1 or several bits elements. Hereafter, the index $t \in \mathbb{N}$ will stand for the discrete-time. On the transmitter part, the plaintext (also called information or message) $m \in \mathcal{M}$ ($\mathcal{M}$ is the message space) is a string of plaintext symbols $m_t \in A$. Each plaintext symbol is encrypted, by means of an encryption (or ciphering) function e, according to:

$$c_{t+r} = e(z_{t+r}, m_t), \tag{A.1}$$

where $z_t \in A$ is a so-called keystream (or running key) symbol delivered by a keystream generator. The function e is invertible for any prescribed z_t. The resulting symbol $c_t \in A$ is the ciphertext symbol. The integer $r \geq 0$ stands for a potential delay between the plaintext m_t and the corresponding ciphertext c_{t+r}. This is explained by computational or implementation reasons, see [DK05] for example. Consequently, for stream ciphers, the way how to encrypt each plaintext symbol changes on each iteration. The resulting ciphertext $c \in \mathcal{C}$ ($\mathcal{C}$ is called the ciphertext space), that is the string of symbols c_t, is conveyed to the receiver through a public channel.
At the receiver side, the ciphertext c_t is deciphered according to a decryption function d which depends on a running key $\widehat{z}_t \in A$ delivered, similarly to the cipher part, by a keystream generator. The decryption function d obeys the following rule. For any two keystream symbols $\widehat{z}_{t+r}, z_{t+r} \in A$, it holds that

$$\widehat{m}_{t+r} := d(c_{t+r}, \widehat{z}_{t+r}) = m_t \text{ whenever } \widehat{z}_{t+r} = z_{t+r}. \tag{A.2}$$

Equation (A.2) means that the running keys z_t and $\widehat{z}_t$ must be synchronized for a proper decryption. The generators delivering the keystreams are parametrized by a secret key denoted by $K \in \mathcal{K}$ ($\mathcal{K}$ is the secret key space). The distinct classes of stream ciphers (synchronous or self-synchronizing) differ each other by the way on how the keystreams are generated and synchronized. Next, we detail the special class of stream ciphers called Self-Synchronizing Stream Ciphers.

A.2.2 Keystream Generators for Self-Synchronizing Stream Ciphers

A well-admitted approach to generate the keystreams has been first suggested in [Mau91]. It is based on the use of so-called finite state automata with finite input memory as described below. This is typically the case in the cipher Moustique [KRB$^+$04]. At the ciphering side, the automaton delivering the keystream takes the form:

$$\begin{cases} x_{t+1} = f_K(x_t, m_t), \\ z_{t+r} = h_K(x_t) \end{cases} \tag{A.3}$$

where $x_t \in A$ is the internal state, f is the next-state transition function parametrized by $K \in \mathcal{K}$. As previously stressed, the delay b is introduced to cope with special situations, in particular when the computation of the output (also called filtering) delivered by the function h involves

r successive operations processed at time instants $t, \dots, t+r$. Those operations will be here matrix multiplications as detailed later in Equation (A.14). Substituting m_t by its expression (A.2) yields an automaton described by

$$\begin{cases} x_{t+1} = g_K(x_t, c_{t+r}), \\ z_{t+r} = h_K(x_t) \end{cases} \tag{A.4}$$

If such an automaton has finite input memory, it means that, by iterating (A.4) a finite number of times, there exists a function ℓ_K and a finite integer M such that

$$x_t = \ell_K(c_{t+r-1}, \dots, c_{t+r-M}), \tag{A.5}$$

and thus,

$$z_{t+r} = h_K(\ell_K(c_{t+r-1}, \dots, c_{t+r-M})). \tag{A.6}$$

Actually, the fact that the keystream symbol can be written in the general form

$$z_{t+r} \quad = \quad \sigma_K(c_{t-\ell}, \dots, c_{t-\ell'}), \tag{A.7}$$

with σ_K a function involving a finite number of past ciphertexts from time $t-\ell$ to $t-\ell'$ $(\ell, \ell' \in \mathbb{Z})$, is a common feature of the SSSC. Equation (A.7) is called the canonical equation.

Remark 1. *The benefits of implementing the recursive forms* (A.3) *or* (A.4) *instead of directly implementing the canonical form* (A.7) *is that we can obtain nonlinear functions σ_K of high complexity by implementing simpler nonlinear functions f_K or g_K. The complexity results from the successive iterations which act as composition operations.*

At the deciphering side, the automaton takes the form

$$\begin{cases} \widehat{x}_{t+1} = g_K(\widehat{x}_t, c_{t+r}), \\ \widehat{z}_{t+r} = h_K(\widehat{x}_t) \end{cases} \tag{A.8}$$

where $\widehat{x}_t$ is the internal state. Similarly to the cipher part, the automaton having a finite input memory, it means that, by iterating Equation (A.8) a finite number of times, one also obtains

$$\hat{x}_t = \ell_K(c_{t+r-1}, \dots, c_{t+r-M}),$$

and thus,

$$\hat{z}_{t+r} = h_K(\ell_K(c_{t+r-1}, \dots, c_{t+r-M})).$$

Hence, it is clear that after a transient time of maximal length equal to M, it holds that, for $t \geq M$,

$$\widehat{x}_t = x_t \text{ and } \widehat{z}_{t+r} = z_{t+r}. \tag{A.9}$$

In other words, the generators synchronize automatically after at most M iterations. Hence, the decryption is automatically and properly achieved after at most M iterations too. No specific synchronizing protocol between the cipher and the decipher is needed. This explains the terminology Self-Synchronizing Stream Ciphers. The integer M is called the synchronization delay.

Hereafter, the considered automata will be assumed to operate on the q elements finite field $\mathbb{F} = \mathbb{F}_q$ where q is a prime power.

A.2.3 Flat LPV Automata and SSSC

For the automaton described by (A.3) (or the equivalent automaton described by (A.4) after substitution) to get a finite input memory feature (see (A.5)), the solutions proposed in the open literature call for state transition functions g_K in the form of shifts or more generally T–functions (T for Triangle). It is recalled that T–functions are functions that propagate dependencies in one direction only. Till now, none of the proposed SSSCs has involved non-triangular state transition functions although T–functions are known to suffer from weakness [JM06]. Indeed, T–functions induce a propagation of differential properties which make them easier to cryptanalyse. It is explained by the fact that no systematic methodology for constructing finite automata with finite input memory and involving general non-triangular state transition functions was proposed so far. Actually, it has been shown in [DGM17a] that such a problem can be fixed for the particular class of automata called flat LPV (Linear Parameter-Varying) automata.

Let us recall what is a flat automaton.

Definition A.20. *An automaton described by the dynamics f verifying*

$$x_{t+1} = f(x_t, m_t) \tag{A.10}$$

where $x_t \in \mathbb{F}^n$ is the state, $m_t \in \mathbb{F}$ is the input, is said to be flat, if there exists a function

$$\begin{aligned} h:\ & \mathbb{F}^n \times \mathbb{F} \to \mathbb{F} \\ & c_t = h(x_t, m_t) \end{aligned}$$

such that all system variables can be expressed as a function of c_t and a finite number of its backward and forward shifts.

The output c_t is called the flat output. Hence, by definition, there exists a function $\mathcal{F}$ such that

$$x_t \quad = \quad \mathcal{F}\big(c_{t+t_0}, \ldots, c_{t+t_1}\big) \tag{A.11}$$

where t_0 and t_1 are $\mathbb{Z}$-valued integers. A central remark is that (A.11) is nothing but the canonical equation of an SSSC (compare with (A.5)). As a direct consequence, a flat automaton acts as a primitive for designing an SSSC.

LPV automata, defined over a field $\mathbb{F}$, are described by the following state space representation:

$$x_{t+1} = A_{\rho(t)} x_t + B m_t \tag{A.12}$$

$x_t \in \mathbb{F}^n$ is the state vector, $m_t \in \mathbb{F}$ is the input. The matrices $A \in \mathbb{F}^{n\times n}$ and $B \in \mathbb{F}^{n\times 1}$ are respectively the dynamical matrix and the input matrix. The output c_t is defined as

$$c_t = C x_t \tag{A.13}$$

with $C \in \mathbb{F}^{1\times n}$ the output matrix. The matrix B is the input matrix and defines the component x_t^i on which the symbol m_t is added. Let us note that m_t can be added to several components. Such a system is called Linear Parameter-Varying because it is written with a linear dependency with respect to the state vector. The set of all varying parameters of A are collected on a vector denoted by

$$\rho(t) = \left[\ \rho^1(t), \rho^2(t), ..., \rho^L(t) \ \right] \in \mathbb{F}^L$$

where L is the total number of non-zero (possibly varying) entries. Such automata can exhibit nonlinear dynamics. Indeed, the nonlinearity is obtained by defining the varying parameters $\rho^i(t)$ as nonlinear functions $\varphi^i : \mathbb{F}^{s+1} \to \mathbb{F}$ of the output c_t (or a finite number of shifts) $\rho^i(t) = \varphi^i(c_t, c_{t-1}, \cdots, c_{t-s})$ with s a natural number. Let us notice that the notation $\rho^i(t)$ (usual in the literature for LPV systems) is somehow abusive because it does not reflect an explicit dependency with respect to the time t but on quantities, here c_t, indexed with t.

As a simple illustration, the automaton governed by Equation (A.12) with the setting

$$A_{\rho(t)} = \begin{pmatrix} a & 0 \\ \rho^1(t) & \rho^2(t) \end{pmatrix}, \; B = \begin{pmatrix} 1 \\ 0 \end{pmatrix}$$

with a a constant element in $\mathbb{F}$, $\rho^1(t) = c_t \cdot c_{t-1}$ and $\rho^2(t) = (c_{t-2})^2$, is an LPV automaton.

A flat LPV automaton is an LPV automaton of which state vector x_t verifies (A.11).

Let us recall from [DGM17a] the main proposition which allows to define a family of SSSCs based on LPV automata. For brevity, we introduce the following notation. For $t_2 \geq t_1$, denote by $\prod_{l=t_2}^{t_1} A_{\rho(l)}$ the product of matrices $A_{\rho(l)}$ from t_2 to t_1. For $t_2 < t_1$, define $\prod_{l=t_2}^{t_1} A_{\rho(l)} = \mathbf{1}_n$ (the identity matrix of dimension n). Finally, let $\mathcal{T}$ be the scalar defined by $\mathcal{T} = C \prod_{l=t+r-1}^{t+1} A_{\rho(l)} B$.

Proposition 1. *If the* LPV *finite state automaton defined by* (A.12) *is flat, defining the keystream with delay r as*

$$z_{t+r} = C \prod_{l=t+r-1}^{t} A_{\rho(l)} x_t \tag{A.14}$$

and the ciphering function as

$$c_{t+r} = z_{t+r} + \mathcal{T} m_t, \tag{A.15}$$

the set of equations (A.12), (A.14) and (A.15) define the ciphering part of an SSSC.

On the other hand, consider the finite state automaton with internal state $\widehat{x}_t$ with dynamics given by

$$\widehat{x}_{t+1} = P_{\rho(t:t+r)} \widehat{x}_t + B \mathcal{T}^{-1} c_{t+r} \tag{A.16}$$

with

$$P_{\rho(t:t+r)} = A_{\rho(t)} - B \mathcal{T}^{-1} C \prod_{l=t+r-1}^{t} A_{\rho(l)} \tag{A.17}$$

along with the keystream $\widehat{z}_t$ defined as

$$\widehat{z}_{t+r} = C \prod_{l=t+r-1}^{t} A_{\rho(l)} \widehat{x}_t \tag{A.18}$$

and the deciphering function obeying

$$\widehat{m}_{t+r} = \mathcal{T}^{-1}(c_{t+r} - \widehat{z}_{t+r}). \tag{A.19}$$

Then, the set of equations (A.16-A.19) define the deciphering part of an SSSC.

The proof given in [DGM17a] consists in showing that, if the LPV finite state automaton defined by (A.12) is flat, then there exists an integer M such that the synchronization error $x_k - \widehat{x}_{t+r}$ reaches zero after a finite transient time of length M. The integer M is the synchronization delay. Actually, it is shown that flatness is equivalent to the existence of an integer M such that for all $t \geq 0$,

$$P_{\rho(t+M-1:t+M-1+r)} P_{\rho(t+M-2:t+M-2+r)} \cdots P_{\rho(t:t+r)} = 0 \tag{A.20}$$

where the product (A.20) results from the composition of the state transition functions of the deciphering automaton. Let us note that T and r are independent.

This LPV framework for the design of SSSC is new regarding the literature devoted to the design of SSSC. In particular, it really differs from the serial and parallel constructions proposed in the 90s by Maurer [Mau91].

According to Remark 1, implementing the recursive form (A.12) and (A.16) instead of the canonical form (A.5) is more efficient from a computational point of view.

It is recalled that the non-linearity is obtained by defining the values of every varying parameters $\rho^i(t)$ $(i = 1, \ldots, L)$ involved in the matrices of (A.12-A.19) as non-linear functions φ^i of a finite number of past cryptograms $(\rho^i(t) = \varphi^i(c_t, c_{t-1}, \cdots, c_{t-s}))$. Those functions will be implemented in the form of S-boxes denoted S.

$$\begin{array}{rccc} \varphi^i : & \mathbb{F}^{s+1} & \rightarrow & \mathbb{F} \\ & (c_t, c_{t-1}, \cdots, c_{t-s}) & \mapsto & S(c_t, c_{t-1}, \cdots, c_{t-s}, SK_i) \end{array} \tag{A.21}$$

where SK_i is the subkey number i derived from the secret key denoted with K.

The point is that the LPV automaton defined by (A.12) must be flat for any secret key K and any realization of $\rho(t)$. In other words, flatness must be a generic property of (A.12). Designing an LPV automaton (A.12) which is generically flat relies on an admissible realization of a corresponding structured linear system. A structured linear system is a linear system only defined by the sparsity pattern of the state space realization matrices. In other words, for a structured linear system, we distinguish between the entries that are fixed to zero and the other ones that can take any value in $\mathbb{F}$, including the ones which are time-varying. Hence, a structured linear discrete-time system, denoted by Σ_Λ, is a system that admits the form:

$$\Sigma_\Lambda : \quad x_{t+1} \quad = \quad I_A x_t + I_B m_t \tag{A.22}$$

The entries of the matrices of (A.22) are '0' or '1'. In particular, the entries $A(i,j)$ of I_A (*resp.* $B(i)$ of I_B) that are '0' mean that there are no relation (dynamical interaction) between the state x^i_{t+1} at time $t+1$ and the state x^j_t at time t (*resp.* the state x^i_{t+1} at time $t+1$ and the input m_t at time t). The entries that are '1' mean that there is a relation. As a simple example, let us consider again the LPV system with the setting

$$A_{\rho(t)} = \begin{pmatrix} a & 0 \\ \rho^1(t) & \rho^2(t) \end{pmatrix} \text{ and } B = \begin{pmatrix} 1 \\ 0 \end{pmatrix}$$

where a is a constant element in $\mathbb{F}$, $\rho^1(t)$ and $\rho^2(t)$ are varying parameters in $\mathbb{F}$. The dynamical matrix and the input matrix I_A and I_B of the corresponding structured linear system read:

$$I_A = \begin{pmatrix} 1 & 0 \\ 1 & 1 \end{pmatrix}, \; I_B = \begin{pmatrix} 1 \\ 0 \end{pmatrix}$$

As a consequence, if the structural linear system (A.22) derived from (A.12) is flat, the flatness will hold for any $\rho(t)$ or equivalently any nonlinearity φ^i (any S-box will be admissible). Hence, the challenge is to define a methodology to construct flat linear structural systems. It is the purpose of the graph-based approach provided in [MB15] which follows the steps recalled in Appendix A.8. Roughly speaking, given a triplet (n, r, n_a) with n the dimension of the state, r the delay and n_a the number of non-zero entries of the matrix A, a digraph $\mathcal{G}(\Sigma_\Lambda)$ is constructed according to given rules and the matrices I_A and I_B are derived.

The triplet (n, r, n_a), the number of non-linear functions φ^i and their locations in the matrix I_A determine a family of flat LPV-based SSSC. Next subsection aims at summarizing the steps needed for the design of such a family. Then, a particular instantiation, leading to the SSSC called Stanislas, is given in next section.

Summary for the construction of SSSC from a flat LPV-based automaton

Choose a triplet (n, r, n_a) with n the dimension of the state, r the delay and n_a the number of non-zero entries of the matrix A.

Step S1: Choose a component x_t^i on which the plaintext symbol m_t is added. It follows that $B = (0\dots\ 1\ 0\ \dots 0)^t$ (the entries 1 is located at column i).

Step S2: Choose a component x_t^i ($i \in \{1, \dots, n\}$) as the desired flat output $y_t = x_t^i$. It follows that $C = (0 \cdots 0\ 1\ 0\ \dots 0)$ (the only entry 1 is located at the i-th column of C). It can be shown that for the special case when $B = (1\ 0\ \dots)$, i must be equal to r for $y_t = x_t^i$ to be a flat output.

Step S3: Construct the corresponding digraph $\mathcal{G}(\Sigma_\Lambda)$ according to Step 1-5 given in Appendix A.8 and derive the matrices I_A and I_B of the structured linear system Σ_Λ.

Step S4: Replace some of the non-zero entries of I_A by a nonlinear function $\rho^i(t) = \varphi^i(c_t, c_{t-1}, \cdots, c_{t-s})$ to construct the matrix $A_{\rho(t)}$ of (A.12) and set $B = I_B$. Not all '1' entries of I_A must be assigned to a non-linear function. Some of them can be merely constant. The choice must obey a trade-off between complexity of the architecture and security (a matter discussed in next section). Since the construction ensures structural flatness, any choice will preserve the self-synchronization property.

Step S5: Complete the design by deriving the equations of Proposition 1. In particular, calculate the matrix (A.17) governing the state transition function of the automata (A.16) ensuring the deciphering.

Example: Consider the triplet $(n = 2, r = 2, n_a = 5)$. Choose

$$B = \begin{pmatrix} 1 \\ 0 \end{pmatrix}, C = (0\ 1)$$

The particular setting of the matrix C means that the component x_t^2 of the state vector of the LPV system (A.12) is the desired flat output.
For $n_a = 5$, Steps 1-5 given in Appendix A.8 give

$$I_A = \begin{pmatrix} 1 & 1 \\ 1 & 1 \end{pmatrix},\ I_B = \begin{pmatrix} 1 \\ 0 \end{pmatrix} \tag{A.23}$$

Let us keep constant and equal to 1 the first three entries of A and let the fourth entry be a nonlinear function. It is denoted by $\rho^1(t)$. This finally leads to the following matrix $A_{\rho(t)}$:

$$A_{\rho(t)} = \begin{pmatrix} 1 & 1 \\ 1 & \rho^1(t) \end{pmatrix}$$

Calculate $P_{\rho(t:t+2)} = A_{\rho(t)} - BCA_{\rho(t+1)}A_{\rho(t)}$. One obtains

$$P_{\rho(t:t+2)} = \begin{pmatrix} -\rho^1(t+1) & -\rho^1(t)\rho^1(t+1) \\ 1 & \rho^1(t) \end{pmatrix}.$$

Next sections are devoted to the specifications, design rationale and security analysis of a complete SSSC based on flat LPV dynamical systems as described above. The cipher is called Stanislas for Secure Transmission Algorithm with Nontriangular Iterative Structure Looking After Self-synchronization.

A.3 Specification of the flat LPV-based SSSC Stanislas

Stanislas operates over the 16 elements of the finite field $\mathbb{F}_{16}$ defined as: $\mathbb{F}_{16} = \mathbb{F}_2[X]/(X^4+X+1)$, the addition being the componentwise exclusive or, simply denoted $+$, and the multiplication, denoted by $\cdot$, being the polynomial multiplication modulo the primitive polynomial X^4+X+1.

A.3.1 Equations of Stanislas

The internal state $x_t \in \mathbb{F}_{16}^{40}$ of the cipher consists in a vector of dimension $n = 40$ with 4-bit components considered as elements in $\mathbb{F}_{16}$. The input m_t and the output c_t of the ciphering function are 4-bit respective elements in $\mathbb{F}_{16}$.

Ciphering equations

The ciphering equation defines the next internal state $x_{t+1} \in \mathbb{F}_{16}^{40}$ and the cipher output $c_t \in \mathbb{F}_{16}$. Note also that all elements that compose the matrix $A_{\rho(t)}$ and the vectors B and C are elements of $\mathbb{F}_{16}$. The ciphering equation obeys, for $t \geq 0$,

$$\text{cipher:} \quad \begin{cases} x_{t+1} &= A_{\rho(t)} x_t + B m_t \\ c_{t+1} &= C x_{t+1} \end{cases} \tag{A.24}$$

where B is the column vector equal to $B = (1_{\mathbb{F}_{16}}, 0_{\mathbb{F}_{16}}, \ldots, 0_{\mathbb{F}_{16}})^T$ and C is the row vector equal to $C = (0_{\mathbb{F}_{16}}, 0_{\mathbb{F}_{16}}, 1_{\mathbb{F}_{16}}, 0_{\mathbb{F}_{16}}, \ldots, 0_{\mathbb{F}_{16}})$ with the $1_{\mathbb{F}_{16}}$ component located at column $r = 3$. In other words, the only non-zero component of C which equals $1_{\mathbb{F}_{16}}$ is the third component. Hence, the ciphertext symbol consists in the third component of the internal state. Let us note that in general, the ciphertext obeys Equation (A.13). Hence, the ciphertext results from a linear combination of the state vector components. Here, we propose a construction for which the linear combination reduces to the selection of one component. To guarantee a self-synchronization property (otherwise stated, to ensure Equation (A.20)), the component that is selected must coincide with r, the delay of the system. This is why here, $r = 3$.

The matrix $A_{\rho(t)}$ is a 40×40 dimensional matrix. The entries a_{ij} of $A_{\rho(t)}$ are either 0 of $\mathbb{F}_{16}$, or 1 of $\mathbb{F}_{16}$ or a nonlinear function of c_t. Among them, $n_a = 115$ entries are non-zero coefficients and $L = 80$ entries correspond to nonlinear functions φ^i ($i = 1, \ldots, 80$). Each function φ^i depends on the current ciphertext symbol c_t ($s = 0$ in Equation (A.21)) and on a subkey SK_i of K. This subkey SK_i thus defines the corresponding function φ^i which depends on only one ciphertext symbol c_t.

$$\begin{array}{rccc} \varphi^i : & \mathbb{F}_{16} & \rightarrow & \mathbb{F}_{16} \\ & c_t & \mapsto & S(c_t \oplus SK_i) \end{array} \tag{A.25}$$

The subkey SK_i is a 4-bit word derived from the secret key K as described in the Key Schedule of subsection A.3.1 detailed further on. The function S is the bijective S-Box extracted from Piccolo [SIH+11]. It is defined on 4-bit words by Table A.1. The entries a_{ij} according to their location (row i, column j for $i, j \in \{1, \ldots, 40\}$) are given in Appendix A.7.1 in a symbolic manner. In the symbolic representation of $A_{\rho(t)}$, denoted A_S, the functions φ^i ($i = 1, \ldots, 80$) are assigned to the entries of S.

In other words, taking into account the particular structure of the matrices B and C with only one non-zero coefficient, the ciphering process is governed by:

$$\begin{array}{rcl} x_{t+1}^1 &=& \sum_{j=1}^n a_{1j} x_t^j + m_t \\ x_{t+1}^i &=& \sum_{j=1}^n a_{ij} x_t^j, \; i = 2, \cdots, n \\ c_{t+1} &=& C x_{t+1} = x_{t+1}^3 \end{array} \tag{A.26}$$

Table A.1: S-box in hexadecimal notation.

x	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
$S(x)$	E	4	B	2	3	8	0	9	1	A	7	F	6	C	5	D

Deciphering equations

The deciphering is governed by two equations defined, for $t \geq 0$, by

$$\text{decipher:} \quad \begin{cases} \widehat{x}_{t+1} &= P_{\rho(t:t+r)}\widehat{x}_t + B\mathcal{T}^{-1}c_{t+r} \\ \widehat{m}_{t+r} &= \mathcal{T}^{-1}(c_{t+r} - C\prod_{l=t+r-1}^{t} A_{\rho(l)}\widehat{x}_t) \end{cases} \tag{A.27}$$

where the 40×40-dimensional matrix $P_{\rho(t:t+r)}$ over $\mathbb{F}_{16}$ verifies $P_{\rho(t:t+r)} = A_{\rho(t)} - B\mathcal{T}^{-1}C\prod_{l=t+r-1}^{t} A_{\rho(l)}$ and $\mathcal{T} = C\prod_{l=t+r-1}^{t+1} A_{\rho(l)}B$.

Otherwise stated, the deciphering consists of two equations. The first one achieves the computation of the next internal state $\hat{x}_{t+1}$ from the current state $\hat{x}_t$ and the delayed ciphertext c_{t+r}. The second equation ensures the recovery of the plaintext symbol m_t. The self-synchronization between the internal states x_t and $\widehat{x}_t$ of the cipher and the decipher automata and thus a proper decryption are guaranteed, by construction, after a finite transient time.

Let $X[i]$ denotes the i-th row of a matrix X. Taking into account the particular structure of the matrices A, B and C, the delay $r = 3$, and noticing that $\mathcal{T}$ equals one, the deciphering process is governed by:

$$\begin{array}{rcl} \widehat{x}_{t+1}^{1} &=& \left(P_{\rho(t:t+3)}[1] \cdot \widehat{x}_t\right) + c_{t+3} \\ \widehat{x}_{t+1}^{i} &=& \left(P_{\rho(t:t+3)}[i] \cdot \widehat{x}_t\right) \ i = 2, \cdots, 40 \\ \widehat{m}_{t+3} &=& \left(A_{\rho(t+2)}A_{\rho(t+1)}A_{\rho(t)}\right)[3] \cdot \widehat{x}_t + c_{t+3} \end{array} \tag{A.28}$$

where

$$\begin{array}{rcl} P_{\rho(t:t+3)}[1] &=& A_{\rho(t)}[1] - \left(A_{\rho(t+2)}A_{\rho(t+1)}A_{\rho(t)}\right)[3] \\ P_{\rho(t:t+3)}[i] &=& A_{\rho(t)}[i], \qquad 2 \leq i \leq n \end{array} \tag{A.29}$$

Key Schedule

The matrix A consists of $n_a = 115$ non-zero entries, and among them, $L = 80$ are functions φ^i depending on the subkey SK_i. Thus, the Key Schedule process aims at generating 80 subkeys of 4-bit length: $SK_1, \cdots, SK_{80}$ from the 80-bit master key K arranged as 20 words $K_1, \ldots, K_{20}$ of 4-bit length. To do so, the ciphering equations (A.26) with $m_t = 0$ are used. During this process, the parameters SK_i involved in the functions φ^i (see Equation (A.25)) are set to zero. The internal state x_0 is initialized by duplicating the master key K as $(x_0^1 \cdots x_0^{20}) = (K_1 \cdots K_{20})$ and $(x_0^{21} \cdots x_0^{40}) = (K_1 \cdots K_{20})$. Then, the initial internal state x_0 is updated ten times by using equations (A.26) with $m_t = 0$ for $t = 0, \ldots, 9$.

After those ten iterations, the ten subkeys $SK_1, \ldots, SK_{10}$ are respectively initialized with the following components of the internal state x_{10}: x_{10}^1, x_{10}^2, x_{10}^7, x_{10}^{10}, x_{10}^{17}, x_{10}^{18}, x_{10}^{22}, x_{10}^{26}, x_{10}^{28}, x_{10}^{34}. This process is repeated 7 more times to initialize the other subkeys $SK_{10i+1}, \ldots, SK_{10i+10}$, for $i = 0, \ldots, 7$.

A.3.2 Ciphering Process

The plaintext consists of ℓ elements of $\mathbb{F}_{16}$: $m = m_0 \cdots m_{\ell-1}$. The initial state x_0 is first initialized with 40 random elements of $\mathbb{F}_{16}$. These initial values are kept secret and are not transmitted to the decipher even if it is possible to recover the secret key. The way those elements are randomly picked is out of the scope of this document. Only consider that you have a source of randomness. x_0 could be considered as a secret nonce.

Then randomly pick $n-1 = 39$ elements $m_{-39}, \cdots, m_{-1}$ of $\mathbb{F}_{16}$ as the synchronization sequence which is placed before the plaintext. It is recalled that at most $n = 40$ iterations are needed for the self-synchronization to be achieved.

Then, because of the parameter value $r = 3$ that induces a delay, randomly pick $r-1 = 3-1 = 2$ more elements $m_\ell, m_{\ell+1}$ of $\mathbb{F}_{16}$ that will be placed at the end of the plaintext sequence m to process the two last plaintext symbols. Finally, the sequence that must feed the cipher is $m^\star = m_{-39}, \ldots, m_0, \ldots, m_\ell, m_{\ell+1}$.

The resulting ciphertext consists of the sequence $c_{-39}, \ldots, c_{\ell+1}$ of $(\ell+41)$ symbols in $\mathbb{F}_{16}$, computed with the ciphering Equations (A.26) using Matrix A_S given in Appendix A.7.1 for t in $-39, \cdots, \ell+1$:

$$\begin{array}{rcl}
x^1_{t+1} & = & S(x^{20}_t \oplus SK_0) \oplus S(x^{26}_t \oplus SK_1) \\
 & & \oplus S(x^{29}_t \oplus SK_2) \oplus S(x^{40}_t \oplus SK_3) \oplus m_t \\
x^2_{t+1} & = & x^1_t \oplus S(x^2_t \oplus SK_4) \oplus S(x^{31}_t \oplus SK_5) \\
 & & \oplus S(x^{35}_t \oplus SK_6) \\
x^3_{t+1} & = & x^2_t \oplus S(x^3_t \oplus SK_7) \\
x^4_{t+1} & = & S(x^2_t \oplus SK_8) \oplus S(x^3_t \oplus SK_9) \\
 & \vdots & \\
c_{t+1} & = & x^3_{t+1}
\end{array}$$

Note that the Matrix A_S given in Appendix A.7.1 could be seen as one round of a block cipher applied on a state with 40 4-bit words where after each round the 4-bit word x^3 is outputted whereas the 4-bit word x^0 is completely updated by other 4-bit words. At each clock, each 4-bit word crosses at least one S-box, except the 4-bit word x^1 that could be considered as a temporary variable (as it receives the 4-bit message m_t). Note also that the updated rule for most of the variables x^i $(5 \le i \le 40)$ could be written as $x^i_{t+1} = S(x^{i-1}_t \oplus SK_k) \oplus f(x^{i-2}_t)$ or $S(x^{i-1}_t \oplus SK_k) \oplus f(x^{i-2}_t) \oplus S(x^j_t \oplus SK_{k'})$ for given j, k and k' and where f is the identity or the S-box S. It could be seen as a generalization of the so-called L-scheme with a circular permutation used in the block cipher MISTY1 with balanced inputs/outputs.

The complete ciphering process of Stanislas is illustrated on Fig. A.1.

A.3.3 Deciphering Process

The decipher receives the cryptogram consisting of $\ell+41$ symbols $c_{-39}, \ldots, c_{\ell+1}$ in $\mathbb{F}_{16}$. The internal state $\widehat{x}_0$ is initialized to an arbitrary value, for example the zero value.

Then, the deciphering Equations (A.28)-(A.29) are applied to recover a $\ell+41$-length message $\widehat{m} = \widehat{m}_{-41}, \ldots \widehat{m}_{\ell-1}$. The plaintext sequence is recovered as the last ℓ symbols $m = \widehat{m}_0 \ldots \widehat{m}_{\ell-1}$.

Remark 2. *It could be surprising that a part of the ciphering process directly depends on a secret nonce (i.e. x_0). Instead, we could imagine that an 80-bit IV is used to generate the first value x_0 adding an IV schedule to the Key Schedule process. First, generate the subkeys using the Key Schedule, then initialize the internal state with the concatenation of the IV and of the key. Then, apply again the Key Schedule process (but including the generated subkeys SK_i in the S-boxes) to initialize the internal state x_0. But, note that in this case, the particular property that the internal states (that must be kept secret) of the ciphering part and of the deciphering part are not required to be equal is lost. Indeed, in this case where an IV is used, we will suppose that the internal state will be computed in the same way in both sides.*

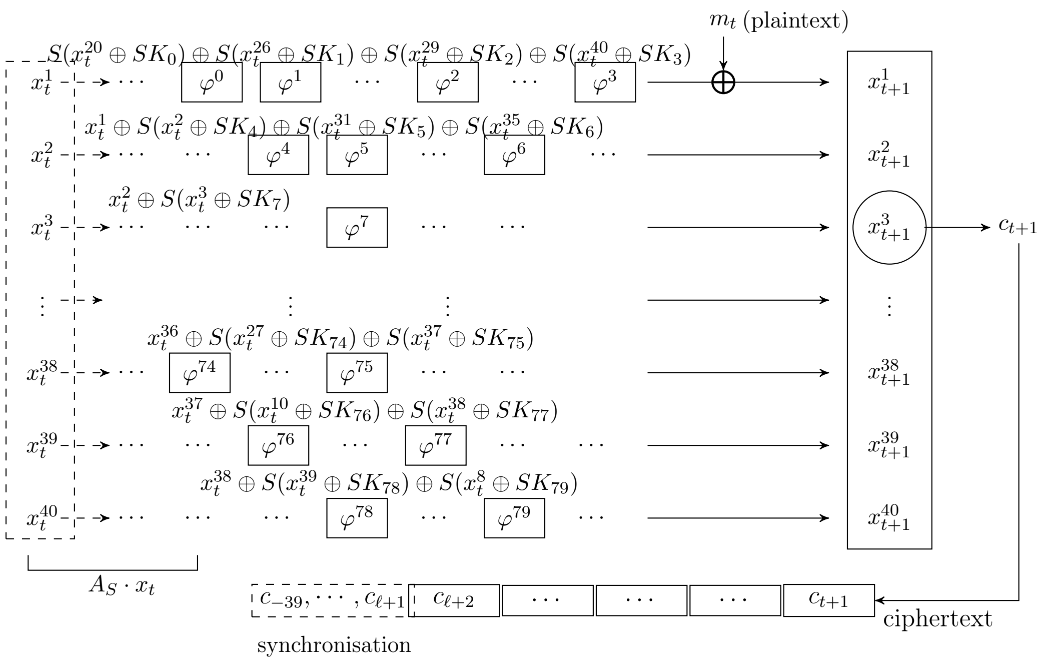

m_t (plaintext)
$S(x_t^{20} \oplus SK_0) \oplus S(x_t^{26} \oplus SK_1) \oplus S(x_t^{29} \oplus SK_2) \oplus S(x_t^{40} \oplus SK_3)$
$x_t^1 \oplus S(x_t^2 \oplus SK_4) \oplus S(x_t^{31} \oplus SK_5) \oplus S(x_t^{35} \oplus SK_6)$
$x_t^2 \oplus S(x_t^3 \oplus SK_7)$
$x_t^{36} \oplus S(x_t^{27} \oplus SK_{74}) \oplus S(x_t^{37} \oplus SK_{75})$
$x_t^{37} \oplus S(x_t^{10} \oplus SK_{76}) \oplus S(x_t^{38} \oplus SK_{77})$
$x_t^{38} \oplus S(x_t^{39} \oplus SK_{78}) \oplus S(x_t^{8} \oplus SK_{79})$
$A_S \cdot x_t$
c_{t+1}
ciphertext
$c_{-39}, \cdots, c_{\ell+1}$
$c_{\ell+2}$
synchronisation

Figure A.1: The complete ciphering process of Stanislas with $\varphi^i(x_t^j) = S(x_t^j \oplus SK_i)$.

The matrix A consists of $n_a = 115$ non-zero entries, and among them, $L = 80$ are functions φ^i depending on the subkey SK_i. Thus, the Key Schedule process aims at generating 80 subkeys of 4-bit length: $SK_1, \cdots, SK_{80}$ from the 80-bit master key K arranged as 20 words $K_1, \ldots, K_{20}$ of 4-bit length. To do so, the ciphering equations (A.26) with $m_t = 0$ are used. During this process, the parameters SK_i involved in the functions φ^i (see Equation (A.25)) are set to zero. The internal state x_0 is initialized by duplicating the master key K as $(x_0^1 \cdots x_0^{20}) = (K_1 \cdots K_{20})$ and $(x_0^{21} \cdots x_0^{40}) = (K_1 \cdots K_{20})$. Then, the initial internal state x_0 is updated ten times by using equations (A.26) with $m_t = 0$ for $t = 0, \ldots, 9$.

A.4 Design Rationale and Security Analysis

In this section, we motivate the choices of the field on which the cryptosystem operates, the dimension n of the internal state, the delay r and the structure of the matrix A. Most of the choices rest on security criteria, other ones take into account practical considerations, regarding in particular the hardware implementation issues.

A.4.1 Design Rationale

Field on which the cryptosystem operates: Galois field $GF(16)$. Any quantities m_t, c_t, components of x_t and $\hat{x}_t$ and non-linear functions φ^i (S-boxes) inputs are 4-bit data. It is motivated by the fact that the cryptosystem is intended to be implemented on a digital equipment. Hence, field extensions and so, power of two are required. On the other hand, 8-bit would be too heavy for an embedded algorithm. In particular, S-boxes would involve too many logic gates.

Dimension n: 40. As the internal state components are 4-bit words, a dimension $n = 40$ provides an internal state of 160 bits and thus a security level of 80 bits. Indeed, to prevent time-memory trade-off attack [Hel80] (an attack which is a trade-off between exhaustive search and table look-up), the internal state must be two times longer than the key length which defines the security level. This level is compatible with a real-world application.

Delay r: 3. The more the delay, the more the algebraic degree of the entries of $P_{\rho(t:t+r)}$, recalling that $P_{\rho(t:t+r)}$ involves the product of r matrices (see (A.29)). Thus, for a good resistance against an algebraic attack ([CKPS00]), the algebraic degree should be as large as possible. On the other hand, the more the delay, the more the complexity of implementation and the less the computational performances. The delay $r = 3$ results from a trade-off between security with respect to algebraic attacks (to increase the overall algebraic degree) and ease of implementation (especially the implementation of the P deciphering matrix which involves several S-box multiplications (see Equation (A.17) in the general case and (A.29) for Stanislas)).

Structure of the matrices A and P. We recall that the matrix A (and thus P from the computation (A.29)) is derived from the construction of a digraph (see Appendix A.8) from which an adjacency matrix I_A is extracted. More precisely, the adjacency matrix I_A determines the entries of $A_{\rho(t)}$ that are zero and the others that are possibly non-zero. The number n_a of edges in the digraph $\mathcal{G}(\Sigma_\Lambda)$ corresponds to the number of non-zero entries of I_A. Hence, the number n_a also determines the number of non-zero entries of the state transition matrix P. Beyond the number of non-zero entries, their location (row and column) must also be chosen. Finally, it must be decided whether a non-zero entry will be 1 or will correspond to an S-box. All those issues have been addressed by considering several criteria regarding the security, in particular the good resistance to classical attacks and the good diffusion delay, while satisfying a trade-off with respect to the computational complexity for the sake of implementation. Let us introduce symbolic representations of $A_{\rho(t)}$ and $P_{\rho(t)}$ denoted by A_S and P_S where the coefficients of A_S and P_S belong to $\mathbb{Z}[S]$, S representing any non-linear function. The following considerations on A_S and P_S can be made.

- **Diffusion Delay and Depth.** The diffusion delay and the depth are properties related to the consideration of the powers A_S^p and of P_S^p as p, a natural integer, increases. Indeed, the power of matrices results from the successive iterations of the ciphering and the deciphering process. Let p denotes the power of a matrix $Z \in M_n(GF(16))$. The diffusion delay, introduced in [ABMP11], is the smallest value, denoted by d_0, of p such that Z^p does not have any zero coefficient. In other words, it is the smallest value of p such as each element of the initial internal state x_0 has influenced every element of x_t for $t \geq d_0$. The depth, introduced in [BM15], is the smallest value, denoted by d_1, of p such that any entry of Z^p are polynomials of degree at least 1. We are looking for the smallest values of d_0 and d_1.

- **Algebraic Degree.** Considering the matrix resulting from successive powers of A_S and P_S, we are first interested in ensuring that at least one entry has the largest algebraic degree. To this end, we must add a cycle on the r-th vertex of the digraph $\mathcal{G}(\Sigma_\Lambda)$, which equivalently means that the entry of A_S and P_S located at row r and column r must be an S-box. Furthermore, the evolution of this quantity, after successive iterations, must meet an ideal shape: it must increase by one at each iteration, must remain constant and equal to its maximum value as long as possible and finally must drop down to zero (let us recall that after 40 iterations, due to (A.20), the product reaches exactly zero).

- **Full Rank Matrix.** The fact that A_S is a full-rank matrix is a necessary condition to ensure a full diffusion of the internal state and to maximize the dependency between the involved terms at time t and the involved terms at time $t+1$. Moreover, this condition guarantees that the encryption process does not collapse. By construction (see Appendix A.8) to ensure flatness, every element of the subdiagonal of I_A from the r-th one is 1. Hence, for A_S to be full-rank, each column and each row must contain at least one non-zero element. And yet, by construction, only the $r-1$ first elements of the last column can be non-zero. Hence, one of them must be non-zero. From the digraph point of view, it means that at least one of the $r-1$ first vertices must be connected to the last vertex.

The symbolic matrix A_S which has been finally selected is given in Appendix A.7.1. It has been obtained after 700000 random runs performed under the aforementioned constraints: best diffusion delay and depth, algebraic degree (especially increase by 1 at each iteration), full rank matrix. Several matrices correspond to the best choices (we add a sum indicator without weighing) and we finally choose the one with the best implementations for A_S and for P_S. The symbolic matrix P_S can be directly obtained by considering Equations (A.29). The matrix A_S involves $n_a = 120$ non-zero entries and its number of S-boxes is $L = 80$. The matrix extracted from A_S and P_S by removing the first r rows and columns have a special feature. The lower subdiagonal is full of coefficients S (corresponding to S-boxes and denoted SB in the symbolic representation) and the subsubdiagonal just above is full of 1. Thus, each line has at least one S-box to ensure that the internal state x_t is updated in a non-linear way: this corresponds to a non-linear shift register.

The corresponding diffusion delay, $d_0 = 7$, could be considered as a good diffusion delay with regard to the synchronization delay that equals 40 (the system dimension). Indeed, the worst diffusion delay is equal to 40 and the best diffusion delay of 1 could only be reached with a matrix full of non-zero coefficients. Thus, we consider that a diffusion delay of 7 is a sufficiently good compromise between the best diffusion delay and a reasonable number of non-zero coefficients.

Let us notice that the dimension n of the system is the upper bound of the diffusion delay. Indeed, due to (A.20), the product reaches exactly zero in $GF(16)$ since the synchronization is achieved after at most 40 iterations. The depth is $d_1 = 7$ and is the best that we manage to achieve.

S-box. Various classifications of 4-bit S-boxes exist in the literature [Can07, LP07, Saa]. For the sake of hardware optimization, the same S-box has been used to define the $L = 80$ nonlinear functions φ^i. Two kinds of criteria have been considered: theoretical and practical. On one hand, we have selected S-boxes to satisfy the maximum differential probability and the maximum (absolute) linear bias 2^{-2}, the algebraic degree 3, and no fixed-point. Four S-boxes that satisfy those criteria and that have simple algebraic expressions (i.e. a minimal number of non-linear transformations) have been selected, each one corresponding respectively to the four classes proposed in [Saa].

From an implementation point of view, it turns out that the S-box depicted in Table A.1 induces the smallest gate count. It is the Piccolo S-box ([SIH$^+$11]). The area is around 23 Gate Equivalents (GEs)[5]. It involves four NOR gates, three XOR gates and one XNOR gate. Let us notice that the masking method can be applied using only three shares, making this S-box suitable for efficient threshold (i.e. side-channel protected) implementations.

[5]Section A.5 describes the meaning of GE metric.

Key Schedule. The Key Schedule has been chosen to reuse existing circuit while sufficiently mixing together the key words. To do so, we use the already implemented matrix A by applying it a sufficient number of times when looking at the diffusion degree and at the induced algebraic degree. The extracted 4-bit words of the internal state x_t - at positions 1, 2, 7, 10, 17, 18, 22, 26, 28, 34 - have been chosen to be among the ones that depend of the maximum number of other elements of x_t to ensure to maximize the diffusion effect of the initial state x_0.

From a theoretical point of view, if we consider that each S-box behaves as a random function (*i.e.* it has a behavior sufficiently near the one of a true random function) and using the direct extension of Lemma 9 and Theorem 7 of [Mau02], we could say, that after $d_0 + 2$ of the matrix A on the input x_0, the applied transformation behaves as a random function. In other words, the operations done to fulfill the subkey words behaves as a random function.

A.4.2 Security Analysis

The following section focuses on the security of Stanislas against known attacks. We claim a 80-bit security level which corresponds to the key length (we could not have a security level greater than the key length). Moreover, this security level is also achieved regarding the length of the main register: 160 bits. Indeed, the Guess-and-Determine attacks described in [Hel80] imply to double the length of the used register compare to the key length to achieve a security level corresponding to the key length. Thus, we try to derive security bounds for all known attacks against Stanislas and we found no attacks that work with a complexity smaller than 2^{80} operations which corresponds with our security claim.

Moreover, it has been proven in [DGM17b] that the canonical form of an SSSC is secure against Chosen Plaintext Attacks (IND-CPA secure) but not against Chosen Ciphertext Attacks (IND-CCA security). We do not claim any security result in this last model. Moreover, we suppose that the attacker has no access to the key and to the initial value of the internal state x_0. To prevent collision search attack, we limit the size of each plaintext to 2^{64} 4-bit words.

First, it seems very difficult to analyze the security of Stanislas in its true settings (i.e. the 160-bit internal state x_0 is a secret nonce). In those settings, we could only say that:

- the time-memory-data trade-off attacks described in [Bab95, BS00, Hel80] apply when the internal state is smaller than two times the key length. This is why we choose the length of the internal state to be twice the key length to prevent this kind of attacks.

- Guess-and-Determine Attacks [Hel80] consist in guessing a part of the state to further determine the remaining part of the state. Thus, at time t, suppose that we know x_t^2, x_t^3 and of course, c_t. Thus, we could suppose that we observe n consecutive outputs from c_t to c_{t+n}. Thus, how much does it cost to recover the induced subkey SK_i? First, from the equation $x_{t+1}^3 = x_t^2 \oplus S(x_t^3 \oplus SK_7)$, we could directly compute SK_7. Thus, we could derive the successive value of x_t^2 from t to n. Thus, from x_t^2 we derive non-linear equations where the unknowns are x_t^1, x_t^{31}, x_t^{35}, SK_5, SK_4 and SK_6 and the known terms are x_t^2 to x_{t+n}^2. Thus, as the Algebraic Normal Form (ANF) of the S-box has 2 equations with 4 terms, 1 equation with 9 terms and 1 equation with 7 terms, each new x_t^2 will induce a system of $4 \times 3 \times (n+1) + 3 \times 4 = 12n + 24$ unknown binary variables with 4 equations with in total 24 non-linear terms. Thus, we could not solve the system without guessing a part of it whatever the n value. Even considering that we guess a part of the system, the combinatorial explosion seems clear because each value of the internal state depend on at least one other value. Thus, we conjecture here that Stanislas is safe against this type of attacks since, even if a part of the internal state is guessed, each 4-bit word of this state is updated through a XOR with a 4-bit word of subkey and an S-box. Moreover, since the diffusion delay of A_S is $d_0 = 7$, this leads to guess after 7 outputs all the key. Thus, guessing a part of the state allows to guess a part of the key which, in then, amounts to a guess-and-determine attack with a complexity greater than the key exhaustive search.

Thus, instead, when looking at classical attacks, we will use the model described in Remark 2 where the initial state is derived in its first components after 20 iterations of the matrix A_S applied on the concatenation of an IV and of the master key K and in its last components after 14 more iterations. Those settings could be considered as a degrading mode of the original Stanislas specifications. Thus considering that the attacker has full access to the IV, we obtain the following bounds against classical attacks:

- **Differential / Linear Cryptanalysis:** we compute the lower bounds on the minimal number of active S-boxes for the computation of the internal state with Remark 2. To do so, we implement the model of Remark 2 using Constraint Programming to explore all the possible paths in the induced graph. Then, we obtain that, for the differential case, after 7 iterations, a minimal number of 38 S-boxes has been crossed, after 10 iterations, 46, after 14 iterations, 65. As the differential probability of the chosen S-box is equal to 2^{-2}, we could guarantee that the 80-bit key exhaustive search is less expensive than passing through more than 40 S-boxes, which is the case after 10 iterations of the A_S matrix. In the same way, for the linear case, we obtain after 7 iterations, 35 active S-boxes, after 10 iterations, 41 and after 14 iterations, 59. Thus, for the same reason, after 10 iterations, a 80-bit key exhaustive search is more efficient.

- **Algebraic Attacks:** This kind of attacks [CKPS00] is possible when the overall degree of the induced system of equations does not sufficiently increase at each clock. Especially, if the overall degree d in each of the n unknown variables (the key variables for example) of the system is such that $(n^d)^{2.5}$ is lower than the security bounds, it means that it is faster to solve the induced system by Gaussian elimination (considering that each new monomial is a new unknown variable) than trying all the keys of the system. Thus, we want to prevent this attack from happening as described below. The algebraic degree of each S-box component is the best one: equal to 3. Thus, each passing through an S-box increases the degree in the equations describing the internal state and in the equations describing the key. Even if the SK_i linearly depends on the master key bits $K_1, \cdots, K_{80}$ (there are 80 key bits that are unknown), if we write the number of variables after crossing the first S-box, we have $(80)^3$ monomials depending on the unknown key bits, after the second pass we obtain $(80)^6$ monomials also depending on the unkonwn key bits and so on. Thus, if we apply those estimations using the bounds on the number of active S-boxes of the differential/linear case, after 10 iterations, we have 46 active S-boxes, which means that the number of unknowns (considering that each new monomial is a new unknown) is lower bounded after 10 iterations by $(80)^{3\times 46} \approx 2^{872,16}$ where 80 are the unknowns coming from the master key, 3 is the algebraic degree of the S-box and 46 is the number of crossed S-boxes. Thus, we conjecture that the complexity of the best algebraic attack is greater than the 80-bit key exhaustive search.

- **Cube Attacks:** As established in [DS09], a cipher is vulnerable to cube attacks if an output bit can be represented as a sufficiently low degree polynomial over $GF(2)$ of key and input bits. It works by summing an output bit value for all possible values of a subset of public input bits, chosen such that the resulting sum is a linear combination of secret bits. Repeated application of this technique gives a set of linear relations between secret bits that can be solved to discover these bits. In [SMT19], the authors analyzed this kind of attacks on the block cipher Piccolo, especially its S-box. They stated that after 8 rounds, no relation with 63 input bits could be found. The minimal number of S-boxes crossed for 8 Piccolo rounds is 58 whereas in our case, it is 46 after 10 iterations. So we conjecture that we cross a sufficient number of S-boxes after few iterations to prevent having low degree relations between secret key bits and public input bits.

In summary, we conjecture that most of the usual attacks which apply in the stream cipher context have a complexity greater than the exhaustive key search for Stanislas.

A.5 Hardware Performance and Implementation Aspects

We give hereafter the implementation results of a straightforward implementation of Stanislas. It produces one 4-bit word of ciphertext per clock cycle. Subkeys are computed in the initialization step using the Key Schedule and stored in dedicated registers, before the cipher state processing. The same material is used for the cipher state and the Key Schedule processes. The hardware implementation of Stanislas is not an optimized version targeting any specific performance. The main area occupation comes from the matrix update as it carries big registers during all the calculations. Those registers are the internal state which are mixed with the subkeys either with binary addition or multiplication. This means updating a 40×4 bits register all along the matrix update. During ciphering process, the matrix update is solely made of S-boxes and XOR of 4-bits words. The lowest line of the matrix in terms of area occupation is made of 1 S-box and 2 additionnal XORs, the biggest of 4 S-boxes and 8 additional XORs. The deciphering process implies adding 23 multiplications, 28 S-boxes and 39 XORs to the previous total which makes deciphering heavier in terms of area occupation. The matrix is implemented line by line, calculated straightfowardly following the equations as depicted in Fig. 1. The S-boxes are implemented in a Look-Up-Table (LUT) way, so we let the compiler do its own optimizations. Our Stanislas implementation combines both the encryption and decryption process in order to ease comparisons with SSSCs.

We implemented Stanislas in VHDL and we provide in Table A.2 FPGA hardware implementations and performance comparisons with synchronous SCs TRIVIUM [CP08] and Grain [HJMM08], final members of the eSTREAM portfolio, another SSSC Moustique and the AES-based SSSC CFB1-AES128 as defined in NIST SP 800-38a [Dwo01]. The chosen FPGA platform for our benchmark is the Xilinx Spartan-6 XC6SLX75T, package FGG676. High effort of the Xilinx ISE Design Suite has been put on area reduction. Post-place-and-route results are provided.

To get TRIVIUM, Grain, Moustique and CFB1-AES128 implementation results, we have designed our own VHDL straightforward reference implementations, without further optimization in mind. For this latter, we have implemented the potential S-boxes as LUTs, to be consistent with the S-boxes LUT implementations of Stanislas.

At first sight, we can check that some well-known properties are visible in the results. For example, TRIVIUM and Grain are very compact, which can be explained by their low combinatorial gate counts. Moreover, the number of initialization cycles needed for TRIVIUM, Grain and Moustique is consistent with the specifications: e.g., TRIVIUM needs a warm-up phase of minimum 1152 steps. This number is really low for Stanislas where it just consists in a Key Schedule and it is equal to 0 for CFB1-AES128 where the key is processed on the fly.

Surprisingly, straightforward implementation of Stanislas provides the best throughput (TP) compared to the other stream ciphers, even self-synchronizing. The reason is that one 4-bit word

	Area (slices)	**Init. (cycles)**	**Synchro. (cycles)**	**Freq. (MHz)**	**TP (Mbps)**
Trivium	47	1603	0	191	191
Grain	48	256	0	355	355
Moustique	166	105	105	309	309
CFB1-AES128	745	0	128	73	849
Stanislas	701	66	40	95	380

Table A.2: Xilinx Spartan-6 XC6SLX75T (FPGA) Straightforward Implementation Results.

is processed by clock cycle, so the throughput is given by: 95 $\times$ 4 (bits) = 380 Mbps. That justifies our design choice of processing 4-bit words instead of individual bits. Compared to the standard approach CFB1-AES128, the time needed for synchronization is also shorter, along with a smaller area.

This encouraging result has to be mitigated if we consider the combined metric TP/Area as shown on Table A.3. This latter allows to estimate the cost of optimized parallel implementations, where many bits can be processed in parallel, at the expense of additional area.

	TP/Area (Mbps/slice)
Trivium	4.06
Grain	7.39
Moustique	1.86
CFB1-AES128	1.13
Stanislas	0.54

Table A.3: Combined Metric TP/Area (Mbps/Slice on Xilinx Spartan-6 XC6SLX75T FPGA).

As we can see, due to its lowest TP/Area value, straightforward implementation of Stanislas will suffer from the comparisons of its competitors' optimized versions. We can then estimate, in Table A.4, the theoretical implementation results of all Stanislas competitors when all of them are unrolled versions which process 4 bits per clock cycle, as Stanislas.

	Area (slices)	**TP (Mbps)**
Trivium	188	764
Grain	192	1420
Moustique	664	1236
CFB4-AES128	2980	3396
Stanislas	701	380

Table A.4: Theoretical Implementation Results of some (SS)SCs on Xilinx Spartan-6 XC6SLX75T FPGA for 4-bit Versions

As we can see, CFB4-AES128 mode has a 4-fold throughput speedup, beating Stanislas proposal with a significant margin, and it requires only 32 steps for synchronization. But it implies to occupy a big amount of FPGA slices, which is not always affordable for some constrained applications.

Future works will include the study of the cost of side-channel protected Stanislas implementations.

A.6 Conclusion

An instantiation, called Stanislas, of a dedicated Self Synchronizing Stream Ciphers (SSSC) has been proposed. Its main peculiarity comes from the fact that it involves an automaton with finite input memory using non-triangular state transition functions. The construction is based on a general and systematic methodology that uses automata (called Linear Parameter Varying, LPV) admitting a matrix representation and a special property called flatness. The security analysis allows to conjecture that most of the usual attacks which apply in the stream cipher context have a complexity greater than the key exhaustive search for Stanislas. But, Stanislas could not be considered having a small hardware footprint.

However, when straightforward implementations are considered, Stanislas provides bigger throughput than the considered stream ciphers, and its intrinsic synchronization delay is much smaller than the SSSC Moustique (40 clock cycles instead of 105) and the standard approach CFB1-AES128 (40 clock cycles instead of 128).

Moreover, the number of surviving Self Synchronizing Stream Ciphers after a phase of public cryptanalysis time is equal to zero. So, we hope that Stanislas will be the first one and we encourage the symmetric key cryptographic community to cryptanalyze it.

A.7 Appendix

A.7.1 The Matrix A_S

The matrix A_S is given in Fig. A.2.

```
[ 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0 SB  0  0  0  0  0 SB  0  0 SB  0  0  0  0  0  0  0  0  0  0 SB]
[ 1 SB  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0 SB  0  0  0 SB  0  0  0  0  0]
[ 0  1 SB  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0]
[ 0 SB SB  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0]
[ 0 SB  1 SB  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0]
[ 0 SB  0  1 SB  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0]
[ 0  0 SB  0  1 SB  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0]
[ 0  0  0  0 SB  1 SB  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0]
[ 0  0  0  0  0 SB  1 SB  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0]
[ 0  0  0 SB  0  0  0  1 SB  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0]
[ 0  0  0  0  0  0 SB  0  1 SB  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0]
[ 0  0  0  0  0 SB  0  0  0  1 SB  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0]
[ 0  0  0 SB  0  0  0  0  0  0  1 SB  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0]
[ 0  0  0  0  0 SB  0  0  0  0  0  1 SB  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0]
[ 0  0  0  0  0  0  0  0  0  0  0  0  1 SB  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0]
[ 0 SB  0  0  0  0  0  0  0  0  0  0  0  1 SB  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0]
[ 0  0  0 SB  0  0  0  0  0  0  0  0  0  0  1 SB  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0]
[ 0  0  0  0  0  0  0  0  0  0  0 SB  0  0  0  1 SB  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0]
[ 0 SB  0  0  0  0  0  0  0  0  0  0  0  0  0  0  1 SB  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0]
[ 0  0  0  0  0  0  0  0 SB  0  0  0  0  0  0  0  0  1 SB  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0]
[ 0 SB  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  1 SB  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0]
[ 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0 SB SB  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0]
[ 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0 SB SB  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0]
[ 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0 SB  1 SB  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0]
[ 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0 SB  0  0  0  0  1 SB  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0]
[ 0  0  0  0  0  0  0  0  0  0 SB  0  0  0  0  0  0  0  0  0  0  0  0  1 SB  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0]
[ 0 SB  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  1 SB  0  0  0  0  0  0  0  0  0  0  0  0  0  0]
[ 0  0  0  0  0  0  0  0  0  0  0  0  0 SB  0  0  0  0  0  0  0  0  0  0  0  1 SB  0  0  0  0  0  0  0  0  0  0  0  0  0]
[ 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  1 SB  0  0  0  0  0  0  0  0  0  0  0  0]
[ 0  0  0  0  0  0  0  0  0  0  0 SB  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  1 SB  0  0  0  0  0  0  0  0  0  0  0]
[ 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0 SB SB  0  0  0  0  0  0  0  0  0  0]
[ 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0 SB  0  0  0  0  0  0  0  1 SB  0  0  0  0  0  0  0  0  0]
[ 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0 SB  0  0  0  0  0  0  0  0  0  0  0  0  0  1 SB  0  0  0  0  0  0  0  0]
[ 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0 SB  0  0  0  0  0  0  0  0  0  0  0  0  0  1 SB  0  0  0  0  0  0  0]
[ 0  0  0  0  0  0  0  0  0  0  0  0  0 SB  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  1 SB  0  0  0  0  0  0]
[ 0  0  0  0  0  0  0  0  0 SB  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  1 SB  0  0  0  0  0]
[ 0 SB  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  1 SB  0  0  0  0]
[ 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0 SB  0  0  0  0  0  0  0  0  1 SB  0  0  0]
[ 0  0  0  0  0  0  0  0  0  0 SB  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  1 SB  0  0]
[ 0  0  0  0  0  0  0 SB  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  1 SB  0]
```

Figure A.2: The Matrix A_S.

A.8 Construction of the Matrices of the SSSC

A digraph $\mathcal{G}(\Sigma_\Lambda)$ describing the structured linear system associated to the state equations (A.12), is the combination of a vertex set $\mathcal{V}$ and an edge set $\mathcal{E}$. The vertices represent the states and the input components of Σ_Λ while the edges describe the dynamic relations between these variables. One has $\mathcal{V} = \mathbf{X} \cup \{\mathbf{m}\}$ where $\mathbf{X}$ is the set of state vertices defined as $\mathbf{X} = \{\mathbf{x^1}, \ldots, \mathbf{x^n}\}$ and $\mathbf{m}$ is the input vertex. The edge set is $\mathcal{E} = \mathcal{E}_A \cup \mathcal{E}_B$, with $\mathcal{E}_A = \{(\mathbf{x^i}, \mathbf{x^j}) \,|A(i,j) \neq 0\}$ and $\mathcal{E}_B = \{(\mathbf{m}, \mathbf{x^i}) \,|B(i) \neq 0\}$. The entries of $A_{\rho(t)}$ correspond to the weights of the edges in the digraph. For convenience, we will denote by $\mathbf{v^j}$, $(j = 0, \ldots, n)$ a vertex of the digraph $\mathcal{G}(\Sigma_\Lambda)$ regardless of whether it is the input or a state vertex.

Given a triplet (n, r, n_a) with n the dimension of the state, r the delay and n_a the number of non-zero entries of the matrix A, the construction of the digraph $\mathcal{G}(\Sigma_\Lambda)$ related to the system Σ_Λ involves the following steps.

The system Σ_Λ is of dimension n and thus, the digraph $\mathcal{G}(\Sigma_\Lambda)$ involves $n+1$ vertices. The input is assigned to the vertex denoted by $\mathbf{v^0}$. The other n vertices are denoted by $\mathbf{v^1}, \ldots, \mathbf{v^n}$. Let $\mathbf{v^r}$ be the vertex that corresponds to the flat output $\mathbf{v^r}$.

Step 1: For, $i = 0, \ldots, n-1$, add the edges $(\mathbf{v^i}, \mathbf{v^{i+1}})$. There are r edges which connect $\mathbf{v^0}$ to $\mathbf{v^r}$. Hence, the delay of the automaton is r.

After Step 1, this line topology corresponds to quite trivial dynamical systems since it corresponds to state transition functions in the form of simple shifts. Let us recall that we aim at designing an automaton possibly involving state transition functions more general than T–functions. A shift is a special and trivial T–function. To this end, the following steps provide a way of adding edges $(\mathbf{v^i}, \mathbf{v^j})$ while guaranteeing flatness.

Step 2: Add the edges $(\mathbf{v^{r+i}}, \mathbf{v^{r+i+1}})$ for $i = 1, \ldots, n-r-1$. Step 2 allows vertex $\mathbf{v^j}$, $j = r+1, \ldots, n$ to have a predecessor. Indeed, if not so, the dynamics of the corresponding vertex $\mathbf{v^j}$ would reduce to $x^j_{k+1} = 0$ and would be clearly useless. The resulting path is a so-called main directed path and is depicted in Figure A.3.

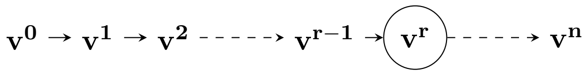


Figure A.3: Digraph obtained after completion of Step 1-2. The vertex $\mathbf{v^r}$ corresponds to the flat output

Step 3: Add the edges $(\mathbf{v^r}, \mathbf{v^i})$, $i = 1, \ldots, n$ that connect the vertex $\mathbf{v^r}$ to any other vertices of the graph (except the vertex $\mathbf{v^0}$ related to the input).

Step 4: For every vertex $\mathbf{v^i}$, $i = 1, \ldots, r-1$, add the directed edge $(\mathbf{v^i}, \mathbf{v^j})$ for $j = 1, \ldots, i$.

The graph obtained after Step 1-4 is depicted in Figure A.4.

Step 5: For every vertex $\mathbf{v_i}$, $i = r+1, \ldots, n$, add the directed edge $(\mathbf{v_i}, \mathbf{v_j})$ for $j = 1, \ldots, r$ and $j = i+2, \ldots, n$.

The resulting digraph after completion of Step 1-5 is depicted in Figure A.5.

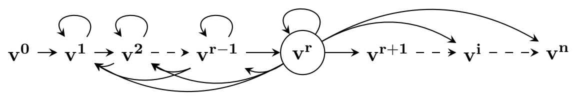


Figure A.4: Graph obtained after Step 1-4.

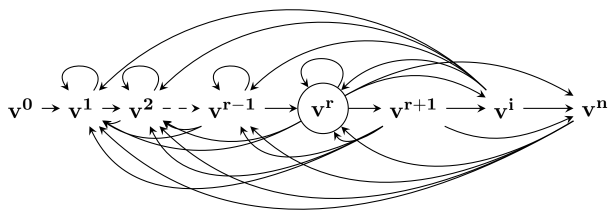


Figure A.5: Graph obtained after completion of Step 1-5.

To sum up, the digraph $\mathcal{G}(\Sigma_\Lambda)$ is parametrized by the triplet (n, r, n_a). The number of vertices of the digraph is equal to $n+1$. Indeed, there are n vertices assigned to the state components and one assigned to the input. The delay r is the number of edges in the main directed path. The integer n_a defines the desired number of edges in the digraph $\mathcal{G}(\Sigma_\Lambda)$. It must satisfy $n_a \leq n_M$, where n_M is the maximal number of edges resulting from the construction Step 1-5. A simple counting leads to:

$$n_M = \frac{n(n+1)}{2} + r. \tag{A.30}$$

During the construction, at each step, we can decide whether we actually add the edges or not. That introduces flexibility in the perspective of providing distinct graphs and thus, distinct SSSC as detailed in Subsection A.2.3.

Finally, the matrices I_A and I_B of the structural system Σ_Λ can be extracted from the adjacency matrix, denoted by $\mathcal{I}$, associated to the digraph $\mathcal{G}(\Sigma_\Lambda)$. Indeed, the adjacency matrix $\mathcal{I}$ associated to the digraph $\mathcal{G}(\Sigma_\Lambda)$ is the $(n+1)\times(n+1)$ matrix

$$\mathcal{I} = \left(\begin{array}{c:c} 0 & I_B^t \\ \hdashline 0 & \\ \vdots & I_A^t \\ 0 & \end{array}\right) \tag{A.31}$$

where I_A^t and I_B^t stands respectively for the transpose of the structured matrices I_A and I_B. The entries $\mathcal{I}_{ij}$ are defined as follows for $1 \leq i, j \leq n$

$$\mathcal{I}_{ij} = \begin{cases} 1 \text{ if there exists an edge from } \mathbf{v^j} \text{ to } \mathbf{v^i} \\ 0 \text{ otherwise.} \end{cases} \tag{A.32}$$

The adjacency matrix associated to $\mathcal{G}(\Sigma_\Lambda)$, obtained after completion of Step 1-5, is given by

$$
\begin{array}{c|cccccccccc}
 & \mathbf{v^0} & \mathbf{v^1} & \mathbf{v^2} & \mathbf{v^3} & \cdots & \mathbf{v^r} & \mathbf{v^{r+1}} & \cdots & \mathbf{v^{n-1}} & \mathbf{v^n} \\
\hline
\mathbf{v^0} & 0 & 1 & 0 & 0 & \cdots & 0 & \cdots & 0 & 0 & 0 \\
\mathbf{v^1} & 0 & 1 & 1 & 0 & \cdots & 0 & \cdots & 0 & 0 & 0 \\
\mathbf{v^2} & 0 & 1 & 1 & 1 & \cdots & 0 & \cdots & 0 & 0 & 0 \\
\mathbf{v^3} & 0 & 1 & 1 & 1 & \cdots & 0 & \cdots & 0 & 0 & 0 \\
\vdots & \vdots & \vdots & \vdots & \vdots & \ddots & 0 & 0 & 0 & 0 & 0 \\
\mathbf{v^r} & 0 & 1 & 1 & 1 & \cdots & 1 & 1 & 1 & 1 & 1 \\
\mathbf{v^{r+1}} & 0 & 1 & 1 & 1 & \cdots & 1 & 0 & 1 & 1 & 1 \\
\vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \ddots & \ddots & \vdots \\
\mathbf{v^{n-1}} & 0 & 1 & 1 & 1 & \cdots & 1 & 0 & 0 & 0 & 1 \\
\mathbf{v^n} & 0 & 1 & 1 & 1 & \cdots & 1 & 0 & 0 & 0 & 0
\end{array}
$$

The open source software Sagemath [CCC+13] has been used to elaborate the digraph $\mathcal{G}(\Sigma_\Lambda)$ corresponding to the triplet $(n = 40, r = 3, n_a = 120)$. The construction has been performed on an Intel CORE i7 CPU 2.26 GHz running Linux Ubuntu 14.04. All experiments ran single-threaded on the processors. It took 21 *ms* on the computer to obtain the digraph $\mathcal{G}(\Sigma_\Lambda)$.

Bibliography

[ABC+17] Ralph Ankele, Subhadeep Banik, Avik Chakraborti, Eik List, Florian Mendel, Siang Meng Sim, and Gaoli Wang. Related-key impossible-differential attack on reduced-round skinny. In Dieter Gollmann, Atsuko Miyaji, and Hiroaki Kikuchi, editors, *ACNS 17*, volume 10355 of *LNCS*, pages 208–228. Springer, Heidelberg, July 2017. doi:10.1007/978-3-319-61204-1_11.

[ABC+18] Alexandre Adomnicai, Thierry P. Berger, Christophe Clavier, Julien Francq, Paul Huynh, Virginie Lallemand, Kévin Le Gouguec, Marine Minier, Léo Reynaud, and Gaël Thomas. Lilliput-AE: a new lightweight tweakable block cipher for authenticated encryption with associated data. Submission to the NIST Lightweight Cryptography project. Available online https://csrc.nist.gov/CSRC/media/Projects/Lightweight-Cryptography/documents/round-1/spec-doc/LILLIPUT-AE-spec.pdf., 2018.

[ABD+12] Wim Aerts, Eli Biham, Dieter De Moitie, Elke De Mulder, Orr Dunkelman, Sebastiaan Indesteege, Nathan Keller, Bart Preneel, Guy A. E. Vandenbosch, and Ingrid Verbauwhede. A practical attack on KeeLoq. *Journal of Cryptology*, 25(1):136–157, January 2012. doi:10.1007/s00145-010-9091-9.

[ABI+18] Gianira Alfarano, Christof Beierle, Takanori Isobe, Stefan Kölbl, and Gregor Leander. Shiftrows alternatives for AES-like ciphers and optimal cell permutations for midori and skinny. *IACR Trans. Symm. Cryptol.*, 2018(2):20–47, 2018. doi:10.13154/tosc.v2018.i2.20-47.

[ABL+09] F. Arnault, T. P. Berger, C. Lauradoux, M. Minier, and B. Pousse. A new approach for fcsrs. In *Selected Areas in Cryptography - SAC 2009*, volume 5867 of *Lecture Notes in Computer Science*, pages 433–448. Springer, 2009.

[ABMP11] Francois Arnault, Thierry P. Berger, Marine Minier, and Benjamin Pousse. Revisiting LFSRs for Cryptographic Applications. *IEEE Trans. on Info. Theory*, 57(12):8095–8113, 2011.

[AC09] Martin Albrecht and Carlos Cid. Algebraic techniques in differential cryptanalysis. In Orr Dunkelman, editor, *FSE 2009*, volume 5665 of *LNCS*, pages 193–208. Springer, Heidelberg, February 2009. doi:10.1007/978-3-642-03317-9_12.

[ADK+14] Martin R. Albrecht, Benedikt Driessen, Elif Bilge Kavun, Gregor Leander, Christof Paar, and Tolga Yalçin. Block ciphers - focus on the linear layer (feat. PRIDE). In Juan A. Garay and Rosario Gennaro, editors, *CRYPTO 2014, Part I*, volume 8616 of *LNCS*, pages 57–76. Springer, Heidelberg, August 2014. doi:10.1007/978-3-662-44371-2_4.

[AES01] Advanced Encryption Standard (AES). National Institute of Standards and Technology (NIST), FIPS PUB 197, U.S. Department of Commerce, November 2001.

[AH16] Roberto Avanzi and Howard M. Heys, editors. *SAC 2016*, volume 10532 of *LNCS*. Springer, Heidelberg, August 2016.

[AIK$^+$01] Kazumaro Aoki, Tetsuya Ichikawa, Masayuki Kanda, Mitsuru Matsui, Shiho Moriai, Junko Nakajima, and Toshio Tokita. Camellia: A 128-bit block cipher suitable for multiple platforms - Design and analysis. In Douglas R. Stinson and Stafford E. Tavares, editors, *SAC 2000*, volume 2012 of *LNCS*, pages 39–56. Springer, Heidelberg, August 2001. doi:10.1007/3-540-44983-3_4.

[AK19] Ralph Ankele and Stefan Kölbl. Mind the gap - A closer look at the security of block ciphers against differential cryptanalysis. In Cid and Jacobson Jr: [CJ19], pages 163–190. doi:10.1007/978-3-030-10970-7_8.

[ALP$^+$19] Elena Andreeva, Virginie Lallemand, Antoon Purnal, Reza Reyhanitabar, Arnab Roy, and Damian Vizár. Forkcipher: A new primitive for authenticated encryption of very short messages. In Steven D. Galbraith and Shiho Moriai, editors, *ASIACRYPT 2019, Part II*, volume 11922 of *LNCS*, pages 153–182. Springer, Heidelberg, December 2019. doi:10.1007/978-3-030-34621-8_6.

[ANS14] Référentiel général de sécurité – Mécanismes cryptographiques. Technical report, 2014. Agence Nationale de la Sécurité des Systeèmes d'Information, https://www.ssi.gouv.fr/uploads/2014/11/RGS_v-2-0_B1.pdf.

[AP13] Nadhem J. AlFardan and Kenneth G. Paterson. Lucky thirteen: Breaking the TLS and DTLS record protocols. In *2013 IEEE Symposium on Security and Privacy*, pages 526–540. IEEE Computer Society Press, May 2013. doi:10.1109/SP.2013.42.

[AST$^+$17] Ahmed Abdelkhalek, Yu Sasaki, Yosuke Todo, Mohamed Tolba, and Amr M. Youssef. MILP modeling for (large) s-boxes to optimize probability of differential characteristics. *IACR Trans. Symm. Cryptol.*, 2017(4):99–129, 2017. doi:10.13154/tosc.v2017.i4.99-129.

[Bab95] S. Babbage. A Space/Time Trade-Off in Exhaustive Search Attacks on Stream Ciphers. In *European Convention on Security and Detection*, number 408. IEEE Conference Publication, 1995.

[Bar20] Elaine Barker. NIST SP 800-57. Recommendation for key management: Part 1—general (revised). Technical report, Gaithersburg, MD, USA, 2020. https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-57pt1r5.pdf.

[BBB$^+$19] Davide Bellizia, Francesco Berti, Olivier Bronchain, Gaëtan Cassiers, Sébastien Duval, Chun Guo, Gregor Leander, Gaëtan Leurent, Itamar Levi, Charles Momin, Olivier Pereira, Thomas Peters, François-Xavier Standaert, and Friedrich Wiemer. Spook: Sponge-based leakage-resilient authenticated encryption with a masked tweakable block cipher. Submission to the NIST Lightweight Cryptography project. Available online https://csrc.nist.gov/CSRC/media/Projects/lightweight-cryptography/documents/round-2/spec-doc-rnd2/Spook-spec-round2.pdf., 2019.

[BBB+20] Davide Bellizia, Francesco Berti, Olivier Bronchain, Gaëtan Cassiers, Sébastien Duval, Chun Guo, Gregor Leander, Gaëtan Leurent, Itamar Levi, Charles Momin, Olivier Pereira, Thomas Peters, François-Xavier Standaert, Balazs Udvarhelyi, and Friedrich Wiemer. Spook: Sponge-based leakage-resistant authenticated encryption with a masked tweakable block cipher. *IACR Trans. Symm. Cryptol.*, 2020(S1):295–349, 2020. doi:10.13154/tosc.v2020.iS1.295-349.

[BBC+08] Côme Berbain, Olivier Billet, Anne Canteaut, Nicolas T. Courtois, Henri Gilbert, Louis Goubin, Aline Gouget, Louis Granboulan, Cédric Lauradoux, Marine Minier, Thomas Pornin, and Hervé Sibert. Sosemanuk, a fast software-oriented stream cipher. 4986:98–118, 2008. URL: https://doi.org/10.1007/978-3-540-68351-3_9, doi:10.1007/978-3-540-68351-3_9.

[BBdS+19] Christof Beierle, Alex Biryukov, Luan Cardoso dos Santos, Johann Großschädl, Léo Perrin, Aleksei Udovenko, Vesselin Velichkov, and Qingju Wang. SCHWAEMM and ESCH: lightweight authenticated encryption and hashing using the SPARKLE permutation family. Submission to the 2nd round of the NIST lightweight process., 2019.

[BBI+15] Subhadeep Banik, Andrey Bogdanov, Takanori Isobe, Kyoji Shibutani, Harunaga Hiwatari, Toru Akishita, and Francesco Regazzoni. Midori: A block cipher for low energy. In Tetsu Iwata and Jung Hee Cheon, editors, *ASIACRYPT 2015, Part II*, volume 9453 of *LNCS*, pages 411–436. Springer, Heidelberg, November / December 2015. doi:10.1007/978-3-662-48800-3_17.

[BBK+13] Begül Bilgin, Andrey Bogdanov, Miroslav Knežević, Florian Mendel, and Qingju Wang. Fides: Lightweight authenticated cipher with side-channel resistance for constrained hardware. In Bertoni and Coron [BC13], pages 142–158. doi:10.1007/978-3-642-40349-1_9.

[BBL13] Céline Blondeau, Andrey Bogdanov, and Gregor Leander. Bounds in shallows and in miseries. In Canetti and Garay [CG13], pages 204–221. doi:10.1007/978-3-642-40041-4_12.

[BBP+19] Subhadeep Banik, Andrey Bogdanov, Thomas Peyrin, Yu Sasaki, Siang, Meng Sim, Elmar Tischhauser, and Yosuke Todo. SUNDAE-GIFT. Submission to the NIST Lightweight Cryptography project. Available online https://csrc.nist.gov/CSRC/media/Projects/lightweight-cryptography/documents/round-2/spec-doc-rnd2/SUNDAE-GIFT-spec-round2.pdf., 2019.

[BBR16] Subhadeep Banik, Andrey Bogdanov, and Francesco Regazzoni. Atomic-AES v2.0. Cryptology ePrint Archive, Report 2016/1005, 2016. http://eprint.iacr.org/2016/1005.

[BBS99] Eli Biham, Alex Biryukov, and Adi Shamir. Cryptanalysis of Skipjack reduced to 31 rounds using impossible differentials. In Jacques Stern, editor, *EUROCRYPT'99*, volume 1592 of *LNCS*, pages 12–23. Springer, Heidelberg, May 1999. doi:10.1007/3-540-48910-X_2.

[BC13] Guido Bertoni and Jean-Sébastien Coron, editors. *CHES 2013*, volume 8086 of *LNCS*. Springer, Heidelberg, August 2013.

[BC16] Christina Boura and Anne Canteaut. Another view of the division property. In Robshaw and Katz [RK16a], pages 654–682. doi:10.1007/978-3-662-53018-4_24.

[BC18] Christina Boura and Anne Canteaut. On the boomerang uniformity of cryptographic sboxes. *IACR Trans. Symm. Cryptol.*, 2018(3):290–310, 2018. doi:10.13154/tosc.v2018.i3.290-310.

[BCD+99] Carolynn Burwick, Don Coppersmith, Edward D'Avignon, Rosario Gennaro, Shai Halevi, Charanjit Jutla, Stephen M. Matyas Jr, Luke O'Connor, Mohammad Peyravian, Jr. Luke, O'connor Mohammad Peyravian, David Stafford, and Nevenko Zunic. Mars - a candidate cipher for aes. *NIST AES Proposal*, 1999.

[BCD11] Christina Boura, Anne Canteaut, and Christophe De Cannière. Higher-order differential properties of Keccak and Luffa. In Joux [Jou11], pages 252–269. doi:10.1007/978-3-642-21702-9_15.

[BCG+12] Julia Borghoff, Anne Canteaut, Tim Güneysu, Elif Bilge Kavun, Miroslav Knežević, Lars R. Knudsen, Gregor Leander, Ventzislav Nikov, Christof Paar, Christian Rechberger, Peter Rombouts, Søren S. Thomsen, and Tolga Yalçin. PRINCE - A low-latency block cipher for pervasive computing applications - extended abstract. In Xiaoyun Wang and Kazue Sako, editors, *ASIACRYPT 2012*, volume 7658 of *LNCS*, pages 208–225. Springer, Heidelberg, December 2012. doi:10.1007/978-3-642-34961-4_14.

[BCI+19] Subhadeep Banik, Avik Chakraborti, Tetsu Iwata, Kazuhiko Minematsu, Mridul Nandi, Thomas Peyrin, Yu Sasaki, Siang Meng Sim, and Yosuke Todo. GIFT-COFB. Submission to the NIST Lightweight Cryptography project. Available online https://csrc.nist.gov/CSRC/media/Projects/lightweight-cryptography/documents/round-2/spec-doc-rnd2/gift-cofb-spec-round2.pdf., 2019.

[BCKL17] Christina Boura, Anne Canteaut, Lars R. Knudsen, and Gregor Leander. Reflection ciphers. *Des. Codes Cryptogr.*, 82(1-2):3–25, 2017. URL: https://doi.org/10.1007/s10623-015-0143-x, doi:10.1007/s10623-015-0143-x.

[BCLR17] Christof Beierle, Anne Canteaut, Gregor Leander, and Yann Rotella. Proving resistance against invariant attacks: How to choose the round constants. In Jonathan Katz and Hovav Shacham, editors, *CRYPTO 2017, Part II*, volume 10402 of *LNCS*, pages 647–678. Springer, Heidelberg, August 2017. doi:10.1007/978-3-319-63715-0_22.

[BD08] Steve Babbage and Matthew Dodd. The MICKEY stream ciphers. In Matthew J. B. Robshaw and Olivier Billet, editors, *New Stream Cipher Designs - The eSTREAM Finalists*, volume 4986 of *Lecture Notes in Computer Science*, pages 191–209. Springer, 2008. URL: https://doi.org/10.1007/978-3-540-68351-3_15, doi:10.1007/978-3-540-68351-3_15.

[BDF11] Charles Bouillaguet, Patrick Derbez, and Pierre-Alain Fouque. Automatic search of attacks on round-reduced AES and applications. In Rogaway [Rog11], pages 169–187. doi:10.1007/978-3-642-22792-9_10.

[BDK01] Eli Biham, Orr Dunkelman, and Nathan Keller. The rectangle attack - rectangling the Serpent. In Pfitzmann [Pfi01], pages 340–357. doi:10.1007/3-540-44987-6_21.

[BDK05] Eli Biham, Orr Dunkelman, and Nathan Keller. Related-key boomerang and rectangle attacks. In Ronald Cramer, editor, *EUROCRYPT 2005*, volume 3494 of *LNCS*, pages 507–525. Springer, Heidelberg, May 2005. doi:10.1007/11426639_30.

[BDKW19] Achiya Bar-On, Orr Dunkelman, Nathan Keller, and Ariel Weizman. DLCT: A new tool for differential-linear cryptanalysis. In Yuval Ishai and Vincent Rijmen, editors, *EUROCRYPT 2019, Part I*, volume 11476 of *LNCS*, pages 313–342. Springer, Heidelberg, May 2019. doi:10.1007/978-3-030-17653-2_11.

[BDP$^+$16] Guido Bertoni, Joan Daemen, Michaël Peeters, Gilles Van Assche, and Ronny Van Keer. CAESAR submission: Ketje v2. Submission to the CAESAR Competition, 2016.

[BDQ04] Alex Biryukov, Christophe De Cannière, and Michaël Quisquater. On multiple linear approximations. In Matthew Franklin, editor, *CRYPTO 2004*, volume 3152 of *LNCS*, pages 1–22. Springer, Heidelberg, August 2004. doi:10.1007/978-3-540-28628-8_1.

[Bee19] Daniel Beer. mspdebug. https://github.com/dlbeer/mspdebug, 2019. Accessed: 2019-03-07.

[Ber08] Daniel J. Bernstein. The salsa20 family of stream ciphers. In Matthew J. B. Robshaw and Olivier Billet, editors, *New Stream Cipher Designs - The eSTREAM Finalists*, volume 4986 of *Lecture Notes in Computer Science*, pages 84–97. Springer, 2008. URL: https://doi.org/10.1007/978-3-540-68351-3_8, doi:10.1007/978-3-540-68351-3_8.

[Ber16] Daniel J. Bernstein. CAESAR use cases. https://groups.google.com/forum/#!msg/crypto-competitions/DLv193SPSDc/4CeHPvIoBgAJ, 2016.

[BFMT16] Thierry P. Berger, Julien Francq, Marine Minier, and Gaël Thomas. Extended generalized feistel networks using matrix representation to propose a new lightweight block cipher: Lilliput. *IEEE Trans. Computers*, 65(7):2074–2089, 2016. URL: https://doi.org/10.1109/TC.2015.2468218, doi:10.1109/TC.2015.2468218.

[BG10] Céline Blondeau and Benoît Gérard. Links between theoretical and effective differential probabilities: Experiments on PRESENT. Cryptology ePrint Archive, Report 2010/261, 2010. http://eprint.iacr.org/2010/261.

[BG11] Céline Blondeau and Benoît Gérard. Multiple differential cryptanalysis: Theory and practice. In Joux [Jou11], pages 35–54. doi:10.1007/978-3-642-21702-9_3.

[BGG$^+$16] Erik Boss, Vincent Grosso, Tim Güneysu, Gregor Leander, Amir Moradi, and Tobias Schneider. Strong 8-bit sboxes with efficient masking in hardware. In Gierlichs and Poschmann [GP16], pages 171–193. doi:10.1007/978-3-662-53140-2_9.

[BGG$^+$20] Fabrice Boudot, Pierrick Gaudry, Aurore Guillevic, Nadia Heninger, Emmanuel Thomé, and Paul Zimmermann. Comparing the difficulty of factorization and discrete logarithm: A 240-digit experiment. In Daniele Micciancio and Thomas Ristenpart, editors, *CRYPTO 2020, Part II*, volume 12171 of *LNCS*, pages 62–91. Springer, Heidelberg, August 2020. doi:10.1007/978-3-030-56880-1_3.

[BGJT14] Razvan Barbulescu, Pierrick Gaudry, Antoine Joux, and Emmanuel Thomé. A heuristic quasi-polynomial algorithm for discrete logarithm in finite fields of small characteristic. In Nguyen and Oswald [NO14], pages 1–16. doi:10.1007/978-3-642-55220-5_1.

[BGW$^+$14] Andrey Bogdanov, Huizheng Geng, Meiqin Wang, Long Wen, and Baudoin Collard. Zero-correlation linear cryptanalysis with FFT and improved attacks on ISO standards Camellia and CLEFIA. In Lange et al. [LLL14], pages 306–323. doi:10.1007/978-3-662-43414-7_16.

[BHL$^+$20] Hamid Boukerrou, Paul Huynh, Virginie Lallemand, Bimal Mandal, and Marine Minier. On the Feistel counterpart of the boomerang connectivity table (long paper). *IACR Trans. Symm. Cryptol.*, 2020(1):331–362, 2020. doi:10.13154/tosc.v2020.i1.331-362.

[Bie14] Armin Biere. Yet another local search solver and lingeling and friends entering the sat competition 2014. *Sat competition*, 2014(2):65, 2014.

[Bih94] Eli Biham. New types of cryptanalytic attacks using related keys (extended abstract). In Helleseth [Hel94], pages 398–409. doi:10.1007/3-540-48285-7_34.

[Bih97a] Eli Biham. Cryptanalysis of Ladder-DES. In FSE 1997 [Bih97b], pages 134–138. doi:10.1007/BFb0052341.

[Bih97b] Eli Biham, editor. *FSE'97*, volume 1267 of *LNCS*. Springer, Heidelberg, January 1997.

[BJK$^+$16] Christof Beierle, Jérémy Jean, Stefan Kölbl, Gregor Leander, Amir Moradi, Thomas Peyrin, Yu Sasaki, Pascal Sasdrich, and Siang Meng Sim. The SKINNY family of block ciphers and its low-latency variant MANTIS. In Robshaw and Katz [RK16b], pages 123–153. doi:10.1007/978-3-662-53008-5_5.

[BJK$^+$19] Christof Beierle, Jérémy Jean, Stefan Kölbl, Gregor Leander, Amir Moradi, Thomas Peyrin, Yu Sasaki, Pascal Sasdrich, and Siang Meng Sim. SKINNY-AEAD and SKINNY-Hash. Submission to the NIST Lightweight Cryptography project. Available online https://csrc.nist.gov/CSRC/media/Projects/lightweight-cryptography/documents/round-2/spec-doc-rnd2/SKINNY-spec-round2.pdf., 2019.

[BJM$^+$14] Lejla Batina, Domagoj Jakobovic, Nele Mentens, Stjepan Picek, Antonio De La Piedra, and Dominik Sisejkovic. S-box pipelining using genetic algorithms for high-throughput AES implementations: How fast can we go? In Willi Meier and Debdeep Mukhopadhyay, editors, *INDOCRYPT 2014*, volume 8885 of *LNCS*, pages 322–337. Springer, Heidelberg, December 2014. doi:10.1007/978-3-319-13039-2_19.

[BK09] Alex Biryukov and Dmitry Khovratovich. Related-key cryptanalysis of the full AES-192 and AES-256. In Mitsuru Matsui, editor, *ASIACRYPT 2009*, volume 5912 of *LNCS*, pages 1–18. Springer, Heidelberg, December 2009. doi:10.1007/978-3-642-10366-7_1.

[BKL$^+$07] Andrey Bogdanov, Lars R. Knudsen, Gregor Leander, Christof Paar, Axel Poschmann, Matthew J. B. Robshaw, Yannick Seurin, and C. Vikkelsoe. PRESENT: An ultra-lightweight block cipher. In Pascal Paillier and Ingrid Verbauwhede, editors, *CHES 2007*, volume 4727 of *LNCS*, pages 450–466. Springer, Heidelberg, September 2007. doi:10.1007/978-3-540-74735-2_31.

[BKLT11] Julia Borghoff, Lars R. Knudsen, Gregor Leander, and Søren S. Thomsen. Cryptanalysis of PRESENT-like ciphers with secret S-boxes. In Joux [Jou11], pages 270–289. doi:10.1007/978-3-642-21702-9_16.

[BKN02] Mihir Bellare, Tadayoshi Kohno, and Chanathip Namprempre. Breaking and provably repairing the SSH authenticated encryption scheme: A case study of the Encode-then-Encrypt-and-MAC paradigm. Cryptology ePrint Archive, Report 2002/078, 2002. http://eprint.iacr.org/2002/078.

[BKS09] Julia Borghoff, Lars R. Knudsen, and Mathias Stolpe. Bivium as a mixed-integer linear programming problem. In Matthew G. Parker, editor, *12th IMA International Conference on Cryptography and Coding*, volume 5921 of *LNCS*, pages 133–152. Springer, Heidelberg, December 2009.

[BM15] T. P. Berger and M. Minier. Some results using the matrix methods on impossible, integral and zero-correlation distinguishers for feistel-like ciphers. In *Progress in Cryptology - INDOCRYPT 2015*, volume 9462 of *Lecture Notes in Computer Science*, pages 180–197. Springer, 2015.

[BMNS14] Christina Boura, Marine Minier, María Naya-Plasencia, and Valentin Suder. Improved impossible differential attacks against round-reduced LBlock. Cryptology ePrint Archive, Report 2014/279, 2014. http://eprint.iacr.org/2014/279.

[BMP09] Thierry P. Berger, Marine Minier, and Benjamin Pousse. Software oriented stream ciphers based upon FCSRs in diversified mode. In Bimal K. Roy and Nicolas Sendrier, editors, *INDOCRYPT 2009*, volume 5922 of *LNCS*, pages 119–135. Springer, Heidelberg, December 2009.

[BMT14] Thierry P. Berger, Marine Minier, and Gaël Thomas. Extended generalized Feistel networks using matrix representation. In Lange et al. [LLL14], pages 289–305. doi:10.1007/978-3-662-43414-7_15.

[BN00] Mihir Bellare and Chanathip Namprempre. Authenticated encryption: Relations among notions and analysis of the generic composition paradigm. In Tatsuaki Okamoto, editor, *ASIACRYPT 2000*, volume 1976 of *LNCS*, pages 531–545. Springer, Heidelberg, December 2000. doi:10.1007/3-540-44448-3_41.

[BN10] Alex Biryukov and Ivica Nikolic. Automatic search for related-key differential characteristics in byte-oriented block ciphers: Application to AES, Camellia, Khazad and others. In Henri Gilbert, editor, *EUROCRYPT 2010*, volume 6110 of *LNCS*, pages 322–344. Springer, Heidelberg, May / June 2010. doi:10.1007/978-3-642-13190-5_17.

[BN14] Céline Blondeau and Kaisa Nyberg. Links between truncated differential and multidimensional linear properties of block ciphers and underlying attack complexities. In Nguyen and Oswald [NO14], pages 165–182. doi:10.1007/978-3-642-55220-5_10.

[BNN+12] Begül Bilgin, Svetla Nikova, Ventzislav Nikov, Vincent Rijmen, and Georg Stütz. Threshold implementations of all 3×3 and 4×4 S-boxes. In Emmanuel Prouff and Patrick Schaumont, editors, *CHES 2012*, volume 7428 of *LNCS*, pages 76–91. Springer, Heidelberg, September 2012. doi:10.1007/978-3-642-33027-8_5.

[BNS14] Christina Boura, María Naya-Plasencia, and Valentin Suder. Scrutinizing and improving impossible differential attacks: Applications to CLEFIA, Camellia, LBlock and Simon. In Sarkar and Iwata [SI14], pages 179–199. doi:10.1007/978-3-662-45611-8_10.

[BP09] Joan Boyar and Rene Peralta. New logic minimization techniques with applications to cryptology. Cryptology ePrint Archive, Report 2009/191, 2009. http://eprint.iacr.org/2009/191.

[BP15] Alex Biryukov and Léo Perrin. Lightweight cryptography lounge. http://cryptolux.org/index.php/Lightweight_Cryptography, 2015.

[BP17] Alex Biryukov and Leo Perrin. State of the art in lightweight symmetric cryptography. Cryptology ePrint Archive, Report 2017/511, 2017. http://eprint.iacr.org/2017/511.

[BPPS17] Francesco Berti, Olivier Pereira, Thomas Peters, and François-Xavier Standaert. On leakage-resilient authenticated encryption with decryption leakages. *IACR Trans. Symm. Cryptol.*, 2017(3):271–293, 2017. doi:10.13154/tosc.v2017.i3.271-293.

[BPT19] Christina Boura, Léo Perrin, and Shizhu Tian. Boomerang uniformity of popular s-box constructions. In *Proceedings of The Eleventh International Workshop on Coding and Cryptograph (WCC)*, 2019.

[BPW15] Céline Blondeau, Thomas Peyrin, and Lei Wang. Known-key distinguisher on full PRESENT. In Gennaro and Robshaw [GR15], pages 455–474. doi:10.1007/978-3-662-47989-6_22.

[BR11] Andrey Bogdanov and Vincent Rijmen. Linear hulls with correlation zero and linear cryptanalysis of block ciphers. Cryptology ePrint Archive, Report 2011/123, 2011. http://eprint.iacr.org/2011/123.

[BRAN00] Paulo S. L. M. Barreto, Vincent Rijmen, Scopus Tecnologia S. A, and Cryptomathic Nv. The Whirlpool Hashing Function. In *First open NESSIE Workshop*, 2000.

[BRW04] Mihir Bellare, Phillip Rogaway, and David Wagner. The EAX mode of operation. In Roy and Meier [RM04], pages 389–407. doi:10.1007/978-3-540-25937-4_25.

[BS91a] Eli Biham and Adi Shamir. Differential cryptanalysis of DES-like cryptosystems. In Menezes and Vanstone [MV91], pages 2–21. doi:10.1007/3-540-38424-3_1.

[BS91b] Eli Biham and Adi Shamir. Differential cryptanalysis of Feal and N-hash. In Davies [Dav91], pages 1–16. doi:10.1007/3-540-46416-6_1.

[BS93] Eli Biham and Adi Shamir. Differential cryptanalysis of the full 16-round DES. In Ernest F. Brickell, editor, *CRYPTO'92*, volume 740 of *LNCS*, pages 487–496. Springer, Heidelberg, August 1993. doi:10.1007/3-540-48071-4_34.

[BS00] A. Biryukov and A. Shamir. Cryptanalytic time/memory/data tradeoffs for stream ciphers. In *Advances in Cryptology - ASIACRYPT 2000*, volume 1976 of *Lecture Notes in Computer Science*, pages 1–13. Springer, 2000.

[BSK14] Frederick J. Bruwer, Willem Smit, and Gideon J Kuhn. Microchips and remote control devices comprising same. US Patent 5517187, 1996-05-14.

[BSS$^+$13] Ray Beaulieu, Douglas Shors, Jason Smith, Stefan Treatman-Clark, Bryan Weeks, and Louis Wingers. The SIMON and SPECK families of lightweight block ciphers. Cryptology ePrint Archive, Report 2013/404, 2013. http://eprint.iacr.org/2013/404.

[BTV18] Andrey Bogdanov, Elmar Tischhauser, and Philip S. Vejre. Multivariate profiling of hulls for linear cryptanalysis. *IACR Trans. Symm. Cryptol.*, 2018(1):101–125, 2018. doi:10.13154/tosc.v2018.i1.101-125.

[BV14] Alex Biryukov and Vesselin Velichkov. Automatic search for differential trails in ARX ciphers. In Josh Benaloh, editor, *CT-RSA 2014*, volume 8366 of *LNCS*, pages 227–250. Springer, Heidelberg, February 2014. doi:10.1007/978-3-319-04852-9_12.

[BVP$^+$03] Martin Boesgaard, Mette Vesterager, Thomas Pedersen, Jesper Christiansen, and Ove Scavenius. Rabbit: A new high-performance stream cipher. In Johansson [Joh03], pages 307–329. doi:10.1007/978-3-540-39887-5_23.

[BW99] Alex Biryukov and David Wagner. Slide attacks. In Knudsen [Knu99], pages 245–259. doi:10.1007/3-540-48519-8_18.

[CAE] Crypto competitions: CAESAR submissions. https://competitions.cr.yp.to/caesar-submissions.html. Accessed: 2019-07-23.

[Can05] D. Canright. A very compact S-box for AES. In Josyula R. Rao and Berk Sunar, editors, *CHES 2005*, volume 3659 of *LNCS*, pages 441–455. Springer, Heidelberg, August / September 2005. doi:10.1007/11545262_32.

[Can07] C. De Cannière. *Analysis and Design of Symmetric Encryption Algorithms.* PhD thesis, Katholieke Universiteit Leuven, 2007.

[Car10] Claude Carlet. Vectorial boolean functions for cryptography. *Boolean models and methods in mathematics, computer science, and engineering*, 134:398–469, 2010.

[CBW08] Nicolas Courtois, Gregory V. Bard, and David Wagner. Algebraic and slide attacks on KeeLoq. In Nyberg [Nyb08], pages 97–115. doi:10.1007/978-3-540-71039-4_6.

[CCC$^+$13] A. Casamayou, N. Cohen, G. Connan, T. Dumont, L. Fousse, F. Maltey, M. Meulien, M. Mezzarobba, C. Pernet, N. Thiéry, et al. *Calcul mathématique avec Sage*. available online: https://hal.inria.fr/inria-00540485v2/document, 2013.

[CCZ98] Claude Carlet, Pascale Charpin, and Victor A. Zinoviev. Codes, bent functions and permutations suitable for des-like cryptosystems. *Des. Codes Cryptogr.*, 15(2):125–156, 1998. URL: https://doi.org/10.1023/A:1008344232130, doi:10.1023/A:1008344232130.

[CDJ+19] Avik Chakraborti, Nilanjan Datta, Ashwin Jha, Cuauhtemoc Mancilias Lopez, Mridul Nandi, and Yu Sasaki. LOTUS-AEAD and LOCUS-AEAD. Submission to the NIST Lightweight Cryptography project. Available online https://csrc.nist.gov/CSRC/media/Projects/lightweight-cryptography/documents/round-2/spec-doc-rnd2/lotus-locus-spec-round2.pdf., 2019.

[CDJN19] Avik Chakraborti, Nilanjan Datta, Ashwin Jha, and Mridul Nandi. HyENA. Submission to the NIST Lightweight Cryptography project. Available online https://csrc.nist.gov/CSRC/media/Projects/lightweight-cryptography/documents/round-2/spec-doc-rnd2/hyena-spec-round2.pdf., 2019.

[CDL16] Anne Canteaut, Sébastien Duval, and Gaëtan Leurent. Construction of lightweight S-boxes using Feistel and MISTY structures. In Dunkelman and Keliher [DK16], pages 373–393. doi:10.1007/978-3-319-31301-6_22.

[CG13] Ran Canetti and Juan A. Garay, editors. *CRYPTO 2013, Part I*, volume 8042 of *LNCS*. Springer, Heidelberg, August 2013.

[CGT19] Victor Cauchois, Clément Gomez, and Gaël Thomas. General diffusion analysis: How to find optimal permutations for generalized type-ii feistel schemes. *IACR Trans. Symmetric Cryptol.*, 2019(1):264–301, 2019. URL: https://doi.org/10.13154/tosc.v2019.i1.264-301, doi:10.13154/tosc.v2019.i1.264-301.

[CHN09] Joo Yeon Cho, Miia Hermelin, and Kaisa Nyberg. A new technique for multidimensional linear cryptanalysis with applications on reduced round Serpent. In Pil Joong Lee and Jung Hee Cheon, editors, *ICISC 08*, volume 5461 of *LNCS*, pages 383–398. Springer, Heidelberg, December 2009.

[CHP+17] Carlos Cid, Tao Huang, Thomas Peyrin, Yu Sasaki, and Ling Song. A security analysis of Deoxys and its internal tweakable block ciphers. *IACR Trans. Symm. Cryptol.*, 2017(3):73–107, 2017. doi:10.13154/tosc.v2017.i3.73-107.

[CHP+18] Carlos Cid, Tao Huang, Thomas Peyrin, Yu Sasaki, and Ling Song. Boomerang connectivity table: A new cryptanalysis tool. In Jesper Buus Nielsen and Vincent Rijmen, editors, *EUROCRYPT 2018, Part II*, volume 10821 of *LNCS*, pages 683–714. Springer, Heidelberg, April / May 2018. doi:10.1007/978-3-319-78375-8_22.

[CJ19] Carlos Cid and Michael J. Jacobson Jr:, editors. *SAC 2018*, volume 11349 of *LNCS*. Springer, Heidelberg, August 2019.

[CKPS00] N. Courtois, A. Klimov, J. Patarin, and A. Shamir. Efficient algorithms for solving overdefined systems of multivariate polynomial equations. In *Advances in Cryptology - EUROCRYPT 2000*, volume 1807 of *Lecture Notes in Computer Science*, pages 392–407. Springer, 2000.

[CLN+17] Anne Canteaut, Eran Lambooij, Samuel Neves, Shahram Rasoolzadeh, Yu Sasaki, and Marc Stevens. Refined probability of differential characteristics including dependency between multiple rounds. *IACR Trans. Symm. Cryptol.*, 2017(2):203–227, 2017. doi:10.13154/tosc.v2017.i2.203-227.

[CM03] Nicolas Courtois and Willi Meier. Algebraic attacks on stream ciphers with linear feedback. In Eli Biham, editor, *EUROCRYPT 2003*, volume 2656 of *LNCS*, pages 345–359. Springer, Heidelberg, May 2003. doi:10.1007/3-540-39200-9_21.

[CM13] Jiageng Chen and Atsuko Miyaji. Differential Cryptanalysis and Boomerang Cryptanalysis of LBlock. In Alfredo Cuzzocrea, Christian Kittl, Dimitris E. Simos, Edgar R. Weippl, and Lida Xu, editors, *Security Engineering and Intelligence Informatics - CD-ARES 2013 Workshops: MoCrySEn and SeCIHD*, volume 8128 of *LNCS*, pages 1–15. Springer, 2013. URL: https://doi.org/10.1007/978-3-642-40588-4_1, doi:10.1007/978-3-642-40588-4_1.

[CP02] Nicolas Courtois and Josef Pieprzyk. Cryptanalysis of block ciphers with overdefined systems of equations. In Yuliang Zheng, editor, *ASIACRYPT 2002*, volume 2501 of *LNCS*, pages 267–287. Springer, Heidelberg, December 2002. doi:10.1007/3-540-36178-2_17.

[CP08] Christophe De Cannière and Bart Preneel. Trivium. In *New Stream Cipher Designs - The eSTREAM Finalists*, volume 4986 of *Lecture Notes in Computer Science*, pages 244–266. Springer, 2008.

[CR15] Carlos Cid and Christian Rechberger, editors. *FSE 2014*, volume 8540 of *LNCS*. Springer, Heidelberg, March 2015.

[CR19] Christophe Clavier and Léo Reynaud. Systematic and random searches for compact 4-bit and 8-bit cryptographic S-boxes. Cryptology ePrint Archive, Report 2019/1379, 2019. https://eprint.iacr.org/2019/1379.

[CS09] Baudoin Collard and François-Xavier Standaert. A statistical saturation attack against the block cipher PRESENT. In Marc Fischlin, editor, *CT-RSA 2009*, volume 5473 of *LNCS*, pages 195–210. Springer, Heidelberg, April 2009. doi:10.1007/978-3-642-00862-7_13.

[CSSH19] Qiu Chen, Danping Shi, Siwei Sun, and Lei Hu. Automatic demirci-selçuk meet-in-the-middle attack on SKINNY with key-bridging. In Jianying Zhou, Xiapu Luo, Qingni Shen, and Zhen Xu, editors, *ICICS 19*, volume 11999 of *LNCS*, pages 233–247. Springer, Heidelberg, December 2019. doi:10.1007/978-3-030-41579-2_14.

[CT16] Jung Hee Cheon and Tsuyoshi Takagi, editors. *ASIACRYPT 2016, Part I*, volume 10031 of *LNCS*. Springer, Heidelberg, December 2016.

[CY09] Jiali Choy and Huihui Yap. Impossible boomerang attack for block cipher structures. In Tsuyoshi Takagi and Masahiro Mambo, editors, *IWSEC 09*, volume 5824 of *LNCS*, pages 22–37. Springer, Heidelberg, October 2009.

[Dav91] Donald W. Davies, editor. *EUROCRYPT'91*, volume 547 of *LNCS*. Springer, Heidelberg, April 1991.

[DBG$^+$15] Dumitru-Daniel Dinu, Alex Biryukov, Johann Groszschaedl, Dmitry Khovratovich, Yann Le Corre, and Léo Paul Perrin. FELICS - Fair Evaluation of LIghtweight Cryptographic Systems. https://csrc.nist.gov/csrc/media/events/lightweight-cryptography-workshop-2015/documents/papers/session7-dinu-paper.pdf, 2015.

[DCK$^+$15] Daniel Dinu, Yann Le Corre, Dmitry Khovratovich, Léo Perrin, Johann Großschädl, and Alex Biryukov. Triathlon of lightweight block ciphers for the internet of things. Cryptology ePrint Archive, Report 2015/209, 2015. http://eprint.iacr.org/2015/209.

[DCK+19] Daniel Dinu, Yann Le Corre, Dmitry Khovratovich, Léo Perrin, Johann Großschädl, and Alex Biryukov. Triathlon of lightweight block ciphers for the internet of things. *Journal of Cryptographic Engineering*, 9(3):283–302, September 2019. doi:10.1007/s13389-018-0193-x.

[De 06] Christophe De Cannière. Trivium: A stream cipher construction inspired by block cipher design principles. In Sokratis K. Katsikas, Javier Lopez, Michael Backes, Stefanos Gritzalis, and Bart Preneel, editors, *ISC 2006*, volume 4176 of *LNCS*, pages 171–186. Springer, Heidelberg, August / September 2006.

[DEM+17] Christoph Dobraunig, Maria Eichlseder, Stefan Mangard, Florian Mendel, and Thomas Unterluggauer. ISAP – towards side-channel secure authenticated encryption. *IACR Trans. Symm. Cryptol.*, 2017(1):80–105, 2017. doi:10.13154/tosc.v2017.i1.80-105.

[DEMS16] Christoph Dobraunig, Maria Eichlseder, Florian Mendel, and Martin Schläffer. Ascon v1.2. Submission to the CAESAR competition: https://competitions.cr.yp.to/round3/asconv12.pdf, 2016. URL: https://ascon.iaik.tugraz.at.

[DES77] Data encryption standard. National Bureau of Standards, NBS FIPS PUB 46, U.S. Department of Commerce, January 1977.

[DF16] Patrick Derbez and Pierre-Alain Fouque. Automatic search of meet-in-the-middle and impossible differential attacks. In Robshaw and Katz [RK16b], pages 157–184. doi:10.1007/978-3-662-53008-5_6.

[DFL19] Patrick Derbez, Pierre-Alain Fouque, and Baptiste Lambin. Linearly equivalent S-boxes and the division property. Cryptology ePrint Archive, Report 2019/097, 2019. https://eprint.iacr.org/2019/097.

[DFLM19] Patrick Derbez, Pierre-Alain Fouque, Baptiste Lambin, and Victor Mollimard. Efficient search for optimal diffusion layers of generalized feistel networks. *IACR Trans. Symmetric Cryptol.*, 2019(2):218–240, 2019. URL: https://doi.org/10.13154/tosc.v2019.i2.218-240, doi:10.13154/tosc.v2019.i2.218-240.

[DGM17a] B. Dravie, P. Guillot, and G. Millérioux. Design of self-synchronizing stream ciphers: A new control-theoretical paradigm. In *IFAC World Congress, (IFAC 2017)*, Toulouse, France, July 2017.

[DGM17b] B. Dravie, P. Guillot, and G. Millérioux. Security proof of the canonical form of self-synchronizing stream ciphers. *Des. Codes Cryptography*, 82(1-2):377–388, 2017.

[DGM+19] Nilanjan Datta, Ashrujit Ghoshal, Debdeep Mukhopadhyay, Sikhar Patranabis, Stjepan Picek, and Rajat Sadhukhan. TRIFLE. Submission to the NIST Lightweight Cryptography project. Available online https://csrc.nist.gov/CSRC/media/Projects/Lightweight-Cryptography/documents/round-1/spec-doc/trifle-spec.pdf., 2019.

[DGV92] J. Daemen, R. Govaerts, and J. Vandewalle. On the design of high speed self-synchronizing stream ciphers. In *Proc. of the ICCS/ISITA'92 conference*, volume 1, pages 279–283, Singapore, November 1992.

[DH77] W. Diffie and M. E. Hellman. Special feature exhaustive cryptanalysis of the nbs data encryption standard. *Computer*, 10(6):74–84, June 1977. URL: https://doi.org/10.1109/C-M.1977.217750, doi:10.1109/C-M.1977.217750.

[DHL$^+$16] Jordan Demeulenaere, Renaud Hartert, Christophe Lecoutre, Guillaume Perez, Laurent Perron, Jean-Charles Régin, and Pierre Schaus. Compact-table: Efficiently filtering table constraints with reversible sparse bit-sets. In *Principles and Practice of Constraint Programming - CP 2016*, volume 9892 of *LNCS*, pages 207–223. Springer, 2016.

[DHL$^+$20] Patrick Derbez, Paul Huynh, Virginie Lallemand, María Naya-Plasencia, Léo Perrin, and André Schrottenloher. Cryptanalysis results on spook - bringing full-round shadow-512 to the light. In Daniele Micciancio and Thomas Ristenpart, editors, *CRYPTO 2020, Part III*, volume 12172 of *LNCS*, pages 359–388. Springer, Heidelberg, August 2020. doi:10.1007/978-3-030-56877-1_13.

[DK05] J. Daemen and P. Kitsos. The Self-Synchronizing Stream Cipher MOSQUITO: eSTREAM Documentation, Version 2. *eSTREAM, ECRYPT Stream Cipher Project, Report 2005/018*, 2005. Available online at http://www.ecrypt.eu.org/stream/p3ciphers/mosquito/mosquito.pdf.

[DK08] J. Daemen and P. Kitsos. The self-synchronizing stream cipher moustique. In *New Stream Cipher Designs - The eSTREAM Finalists*, volume 4986 of *Lecture Notes in Computer Science*, pages 210–223. Springer, 2008.

[DK16] Orr Dunkelman and Liam Keliher, editors. *SAC 2015*, volume 9566 of *LNCS*. Springer, Heidelberg, August 2016.

[DKLS19] Orr Dunkelman, Nathan Keller, Eran Lambooij, and Yu Sasaki. A practical forgery attack on lilliput-AE. Cryptology ePrint Archive, Report 2019/867, 2019. https://eprint.iacr.org/2019/867.

[DKR97] Joan Daemen, Lars R. Knudsen, and Vincent Rijmen. The block cipher Square. In Biham [Bih97b], pages 149–165. doi:10.1007/BFb0052343.

[DKS10] Orr Dunkelman, Nathan Keller, and Adi Shamir. A practical-time related-key attack on the KASUMI cryptosystem used in GSM and 3G telephony. In Rabin [Rab10], pages 393–410. doi:10.1007/978-3-642-14623-7_21.

[DL18] Sébastien Duval and Gaëtan Leurent. MDS matrices with lightweight circuits. *IACR Trans. Symm. Cryptol.*, 2018(2):48–78, 2018. doi:10.13154/tosc.v2018.i2.48-78.

[DLU19] Patrick Derbez, Virginie Lallemand, and Aleksei Udovenko. Cryptanalysis of SKINNY in the framework of the SKINNY 2018-2019 cryptanalysis competition. In Kenneth G. Paterson and Douglas Stebila, editors, *SAC 2019*, volume 11959 of *LNCS*, pages 124–145. Springer, Heidelberg, August 2019. doi:10.1007/978-3-030-38471-5_6.

[DPAR00] Joan Daemen, Michaël Peeters, Gilles Van Assche, and Vincent Rijmen. Nessie proposal: the block cipher NOEKEON. Nessie submission, 2000. http://gro.noekeon.org/.

[DPU+16] Daniel Dinu, Léo Perrin, Aleksei Udovenko, Vesselin Velichkov, Johann Großschädl, and Alex Biryukov. Design strategies for ARX with provable bounds: Sparx and LAX. In Cheon and Takagi [CT16], pages 484–513. doi:10.1007/978-3-662-53887-6_18.

[DR01] Joan Daemen and Vincent Rijmen. The wide trail design strategy. In Bahram Honary, editor, *8th IMA International Conference on Cryptography and Coding*, volume 2260 of *LNCS*, pages 222–238. Springer, Heidelberg, December 2001.

[DR02] Joan Daemen and Vincent Rijmen. *The Design of Rijndael: AES - The Advanced Encryption Standard.* Information Security and Cryptography. Springer, 2002. URL: https://doi.org/10.1007/978-3-662-04722-4, doi:10.1007/978-3-662-04722-4.

[DR05] Joan Daemen and Vincent Rijmen. Probability distributions of correlation and differentials in block ciphers. Cryptology ePrint Archive, Report 2005/212, 2005. http://eprint.iacr.org/2005/212.

[DR07] Joan Daemen and Vincent Rijmen. Plateau characteristics. *IET Information Security*, 1(1):11–17, 2007. URL: https://doi.org/10.1049/iet-ifs:20060099, doi:10.1049/iet-ifs:20060099.

[DR11] Thai Duong and Juliano Rizzo. BEAST: Surprising Crypto Attack Against HTTPS. Blog post. http://netifera.com/research/beast/beast_DRAFT_0621.pdf, 2011.

[DS08] Hüseyin Demirci and Ali Aydin Selçuk. A meet-in-the-middle attack on 8-round AES. In Nyberg [Nyb08], pages 116–126. doi:10.1007/978-3-540-71039-4_7.

[DS09] Itai Dinur and Adi Shamir. Cube attacks on tweakable black box polynomials. In *Advances in Cryptology - EUROCRYPT 2009*, volume 5479 of *Lecture Notes in Computer Science*, pages 278–299. Springer, 2009.

[Dt08] Whitfield Diffie and George Ledin (translators). SMS4 encryption algorithm for wireless networks. Cryptology ePrint Archive, Report 2008/329, 2008. http://eprint.iacr.org/2008/329.

[Dwo01] Morris Dworkin. Nist special publication 800-38a, recommendation for block cipher modes of operation-methods and techniques. *National Institute of Standards and Technology/US Department of Commerce*, 2001. URL: https://csrc.nist.gov/publications/detail/sp/800-38a/final.

[Dwo04] Morris J. Dworkin. NIST SP 800-38C. Recommendation for block cipher modes of operation: The ccm mode for authentication and confidentiality. Technical report, Gaithersburg, MD, USA, 2004.

[Dwo07] Morris J. Dworkin. NIST SP 800-38D. Recommendation for block cipher modes of operation: Galois/counter mode (gcm) and gmac. Technical report, Gaithersburg, MD, USA, 2007.

[ECR18] Algorithms, key size and protocols report (2018), 2018. H2020-ICT-2014 - Project 645421, D5.4, https://www.ecrypt.eu.org/csa/documents/D5.4-FinalAlgKeySizeProt.pdf.

[EKKT19] Zahra Eskandari, Andreas Brasen Kidmose, Stefan Kölbl, and Tyge Tiessen. Finding integral distinguishers with ease. In Cid and Jacobson Jr: [CJ19], pages 115–138. doi:10.1007/978-3-030-10970-7_6.

[EKM+08] Thomas Eisenbarth, Timo Kasper, Amir Moradi, Christof Paar, Mahmoud Salmasizadeh, and Mohammad T. Manzuri Shalmani. On the power of power analysis in the real world: A complete break of the keeloqcode hopping scheme. In David Wagner, editor, *CRYPTO 2008*, volume 5157 of *LNCS*, pages 203–220. Springer, Heidelberg, August 2008. doi:10.1007/978-3-540-85174-5_12.

[EMPS13] Sareh Emami, Cameron McDonald, Josef Pieprzyk, and Ron Steinfeld. Truncated differential analysis of reduced-round LBlock. In Michel Abdalla, Cristina Nita-Rotaru, and Ricardo Dahab, editors, *CANS 13*, volume 8257 of *LNCS*, pages 291–308. Springer, Heidelberg, November 2013. doi:10.1007/978-3-319-02937-5_16.

[ENP19] Maria Eichlseder, Marcel Nageler, and Robert Primas. Analyzing the linear keystream biases in AEGIS. *IACR Trans. Symm. Cryptol.*, 2019(4):348–368, 2019. doi:10.13154/tosc.v2019.i4.348-368.

[Fei74] Horst Feistel. Block cipher cryptographic system, March 19 1974. US Patent 3,798,359.

[FEL] Fair Evaluation of LIghtweight Cryptographic Systems. https://www.cryptolux.org/index.php/FELICS. Accessed: 2019-02-14.

[FEL19] CryptoLUX > FELICS Avrora patch. https://www.cryptolux.org/index.php/FELICS_Avrora_patch, 2019. Accessed: 2019-03-07.

[Fer02] Niels Ferguson. Collision attacks on OCB. Technical report, 2002.

[Fiv16] Michael Fivez. Energy Efficient Hardware Implementations of CAESAR Submissions. Master's thesis, KU Leuven, 2016. Ingrid Verbauwhede (promotor).

[FJP13] Pierre-Alain Fouque, Jérémy Jean, and Thomas Peyrin. Structural evaluation of AES and chosen-key distinguisher of 9-round AES-128. In Canetti and Garay [CG13], pages 183–203. doi:10.1007/978-3-642-40041-4_11.

[FN20] Antonio Flórez-Gutiérrez and María Naya-Plasencia. Improving key-recovery in linear attacks: Application to 28-round PRESENT. In Anne Canteaut and Yuval Ishai, editors, *EUROCRYPT 2020, Part I*, volume 12105 of *LNCS*, pages 221–249. Springer, Heidelberg, May 2020. doi:10.1007/978-3-030-45721-1_9.

[FWG+16] Kai Fu, Meiqin Wang, Yinghua Guo, Siwei Sun, and Lei Hu. MILP-based automatic search algorithms for differential and linear trails for speck. In Peyrin [Pey16], pages 268–288. doi:10.1007/978-3-662-52993-5_14.

[GC91] Henri Gilbert and Guy Chassé. A statistical attack of the FEAL-8 cryptosystem. In Menezes and Vanstone [MV91], pages 22–33. doi:10.1007/3-540-38424-3_2.

[GD02] Virgil D. Gligor and Pompiliu Donescu. Fast encryption and authentication: XCBC encryption and XECB authentication modes. In Matsui [Mat02], pages 92–108. doi:10.1007/3-540-45473-X_8.

[GGNS13] Benoît Gérard, Vincent Grosso, María Naya-Plasencia, and François-Xavier Standaert. Block ciphers that are easier to mask: How far can we go? In Bertoni and Coron [BC13], pages 383–399. doi:10.1007/978-3-642-40349-1_22.

[GJK$^+$19] Dahmun Goudarzi, Jérémy Jean, Stefan Kölbl, Thomas Peyrin, Matthieu Rivain, Yu Sasaki, and Siang Meng Sim. Pyjamask. Submission to the NIST Lightweight Cryptography project. Available online https://csrc.nist.gov/CSRC/media/Projects/lightweight-cryptography/documents/round-2/spec-doc-rnd2/pyjamask-spec-round2.pdf., 2019.

[GL16] David Gérault and Pascal Lafourcade. Related-key cryptanalysis of midori. In Orr Dunkelman and Somitra Kumar Sanadhya, editors, *INDOCRYPT 2016*, volume 10095 of *LNCS*, pages 287–304. Springer, Heidelberg, December 2016. doi:10.1007/978-3-319-49890-4_16.

[GLMS18] David Gérault, Pascal Lafourcade, Marine Minier, and Christine Solnon. Revisiting AES related-key differential attacks with constraint programming. *Inf. Process. Lett.*, 139:24–29, 2018. URL: https://doi.org/10.1016/j.ipl.2018.07.001, doi:10.1016/j.ipl.2018.07.001.

[GLMS20] David Gérault, Pascal Lafourcade, Marine Minier, and Christine Solnon. Computing AES related-key differential characteristics with constraint programming. *Artificial Intelligence*, 278:103183, January 2020. URL: https://hal.archives-ouvertes.fr/hal-02327893, doi:10.1016/j.artint.2019.103183.

[GLS$^+$15] Vincent Grosso, Gaëtan Leurent, François-Xavier Standaert, Kerem Varıcı, François Durvaux, Lubos Gaspar, and Stéphanie Kerckhof. SCREAM v3, August 2015. Submission to the CAESAR competition. URL: https://competitions.cr.yp.to/round2/screamv3.pdf.

[GLSV15] Vincent Grosso, Gaëtan Leurent, François-Xavier Standaert, and Kerem Varici. LS-designs: Bitslice encryption for efficient masked software implementations. In Cid and Rechberger [CR15], pages 18–37. doi:10.1007/978-3-662-46706-0_2.

[GMO09] Juan A. Garay, Atsuko Miyaji, and Akira Otsuka, editors. *CANS 09*, volume 5888 of *LNCS*. Springer, Heidelberg, December 2009.

[GMS16] David Gérault, Marine Minier, and Christine Solnon. Constraint programming models for chosen key differential cryptanalysis. In *Principles and Practice of Constraint Programming - CP 2016*, volume 9892 of *LNCS*, pages 584–601. Springer, 2016.

[GNL11] Zheng Gong, Svetla Nikova, and Yee Wei Law. KLEIN: A new family of lightweight block ciphers. In Ari Juels and Christof Paar, editors, *RFID. Security and Privacy - 7th International Workshop, RFIDSec 2011, Amherst, USA, June 26-28, 2011, Revised Selected Papers*, volume 7055 of *Lecture Notes in Computer Science*, pages 1–18. Springer, 2011. URL: https://doi.org/10.1007/978-3-642-25286-0_1, doi:10.1007/978-3-642-25286-0_1.

[GNL12] Zheng Gong, Svetla Nikova, and Yee Wei Law. Klein: A new family of lightweight block ciphers. In Ari Juels and Christof Paar, editors, *RFID. Security and Privacy*, pages 1–18, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg.

[GP10] Henri Gilbert and Thomas Peyrin. Super-sbox cryptanalysis: Improved attacks for AES-like permutations. In Hong and Iwata [HI10], pages 365–383. doi:10.1007/978-3-642-13858-4_21.

[GP16] Benedikt Gierlichs and Axel Y. Poschmann, editors. *CHES 2016*, volume 9813 of *LNCS*. Springer, Heidelberg, August 2016.

[GPPR11] Jian Guo, Thomas Peyrin, Axel Poschmann, and Matthew J. B. Robshaw. The LED block cipher. In Preneel and Takagi [PT11], pages 326–341. doi:10.1007/978-3-642-23951-9_22.

[GR15] Rosario Gennaro and Matthew J. B. Robshaw, editors. *CRYPTO 2015, Part I*, volume 9215 of *LNCS*. Springer, Heidelberg, August 2015.

[HAV18] Mathias Hall-Andersen and Philip S. Vejre. Generating graphs packed with paths. *IACR Trans. Symm. Cryptol.*, 2018(3):265–289, 2018. doi:10.13154/tosc.v2018.i3.265-289.

[Hel80] M. E. Hellman. A cryptanalytic time-memory trade-off. *IEEE Trans. Information Theory*, 26(4):401–406, 1980.

[Hel94] Tor Helleseth, editor. *EUROCRYPT'93*, volume 765 of *LNCS*. Springer, Heidelberg, May 1994.

[HI10] Seokhie Hong and Tetsu Iwata, editors. *FSE 2010*, volume 6147 of *LNCS*. Springer, Heidelberg, February 2010.

[HJMM08] Martin Hell, Thomas Johansson, Alexander Maximov, and Willi Meier. The grain family of stream ciphers. In Matthew J. B. Robshaw and Olivier Billet, editors, *New Stream Cipher Designs - The eSTREAM Finalists*, volume 4986 of *Lecture Notes in Computer Science*, pages 179–190. Springer, 2008. URL: https://doi.org/10.1007/978-3-540-68351-3_14, doi:10.1007/978-3-540-68351-3_14.

[HKM95] Carlo Harpes, Gerhard G. Kramer, and James L. Massey. A generalization of linear cryptanalysis and the applicability of Matsui's piling-up lemma. In Louis C. Guillou and Jean-Jacques Quisquater, editors, *EUROCRYPT'95*, volume 921 of *LNCS*, pages 24–38. Springer, Heidelberg, May 1995. doi:10.1007/3-540-49264-X_3.

[HLK$^+$14] Deukjo Hong, Jung-Keun Lee, Dong-Chan Kim, Daesung Kwon, Kwon Ho Ryu, and Dong-Geon Lee. LEA: A 128-bit block cipher for fast encryption on common processors. In Yongdae Kim, Heejo Lee, and Adrian Perrig, editors, *WISA 13*, volume 8267 of *LNCS*, pages 3–27. Springer, Heidelberg, August 2014. doi:10.1007/978-3-319-05149-9_1.

[HPRM04] P. Hawkes, M. Paddon, G. G. Rose, and W. V. Miriam. Primitive specification for sss. Technical report, e-Stream Project, 2004. Available at: http://www.ecrypt.eu.org/stream/ciphers/sss/sss.pdf.

[HR10] Viet Tung Hoang and Phillip Rogaway. On generalized Feistel networks. In Rabin [Rab10], pages 613–630. doi:10.1007/978-3-642-14623-7_33.

[HW19] Kai Hu and Meiqin Wang. Automatic search for a variant of division property using three subsets. In Mitsuru Matsui, editor, *CT-RSA 2019*, volume 11405 of *LNCS*, pages 412–432. Springer, Heidelberg, March 2019. doi:10.1007/978-3-030-12612-4_21.

[IIMP19] Akiko Inoue, Tetsu Iwata, Kazuhiko Minematsu, and Bertram Poettering. Cryptanalysis of OCB2: Attacks on authenticity and confidentiality. In Alexandra Boldyreva and Daniele Micciancio, editors, *CRYPTO 2019, Part I*, volume 11692 of *LNCS*, pages 3–31. Springer, Heidelberg, August 2019. doi:10.1007/978-3-030-26948-7_1.

[IK03] Tetsu Iwata and Kaoru Kurosawa. OMAC: One-key CBC MAC. In Johansson [Joh03], pages 129–153. doi:10.1007/978-3-540-39887-5_11.

[IKD$^+$08] Sebastiaan Indesteege, Nathan Keller, Orr Dunkelman, Eli Biham, and Bart Preneel. A practical attack on KeeLoq. In Nigel P. Smart, editor, *EUROCRYPT 2008*, volume 4965 of *LNCS*, pages 1–18. Springer, Heidelberg, April 2008. doi:10.1007/978-3-540-78967-3_1.

[IKMP19a] Tetsu Iwata, Mustafa Khairallah, Kazuhiko Minematsu, and Thomas Peyrin. Remus. Submission to the NIST Lightweight Cryptography project. Available online https://csrc.nist.gov/CSRC/media/Projects/Lightweight-Cryptography/documents/round-1/spec-doc/Remus-spec.pdf., 2019.

[IKMP19b] Tetsu Iwata, Mustafa Khairallah, Kazuhiko Minematsu, and Thomas Peyrin. Romulus. Submission to the NIST Lightweight Cryptography project. Available online https://csrc.nist.gov/CSRC/media/Projects/lightweight-cryptography/documents/round-2/spec-doc-rnd2/Romulus-spec-round2.pdf., 2019.

[IM18] Akiko Inoue and Kazuhiko Minematsu. Cryptanalysis of OCB2. Cryptology ePrint Archive, Report 2018/1040, 2018. https://eprint.iacr.org/2018/1040.

[IMGM15] Tetsu Iwata, Kazuhiko Minematsu, Jian Guo, and Sumio Morioka. CLOC: Authenticated encryption for short input. In Cid and Rechberger [CR15], pages 149–167. doi:10.1007/978-3-662-46706-0_8.

[IMPS17] Tetsu Iwata, Kazuhiko Minematsu, Thomas Peyrin, and Yannick Seurin. ZMAC: A fast tweakable block cipher mode for highly secure message authentication. In Jonathan Katz and Hovav Shacham, editors, *CRYPTO 2017, Part III*, volume 10403 of *LNCS*, pages 34–65. Springer, Heidelberg, August 2017. doi:10.1007/978-3-319-63697-9_2.

[IPS13] Mitsugu Iwamoto, Thomas Peyrin, and Yu Sasaki. Limited-birthday distinguishers for hash functions - collisions beyond the birthday bound can be meaningful. In Kazue Sako and Palash Sarkar, editors, *ASIACRYPT 2013, Part II*, volume 8270 of *LNCS*, pages 504–523. Springer, Heidelberg, December 2013. doi:10.1007/978-3-642-42045-0_26.

[ISO12a] ISO/IEC 29192-1:2012 Information technology – Security techniques – Lightweight cryptography – Part 1: Ggeneral, 2012. https://www.iso.org/standard/56425.html.

[ISO12b] ISO/IEC 29192-2:2012 Information technology – Security techniques – Lightweight cryptography – Part 2: Block ciphers, 2012. https://www.iso.org/standard/56552.html.

[Iwa18] Tetsu Iwata. Plaintext recovery attack of OCB2. Cryptology ePrint Archive, Report 2018/1090, 2018. https://eprint.iacr.org/2018/1090.

[J-L19] J-Link Software and Documentation Pack. https://www.segger.com/downloads/jlink/#J-LinkSoftwareAndDocumentationPack, 2019. Accessed: 2019-02-26.

[Jak98] Thomas Jakobsen. Cryptanalysis of block ciphers with probabilistic non-linear relations of low degree. In Hugo Krawczyk, editor, *CRYPTO'98*, volume 1462 of *LNCS*, pages 212–222. Springer, Heidelberg, August 1998. doi:10.1007/BFb0055730.

[Jea16] Jérémy Jean. TikZ for Cryptographers. https://www.iacr.org/authors/tikz/, 2016.

[JK97] Thomas Jakobsen and Lars R. Knudsen. The interpolation attack on block ciphers. In Biham [Bih97b], pages 28–40. doi:10.1007/BFb0052332.

[JM03] A. Joux and F. Muller. Loosening the KNOT. In *Fast Software Encryption - FSE 2003*, volume 2887 of *Lecture Notes in Computer Science*, pages 87–99. Springer, 2003.

[JM05] A. Joux and F. Muller. Two attacks against the HBB stream cipher. In *Fast Software Encryption - FSE 2005*, volume 3557 of *Lecture Notes in Computer Science*, pages 330–341. Springer, 2005.

[JM06] A. Joux and F. Muller. Chosen-ciphertext attacks against MOSQUITO. In *Fast Software Encryption - FSE 2006*, volume 4047 of *Lecture Notes in Computer Science*, pages 390–404. Springer, 2006.

[JNP14] Jérémy Jean, Ivica Nikolic, and Thomas Peyrin. Tweaks and keys for block ciphers: The TWEAKEY framework. In Palash Sarkar and Tetsu Iwata, editors, *ASIACRYPT 2014, Part II*, volume 8874 of *LNCS*, pages 274–288. Springer, Heidelberg, December 2014. doi:10.1007/978-3-662-45608-8_15.

[JNPS16] Jérémy Jean, Ivica Nikolic, Thomas Peyrin, and Yannick Seurin. Deoxys v1.43. *Submitted to CAESAR*, 2016.

[Joh03] Thomas Johansson, editor. *FSE 2003*, volume 2887 of *LNCS*. Springer, Heidelberg, February 2003.

[Jou11] Antoine Joux, editor. *FSE 2011*, volume 6733 of *LNCS*. Springer, Heidelberg, February 2011.

[Jou14] Antoine Joux. A new index calculus algorithm with complexity $L(1/4 + o(1))$ in small characteristic. In Lange et al. [LLL14], pages 355–379. doi:10.1007/978-3-662-43414-7_18.

[JPS17] Jérémy Jean, Thomas Peyrin, and Siang Meng Sim. Optimizing implementations of lightweight building blocks. Cryptology ePrint Archive, Report 2017/101, 2017. http://eprint.iacr.org/2017/101.

[Jut01] Charanjit S. Jutla. Encryption modes with almost free message integrity. In Pfitzmann [Pfi01], pages 529–544. doi:10.1007/3-540-44987-6_32.

[Jut08] Charanjit S. Jutla. Encryption modes with almost free message integrity. *Journal of Cryptology*, 21(4):547–578, October 2008. doi:10.1007/s00145-008-9024-z.

[JV04] Pascal Junod and Serge Vaudenay. FOX: A new family of block ciphers. In Helena Handschuh and Anwar Hasan, editors, *SAC 2004*, volume 3357 of *LNCS*, pages 114–129. Springer, Heidelberg, August 2004. doi:10.1007/978-3-540-30564-4_8.

[Kah96] David Kahn. *The Codebreakers: The Comprehensive History of Secret Communication from Ancient Times to the Internet.* Simon and Schuster, 1996.

[KCS11] Stéphanie Kerckhof, Baudoin Collard, and François-Xavier Standaert. FPGA implementation of a statistical saturation attack against PRESENT. In Abderrahmane Nitaj and David Pointcheval, editors, *AFRICACRYPT 11*, volume 6737 of *LNCS*, pages 100–116. Springer, Heidelberg, July 2011.

[Ker83] Auguste Kerckhoffs. La cryptographie militaire. (French) [Military cryptography]. *Journal des Sciences Militaires*, IX:5–38, 161–191, January/ February 1883. URL: http://www.cl.cam.ac.uk/~fapp2/kerckhoffs/la_cryptographie_militaire_i.htmhttps://www.petitcolas.net/kerckhoffs/;https://www.petitcolas.net/kerckhoffs/crypto_militaire_1_b.pdf;https://www.petitcolas.net/kerckhoffs/crypto_militaire_2.pdf.

[KG16] Pierre Karpman and Benjamin Grégoire. The littlun s-box and the fly block cipher. 2016. https://pdfs.semanticscholar.org/5487/ccb5b874c1e56f1b8468eef2def91de42d36.pdf.

[KHS+03] Jongsung Kim, Seokhie Hong, Jaechul Sung, Changhoon Lee, and Sangjin Lee. Impossible differential cryptanalysis for block cipher structures. In Thomas Johansson and Subhamoy Maitra, editors, *INDOCRYPT 2003*, volume 2904 of *LNCS*, pages 82–96. Springer, Heidelberg, December 2003.

[KKH+04] Jongsung Kim, Guil Kim, Seokhie Hong, Sangjin Lee, and Dowon Hong. The related-key rectangle attack - application to SHACAL-1. In Huaxiong Wang, Josef Pieprzyk, and Vijay Varadharajan, editors, *ACISP 04*, volume 3108 of *LNCS*, pages 123–136. Springer, Heidelberg, July 2004. doi:10.1007/978-3-540-27800-9_11.

[KKMP09] Markus Kasper, Timo Kasper, Amir Moradi, and Christof Paar. Breaking KeeLoq in a flash: On extracting keys at lightning speed. In Bart Preneel, editor, *AFRICACRYPT 09*, volume 5580 of *LNCS*, pages 403–420. Springer, Heidelberg, June 2009.

[KKS01] John Kelsey, Tadayoshi Kohno, and Bruce Schneier. Amplified boomerang attacks against reduced-round MARS and Serpent. In Schneier [Sch01], pages 75–93. doi:10.1007/3-540-44706-7_6.

[Klí05] V. Klíma. Cryptanalysis of hiji-bij-bij (HBB). IACR Cryptology ePrint Archive, Report 2005/3, 2005.

[KLK13] Taekyoung Kwon, Mun-Kyu Lee, and Daesung Kwon, editors. *ICISC 12*, volume 7839 of *LNCS*. Springer, Heidelberg, November 2013.

[KLT15] Stefan Kölbl, Gregor Leander, and Tyge Tiessen. Observations on the SIMON block cipher family. In Gennaro and Robshaw [GR15], pages 161–185. doi:10.1007/978-3-662-47989-6_8.

[Knu95] Lars R. Knudsen. Truncated and higher order differentials. In Preneel [Pre95], pages 196–211. doi:10.1007/3-540-60590-8_16.

[Knu98] Lars Knudsen. Deal - a 128-bit block cipher. In *NIST AES Proposal*, 1998.

[Knu99] Lars R. Knudsen, editor. *FSE'99*, volume 1636 of *LNCS*. Springer, Heidelberg, March 1999.

[KR11a] Lars R. Knudsen and Matthew Robshaw. *The Block Cipher Companion*. Information Security and Cryptography. Springer, 2011. URL: https://doi.org/10.1007/978-3-642-17342-4, doi:10.1007/978-3-642-17342-4.

[KR11b] Ted Krovetz and Phillip Rogaway. The software performance of authenticated-encryption modes. In Joux [Jou11], pages 306–327. doi:10.1007/978-3-642-21702-9_18.

[KR14] Ted Krovetz and Phillip Rogaway. The OCB Authenticated-Encryption Algorithm. RFC 7253, May 2014. URL: https://rfc-editor.org/rfc/rfc7253.txt, doi:10.17487/RFC7253.

[Kra01] Hugo Krawczyk. The order of encryption and authentication for protecting communications (or: How secure is SSL?). In Joe Kilian, editor, *CRYPTO 2001*, volume 2139 of *LNCS*, pages 310–331. Springer, Heidelberg, August 2001. doi:10.1007/3-540-44647-8_19.

[KRB$^+$04] E. Kasper, V. Rijmen, E. Bjorstad, C. Rechberger, M. Robshaw, and G. Sekar. Correlated Keystreams in MOUSTIQUE. Technical report, ESTREAM Project, 2004.

[KRB$^+$08] E. Käsper, V. Rijmen, T. E. Bjørstad, C. Rechberger, M. J. B. Robshaw, and G. Sekar. Correlated keystreams in moustique. In *Progress in Cryptology - AFRICACRYPT 2008*, volume 5023 of *Lecture Notes in Computer Science*, pages 246–257. Springer, 2008.

[KSK13] Takuma Koyama, Yu Sasaki, and Noboru Kunihiro. Multi-differential cryptanalysis on reduced DM-PRESENT-80: Collisions and other differential properties. In Kwon et al. [KLK13], pages 352–367. doi:10.1007/978-3-642-37682-5_25.

[KVW04] Tadayoshi Kohno, John Viega, and Doug Whiting. CWC: A high-performance conventional authenticated encryption mode. In Roy and Meier [RM04], pages 408–426. doi:10.1007/978-3-540-25937-4_26.

[KW19] Thorsten Kleinjung and Benjamin Wesolowski. Discrete logarithms in quasi-polynomial time in finite fields of fixed characteristic. *CoRR*, abs/1906.10668, 2019. URL: http://arxiv.org/abs/1906.10668, arXiv:1906.10668.

[KY01] Jonathan Katz and Moti Yung. Unforgeable encryption and chosen ciphertext secure modes of operation. In Schneier [Sch01], pages 284–299. doi:10.1007/3-540-44706-7_20.

[KY10] E.B. Kavun and T. Yalcin. A Lightweight Implementation of Keccak Hash Function for Radio-Frequency Identification Applications. In *Radio Frequency Identification:*

Security and Privacy Issues - RFIDSec, volume 6370 of *Lecture Notes in Computer Science*, pages 258–269. Springer, 2010.

[LAAZ11] Gregor Leander, Mohamed Ahmed Abdelraheem, Hoda AlKhzaimi, and Erik Zenner. A cryptanalysis of PRINTcipher: The invariant subspace attack. In Rogaway [Rog11], pages 206–221. doi:10.1007/978-3-642-22792-9_12.

[Laf18] Frédéric Lafitte. Cryptosat: a tool for sat-based cryptanalysis. *IET Information Security*, 12(6):463–474, 2018.

[Lai94] Xuejia Lai. *Higher Order Derivatives and Differential Cryptanalysis*, pages 227–233. Springer US, Boston, MA, 1994. URL: https://doi.org/10.1007/978-1-4615-2694-0_23, doi:10.1007/978-1-4615-2694-0_23.

[LCM+17] Fanghui Liu, Waldemar Cruz, Chujiao Ma, Greg Johnson, and Laurent Michel. A tolerant algebraic side-channel attack on AES using CP. In *Principles and Practice of Constraint Programming - 23rd International Conference, CP 2017, Melbourne, VIC, Australia, August 28 - September 1, 2017, Proceedings*, volume 10416 of *Lecture Notes in Computer Science*, pages 189–205. Springer, 2017.

[Lea11] Gregor Leander. On linear hulls, statistical saturation attacks, PRESENT and a cryptanalysis of PUFFIN. In Kenneth G. Paterson, editor, *EUROCRYPT 2011*, volume 6632 of *LNCS*, pages 303–322. Springer, Heidelberg, May 2011. doi:10.1007/978-3-642-20465-4_18.

[Leu16] Gaëtan Leurent. Differential forgery attack against LAC. In Dunkelman and Keliher [DK16], pages 217–224. doi:10.1007/978-3-319-31301-6_13.

[LGS17] Guozhen Liu, Mohona Ghosh, and Ling Song. Security analysis of SKINNY under related-tweakey settings (long paper). *IACR Trans. Symm. Cryptol.*, 2017(3):37–72, 2017. doi:10.13154/tosc.v2017.i3.37-72.

[LGW12] Shusheng Liu, Zheng Gong, and Libin Wang. Improved related-key differential attacks on reduced-round LBlock. In Tat Wing Chim and Tsz Hon Yuen, editors, *ICICS 12*, volume 7618 of *LNCS*, pages 58–69. Springer, Heidelberg, October 2012. doi:10.1007/978-3-642-34129-8_6.

[LH94] Susan K. Langford and Martin E. Hellman. Differential-linear cryptanalysis. In Yvo Desmedt, editor, *CRYPTO'94*, volume 839 of *LNCS*, pages 17–25. Springer, Heidelberg, August 1994. doi:10.1007/3-540-48658-5_3.

[LKT13] Arjen K. Lenstra, Thorsten Kleinjung, and Emmanuel Thomé. Universal security; from bits and mips to pools, lakes – and beyond. Cryptology ePrint Archive, Report 2013/635, 2013. http://eprint.iacr.org/2013/635.

[LLJ+19] Xianhui Lu, Yamin Liu, Dingding Jia, Haiyang Xue, Jingnan He, Zhenfei Zhang, Zhe Liu, Hao Yang, Bao Li, and Kunpeng Wang. LAC. Technical report, National Institute of Standards and Technology, 2019. available at https://csrc.nist.gov/projects/post-quantum-cryptography/round-2-submissions.

[LLL14] Tanja Lange, Kristin Lauter, and Petr Lisonek, editors. *SAC 2013*, volume 8282 of *LNCS*. Springer, Heidelberg, August 2014.

[LMM91a] Xuejia Lai, James L. Massey, and Sean Murphy. Markov ciphers and differential cryptanalysis. In Donald W. Davies, editor, *Advances in Cryptology - EUROCRYPT '91, Workshop on the Theory and Application of of Cryptographic Techniques, Brighton, UK, April 8-11, 1991, Proceedings*, volume 547 of *Lecture Notes in Computer Science*, pages 17–38. Springer, 1991. URL: https://doi.org/10.1007/3-540-46416-6_2, doi:10.1007/3-540-46416-6_2.

[LMM91b] Xuejia Lai, James L. Massey, and Sean Murphy. Markov ciphers and differential cryptanalysis. In Davies [Dav91], pages 17–38. doi:10.1007/3-540-46416-6_2.

[LMR15] Gregor Leander, Brice Minaud, and Sondre Rønjom. A generic approach to invariant subspace attacks: Cryptanalysis of robin, iSCREAM and Zorro. In Oswald and Fischlin [OF15], pages 254–283. doi:10.1007/978-3-662-46800-5_11.

[LP07] G. Leander and A. Poschmann. On the Classification of 4 Bit S-Boxes. In *WAIFI*, volume 4547 of *Lecture Notes in Computer Science*, pages 159–176. Springer, 2007.

[LPR$^+$19] Elena Andreevaand Virginie Lallemand, Antoon Purnal, Reza Reyhanitabar, Arnab Roy, and Damian Vizár. ForkAE. Submission to the NIST Lightweight Cryptography project. Available online https://csrc.nist.gov/CSRC/media/Projects/lightweight-cryptography/documents/round-2/spec-doc-rnd2/forkae-spec-round2.pdf., 2019.

[LQSL19] Kangquan Li, Longjiang Qu, Bing Sun, and Chao Li. New results about the boomerang uniformity of permutation polynomials. *IEEE Trans. Information Theory*, 65(11):7542–7553, 2019. URL: https://doi.org/10.1109/TIT.2019.2918531, doi:10.1109/TIT.2019.2918531.

[LRW02] Moses Liskov, Ronald L. Rivest, and David Wagner. Tweakable block ciphers. In Moti Yung, editor, *CRYPTO 2002*, volume 2442 of *LNCS*, pages 31–46. Springer, Heidelberg, August 2002. doi:10.1007/3-540-45708-9_3.

[LS19] Yunwen Liu and Yu Sasaki. Related-key boomerang attacks on GIFT with automated trail search including BCT effect. In Julian Jang-Jaccard and Fuchun Guo, editors, *ACISP 19*, volume 11547 of *LNCS*, pages 555–572. Springer, Heidelberg, July 2019. doi:10.1007/978-3-030-21548-4_30.

[LTW18] Gregor Leander, Cihangir Tezcan, and Friedrich Wiemer. Searching for subspace trails and truncated differentials. *IACR Trans. Symm. Cryptol.*, 2018(1):74–100, 2018. doi:10.13154/tosc.v2018.i1.74-100.

[Lu16] Jiqiang Lu. On the security of the LAC authenticated encryption algorithm. In Joseph K. Liu and Ron Steinfeld, editors, *ACISP 16, Part II*, volume 9723 of *LNCS*, pages 395–408. Springer, Heidelberg, July 2016. doi:10.1007/978-3-319-40367-0_25.

[Luc02] Stefan Lucks. The saturation attack - a bait for Twofish. In Matsui [Mat02], pages 1–15. doi:10.1007/3-540-45473-X_1.

[LWLG09] Yiyuan Luo, Zhongming Wu, Xuejia Lai, and Guang Gong. A unified method for finding impossible differentials of block cipher structures. Cryptology ePrint Archive, Report 2009/627, 2009. http://eprint.iacr.org/2009/627.

[LWZ16] Li Lin, Wenling Wu, and Yafei Zheng. Automatic search for key-bridging technique: Applications to LBlock and TWINE. In Peyrin [Pey16], pages 247–267. doi:10.1007/978-3-662-52993-5_13.

[Mas69] James L. Massey. Shift-register synthesis and BCH decoding. *IEEE Trans. Inf. Theory*, 15(1):122–127, 1969. URL: https://doi.org/10.1109/TIT.1969.1054260, doi:10.1109/TIT.1969.1054260.

[Mat94] Mitsuru Matsui. Linear cryptanalysis method for DES cipher. In Helleseth [Hel94], pages 386–397. doi:10.1007/3-540-48285-7_33.

[Mat95] Mitsuru Matsui. On correlation between the order of S-boxes and the strength of DES. In Santis [San95], pages 366–375. doi:10.1007/BFb0053451.

[Mat97] Mitsuru Matsui. New block encryption algorithm MISTY. In Biham [Bih97b], pages 54–68. doi:10.1007/BFb0052334.

[Mat02] Mitsuru Matsui, editor. *FSE 2001*, volume 2355 of *LNCS*. Springer, Heidelberg, April 2002.

[Mau91] U. M. Maurer. New approaches to the design of self-synchronizing stream ciphers. In *Advances in Cryptology - EUROCRYPT '91*, volume 547 of *Lecture Notes in Computer Science*, pages 458–471. Springer, 1991.

[Mau02] U. M. Maurer. Indistinguishability of Random Systems. In *Advances in Cryptology - EUROCRYPT 2002*, volume 2332 of *Lecture Notes in Computer Science*, pages 110–132. Springer, 2002.

[MB15] G. Millerioux and T. Boukhobza. Characterization of flat outputs for LPV discrete-time systems: A graph-oriented approach. In *54th IEEE Conference on Decision and Control, CDC 2015*, pages 759–764. IEEE, 2015.

[MBTM17] Kerry A. McKay, Lawrence E. Bassham, Meltem Sonmez Turan, and Nicky W. Mouha. Report on lightweight cryptography. https://nvlpubs.nist.gov/nistpubs/ir/2017/NIST.IR.8114.pdf, 2017.

[Min14] Kazuhiko Minematsu. Parallelizable rate-1 authenticated encryption from pseudorandom functions. In Nguyen and Oswald [NO14], pages 275–292. doi:10.1007/978-3-642-55220-5_16.

[Miy91] Shoji Miyaguchi. The FEAL cipher family (impromptu talk). In Menezes and Vanstone [MV91], pages 627–638. doi:10.1007/3-540-38424-3_46.

[MMS18] F Moazami, AR Mehrdad, and Hadi Soleimany. Impossible differential cryptanalysis on deoxys-bc-256. *The ISC International Journal of Information Security*, 10(2):93–105, 2018.

[MNV18] Nicolas Marrière, Valérie Nachef, and Emmanuel Volte. Differential attacks on reduced round LILLIPUT. In Willy Susilo and Guomin Yang, editors, *ACISP 18*, volume 10946 of *LNCS*, pages 188–206. Springer, Heidelberg, July 2018. doi:10.1007/978-3-319-93638-3_12.

[MP13] Nicky Mouha and Bart Preneel. A proof that the ARX cipher salsa20 is secure against differential cryptanalysis. *IACR Cryptology ePrint Archive*, 2013:328, 2013. URL: http://eprint.iacr.org/2013/328.

[MSP19] MSP430-GCC-OPENSOURCE GCC - Open Source Compiler fro MSP Microcontrollers. http://www.ti.com/tool/msp430-gcc-opensource, 2019. Accessed: 2019-03-07.

[Mur11] Sean Murphy. The return of the cryptographic boomerang. *IEEE Trans. Information Theory*, 57(4):2517–2521, 2011. URL: https://doi.org/10.1109/TIT.2011.2111091, doi:10.1109/TIT.2011.2111091.

[MV91] Alfred J. Menezes and Scott A. Vanstone, editors. *CRYPTO'90*, volume 537 of *LNCS*. Springer, Heidelberg, August 1991.

[MV04] David A. McGrew and John Viega. The security and performance of the galois/-counter mode of operation (full version). Cryptology ePrint Archive, Report 2004/193, 2004. http://eprint.iacr.org/2004/193.

[MWGP11] Nicky Mouha, Qingju Wang, Dawu Gu, and Bart Preneel. Differential and linear cryptanalysis using mixed-integer linear programming. In Chuankun Wu, Moti Yung, and Dongdai Lin, editors, *Information Security and Cryptology - 7th International Conference, Inscrypt 2011, Beijing, China, November 30 - December 3, 2011. Revised Selected Papers*, volume 7537 of *Lecture Notes in Computer Science*, pages 57–76. Springer, 2011. URL: https://doi.org/10.1007/978-3-642-34704-7_5, doi:10.1007/978-3-642-34704-7_5.

[Nat95] National Institute of Standards and Technology. *FIPS PUB 180-1: Secure Hash Standard*. April 1995. Supersedes FIPS PUB 180 1993 May 11. URL: http://www.itl.nist.gov/fipspubs/fip180-1.htm.

[NISa] Lightweight Cryptography | CSRC. https://csrc.nist.gov/Projects/Lightweight-Cryptography. Accessed: 2019-07-23.

[NISb] Submission requirements and evaluation criteria for the lightweight cryptography standardization process. https://csrc.nist.gov/CSRC/media/Projects/Lightweight-Cryptography/documents/final-lwc-submission-requirements-august2018.pdf. Accessed: 2019-07-24.

[NK95] Kaisa Nyberg and Lars R. Knudsen. Provable security against a differential attack. *Journal of Cryptology*, 8(1):27–37, December 1995. doi:10.1007/BF00204800.

[NO14] Phong Q. Nguyen and Elisabeth Oswald, editors. *EUROCRYPT 2014*, volume 8441 of *LNCS*. Springer, Heidelberg, May 2014.

[NRS11] Svetla Nikova, Vincent Rijmen, and Martin Schläffer. Secure hardware implementation of nonlinear functions in the presence of glitches. *Journal of Cryptology*, 24(2):292–321, April 2011. doi:10.1007/s00145-010-9085-7.

[NRS14] Chanathip Namprempre, Phillip Rogaway, and Thomas Shrimpton. Reconsidering generic composition. In Nguyen and Oswald [NO14], pages 257–274. doi:10.1007/978-3-642-55220-5_15.

[NS19] Yusuke Naito and Takeshi Sugawara. Lightweight authenticated encryption mode of operation for tweakable block ciphers. *IACR TCHES*, 2020(1):66–94, 2019. https://tches.iacr.org/index.php/TCHES/article/view/8393. doi:10.13154/tches.v2020.i1.66-94.

[NSB$^+$07] Nicholas Nethercote, Peter J. Stuckey, Ralph Becket, Sebastian Brand, Gregory J. Duck, and Guido Tack. Minizinc: Towards a standard CP modelling language. In *Principles and Practice of Constraint Programming - CP 2007*, volume 4741 of *LNCS*, pages 529–543. Springer, 2007.

[NSZW09] Jorge Nakahara Jr., Pouyan Sepehrdad, Bingsheng Zhang, and Meiqin Wang. Linear (hull) and algebraic cryptanalysis of the block cipher PRESENT. In Garay et al. [GMO09], pages 58–75.

[Nyb94] Kaisa Nyberg. Differentially uniform mappings for cryptography. In Helleseth [Hel94], pages 55–64. doi:10.1007/3-540-48285-7_6.

[Nyb95] Kaisa Nyberg. Linear approximation of block ciphers (rump session). In Santis [San95], pages 439–444. doi:10.1007/BFb0053460.

[Nyb96] Kaisa Nyberg. Generalized Feistel networks. In Kwangjo Kim and Tsutomu Matsumoto, editors, *ASIACRYPT'96*, volume 1163 of *LNCS*, pages 91–104. Springer, Heidelberg, November 1996. doi:10.1007/BFb0034838.

[Nyb08] Kaisa Nyberg, editor. *FSE 2008*, volume 5086 of *LNCS*. Springer, Heidelberg, February 2008.

[OBSC10] Dag Arne Osvik, Joppe W. Bos, Deian Stefan, and David Canright. Fast software AES encryption. In Hong and Iwata [HI10], pages 75–93. doi:10.1007/978-3-642-13858-4_5.

[OF15] Elisabeth Oswald and Marc Fischlin, editors. *EUROCRYPT 2015, Part I*, volume 9056 of *LNCS*. Springer, Heidelberg, April 2015.

[Opt18] Gurobi Optimization. Gurobi optimizer reference manual, 2018. URL: http://www.gurobi.com.

[oST15] National Institute of Standards and Technology. *SHA-3 Standard: Permutation-Based Hash and Extendable-Output Functions: Federal Information Processing Standards Publications (FIPS PUBS) 202.* 2015. URL: http://dx.doi.org/10.6028/NIST.FIPS.202.

[ÖVTK09] Onur Özen, Kerem Varici, Cihangir Tezcan, and Çelebi Kocair. Lightweight block ciphers revisited: Cryptanalysis of reduced round PRESENT and HIGHT. In Colin Boyd and Juan Manuel González Nieto, editors, *ACISP 09*, volume 5594 of *LNCS*, pages 90–107. Springer, Heidelberg, July 2009.

[Pey16] Thomas Peyrin, editor. *FSE 2016*, volume 9783 of *LNCS*. Springer, Heidelberg, March 2016.

[Pfi01] Birgit Pfitzmann, editor. *EUROCRYPT 2001*, volume 2045 of *LNCS*. Springer, Heidelberg, May 2001.

[PFL16] Charles Prud'homme, Jean-Guillaume Fages, and Xavier Lorca. *Choco Documentation*. TASC, INRIA Rennes, LINA CNRS UMR 6241, COSLING S.A.S., 2016. URL: http://www.choco-solver.org.

[Poe18] Bertram Poettering. Breaking the confidentiality of OCB2. Cryptology ePrint Archive, Report 2018/1087, 2018. https://eprint.iacr.org/2018/1087.

[Pol19] Michel Pollet. simavr. https://github.com/buserror/simavr, 2019. Accessed: 2019-03-07.

[Pre95] Bart Preneel, editor. *FSE'94*, volume 1008 of *LNCS*. Springer, Heidelberg, December 1995.

[PS16] Thomas Peyrin and Yannick Seurin. Counter-in-tweak: Authenticated encryption modes for tweakable block ciphers. In Robshaw and Katz [RK16a], pages 33–63. doi:10.1007/978-3-662-53018-4_2.

[PT11] Bart Preneel and Tsuyoshi Takagi, editors. *CHES 2011*, volume 6917 of *LNCS*. Springer, Heidelberg, September / October 2011.

[Rab10] Tal Rabin, editor. *CRYPTO 2010*, volume 6223 of *LNCS*. Springer, Heidelberg, August 2010.

[RBBK01] Phillip Rogaway, Mihir Bellare, John Black, and Ted Krovetz. OCB: A block-cipher mode of operation for efficient authenticated encryption. In Michael K. Reiter and Pierangela Samarati, editors, *ACM CCS 2001*, pages 196–205. ACM Press, November 2001. doi:10.1145/501983.502011.

[RBW06] Francesca Rossi, Peter van Beek, and Toby Walsh. *Handbook of Constraint Programming (Foundations of Artificial Intelligence)*. Elsevier Science Inc., New York, NY, USA, 2006.

[Riv98] R. L. Rivest. A Description of the RC2(r) Encryption Algorithm. Network Working Group, RFC 2268, march 1998. http://tools.ietf.org/html/rfc2268.

[Riv09] Matthieu Rivain. Differential fault analysis on DES middle rounds. In Christophe Clavier and Kris Gaj, editors, *CHES 2009*, volume 5747 of *LNCS*, pages 457–469. Springer, Heidelberg, September 2009. doi:10.1007/978-3-642-04138-9_32.

[RK16a] Matthew Robshaw and Jonathan Katz, editors. *CRYPTO 2016, Part I*, volume 9814 of *LNCS*. Springer, Heidelberg, August 2016.

[RK16b] Matthew Robshaw and Jonathan Katz, editors. *CRYPTO 2016, Part II*, volume 9815 of *LNCS*. Springer, Heidelberg, August 2016.

[RM04] Bimal K. Roy and Willi Meier, editors. *FSE 2004*, volume 3017 of *LNCS*. Springer, Heidelberg, February 2004.

[Rog02] Phillip Rogaway. Authenticated-encryption with associated-data. In Vijayalakshmi Atluri, editor, *ACM CCS 2002*, pages 98–107. ACM Press, November 2002. doi:10.1145/586110.586125.

[Rog04] Phillip Rogaway. Efficient instantiations of tweakable blockciphers and refinements to modes OCB and PMAC. In Pil Joong Lee, editor, *ASIACRYPT 2004*, volume 3329 of *LNCS*, pages 16–31. Springer, Heidelberg, December 2004. doi:10.1007/978-3-540-30539-2_2.

[Rog11] Phillip Rogaway, editor. *CRYPTO 2011*, volume 6841 of *LNCS*. Springer, Heidelberg, August 2011.

[RS06] Phillip Rogaway and Thomas Shrimpton. A provable-security treatment of the key-wrap problem. In Serge Vaudenay, editor, *EUROCRYPT 2006*, volume 4004 of *LNCS*, pages 373–390. Springer, Heidelberg, May / June 2006. doi:10.1007/11761679_23.

[RSA78] Ronald L. Rivest, Adi Shamir, and Leonard M. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Communications of the Association for Computing Machinery*, 21(2):120–126, 1978.

[RSM$^+$11] Venkatesh Ramamoorthy, Marius-Calin Silaghi, Toshihiro Matsui, Katsutoshi Hirayama, and Makoto Yokoo. The design of cryptographic s-boxes using csps. In *Principles and Practice of Constraint Programming - CP 2011 - 17th International Conference, CP 2011*, volume 6876 of *Lecture Notes in Computer Science*, pages 54–68. Springer, 2011.

[RW03] P. Rogaway and D. Wagner. A critique of CCM. Cryptology ePrint Archive, Report 2003/070, 2003. http://eprint.iacr.org/2003/070.

[Saa] Markku-Juhani O. Saarinen. Cryptographic Analysis of All 4 x 4 - Bit S-Boxes. Cryptology ePrint Archive, Report 2011/218.

[Saa11] Markku-Juhani O. Saarinen. Cryptographic analysis of all 4 x 4 - bit s-boxes. Cryptology ePrint Archive, Report 2011/218, 2011. http://eprint.iacr.org/2011/218.

[San95] Alfredo De Santis, editor. *EUROCRYPT'94*, volume 950 of *LNCS*. Springer, Heidelberg, May 1995.

[Sar03] P. Sarkar. Hiji-Bij-Bij: A New Stream Cipher with a Self-Synchronizing Mode of Operation, 2003.

[SBM18] Pascal Sasdrich, René Bock, and Amir Moradi. Threshold implementation in software - case study of PRESENT. Cryptology ePrint Archive, Report 2018/189, 2018. https://eprint.iacr.org/2018/189.

[Sch01] Bruce Schneier, editor. *FSE 2000*, volume 1978 of *LNCS*. Springer, Heidelberg, April 2001.

[SGL$^+$17] Siwei Sun, David Gerault, Pascal Lafourcade, Qianqian Yang, Yosuke Todo, Kexin Qiao, and Lei Hu. Analysis of AES, SKINNY, and others with constraint programming. *IACR Trans. Symm. Cryptol.*, 2017(1):281–306, 2017. doi:10.13154/tosc.v2017.i1.281-306.

[Sha49] Claude E. Shannon. Communication theory of secrecy systems. *Bell Systems Technical Journal*, 28(4):656–715, 1949.

[SHW+14] Siwei Sun, Lei Hu, Peng Wang, Kexin Qiao, Xiaoshuang Ma, and Ling Song. Automatic security evaluation and (related-key) differential characteristic search: Application to SIMON, PRESENT, LBlock, DES(L) and other bit-oriented block ciphers. In Sarkar and Iwata [SI14], pages 158–178. doi:10.1007/978-3-662-45611-8_9.

[SI14] Palash Sarkar and Tetsu Iwata, editors. *ASIACRYPT 2014, Part I*, volume 8873 of *LNCS*. Springer, Heidelberg, December 2014.

[SIH+11] Kyoji Shibutani, Takanori Isobe, Harunaga Hiwatari, Atsushi Mitsuda, Toru Akishita, and Taizo Shirai. Piccolo: An ultra-lightweight blockcipher. In Preneel and Takagi [PT11], pages 342–357. doi:10.1007/978-3-642-23951-9_23.

[SK96] Bruce Schneier and John Kelsey. Unbalanced Feistel networks and block cipher design. In Dieter Gollmann, editor, *FSE'96*, volume 1039 of *LNCS*, pages 121–144. Springer, Heidelberg, February 1996. doi:10.1007/3-540-60865-6_49.

[SM10] Tomoyasu Suzaki and Kazuhiko Minematsu. Improving the generalized Feistel. In Hong and Iwata [HI10], pages 19–39. doi:10.1007/978-3-642-13858-4_2.

[SMB18] Sadegh Sadeghi, Tahereh Mohammadi, and Nasour Bagheri. Cryptanalysis of reduced round SKINNY block cipher. *IACR Trans. Symm. Cryptol.*, 2018(3):124–162, 2018. doi:10.13154/tosc.v2018.i3.124-162.

[SMMK13] Tomoyasu Suzaki, Kazuhiko Minematsu, Sumio Morioka, and Eita Kobayashi. TWINE : A lightweight block cipher for multiple platforms. In Lars R. Knudsen and Huapeng Wu, editors, *SAC 2012*, volume 7707 of *LNCS*, pages 339–354. Springer, Heidelberg, August 2013. doi:10.1007/978-3-642-35999-6_22.

[SMT19] H. Sato, M. Mimura, and H. Tanaka. Analysis of division property using milp method for lightweight blockcipher piccolo. In *2019 14th Asia Joint Conference on Information Security (AsiaJCIS)*, pages 48–55, 2019.

[SN12] Hadi Soleimany and Kaisa Nyberg. Zero-correlation linear cryptanalysis of reduced-round LBlock. Cryptology ePrint Archive, Report 2012/570, 2012. http://eprint.iacr.org/2012/570.

[SNC09] Mate Soos, Karsten Nohl, and Claude Castelluccia. Extending SAT solvers to cryptographic problems. In *Theory and Applications of Satisfiability Testing - SAT 2009, 12th International Conference, SAT 2009*, volume 5584 of *Lecture Notes in Computer Science*, pages 244–257. Springer, 2009.

[SPQ03] François-Xavier Standaert, Gilles Piret, and Jean-Jacques Quisquater. Cryptanalysis of block ciphers: A survey. 06 2003.

[SQH19] Ling Song, Xianrui Qin, and Lei Hu. Boomerang connectivity table revisited. *IACR Trans. Symm. Cryptol.*, 2019(1):118–141, 2019. doi:10.13154/tosc.v2019.i1.118-141.

[SS16] Peter Schwabe and Ko Stoffelen. All the AES you need on Cortex-M3 and M4. In Avanzi and Heys [AH16], pages 180–194. doi:10.1007/978-3-319-69453-5_10.

[SSA+07] Taizo Shirai, Kyoji Shibutani, Toru Akishita, Shiho Moriai, and Tetsu Iwata. The 128-bit blockcipher CLEFIA (extended abstract). In Alex Biryukov, editor, *FSE 2007*, volume 4593 of *LNCS*, pages 181–195. Springer, Heidelberg, March 2007. doi:10.1007/978-3-540-74619-5_12.

[SSD+18] Danping Shi, Siwei Sun, Patrick Derbez, Yosuke Todo, Bing Sun, and Lei Hu. Programming the Demirci-Selçuk meet-in-the-middle attack with constraints. In Thomas Peyrin and Steven Galbraith, editors, *ASIACRYPT 2018, Part II*, volume 11273 of *LNCS*, pages 3–34. Springer, Heidelberg, December 2018. doi:10.1007/978-3-030-03329-3_1.

[ST16] Yu Sasaki and Yosuke Todo. New differential bounds and division property of Lilliput: Block cipher with extended generalized Feistel network. In Avanzi and Heys [AH16], pages 264–283. doi:10.1007/978-3-319-69453-5_15.

[ST17] Yu Sasaki and Yosuke Todo. New impossible differential search tool from design and cryptanalysis aspects - revealing structural properties of several ciphers. In Jean-Sébastien Coron and Jesper Buus Nielsen, editors, *EUROCRYPT 2017, Part III*, volume 10212 of *LNCS*, pages 185–215. Springer, Heidelberg, April / May 2017. doi:10.1007/978-3-319-56617-7_7.

[ST18] Yu Sasaki and Yosuke Todo. Tight bounds of differentially and linearly active s-boxes and division property of lilliput. *IEEE Trans. Computers*, 67(5):717–732, 2018.

[SUP] SUPERCOP. https://bench.cr.yp.to/supercop.html. Accessed: 2019-02-21.

[SW13] Yu Sasaki and Lei Wang. Comprehensive study of integral analysis on 22-round LBlock. In Kwon et al. [KLK13], pages 156–169. doi:10.1007/978-3-642-37682-5_12.

[SWLW16] Ling Sun, Wei Wang, Ru Liu, and Meiqin Wang. MILP-aided bit-based division property for ARX-based block cipher. Cryptology ePrint Archive, Report 2016/1101, 2016. http://eprint.iacr.org/2016/1101.

[SWW17] Ling Sun, Wei Wang, and Meiqin Wang. Automatic search of bit-based division property for ARX ciphers and word-based division property. In Tsuyoshi Takagi and Thomas Peyrin, editors, *ASIACRYPT 2017, Part I*, volume 10624 of *LNCS*, pages 128–157. Springer, Heidelberg, December 2017. doi:10.1007/978-3-319-70694-8_5.

[SWW18] Ling Sun, Wei Wang, and Meiqin Wang(66). More accurate differential properties of LED64 and Midori64. *IACR Trans. Symm. Cryptol.*, 2018(3):93–123, 2018. doi:10.13154/tosc.v2018.i3.93-123.

[TAY17] Mohamed Tolba, Ahmed Abdelkhalek, and Amr M. Youssef. Impossible differential cryptanalysis of reduced-round SKINNY. In Marc Joye and Abderrahmane Nitaj, editors, *AFRICACRYPT 17*, volume 10239 of *LNCS*, pages 117–134. Springer, Heidelberg, May 2017.

[TG92] Anne Tardy-Corfdir and Henri Gilbert. A known plaintext attack of FEAL-4 and FEAL-6. In Joan Feigenbaum, editor, *CRYPTO'91*, volume 576 of *LNCS*, pages 172–181. Springer, Heidelberg, August 1992. doi:10.1007/3-540-46766-1_12.

[Tho15] Gaël Thomas. *Design and Security Analysis for constructions in symmetric cryptography*. Theses, Université de Limoges, June 2015. URL: https://tel.archives-ouvertes.fr/tel-01184927.

[TLS16] Yosuke Todo, Gregor Leander, and Yu Sasaki. Nonlinear invariant attack - practical attack on full SCREAM, iSCREAM, and Midori64. In Jung Hee Cheon and Tsuyoshi Takagi, editors, *ASIACRYPT 2016, Part II*, volume 10032 of *LNCS*, pages 3–33. Springer, Heidelberg, December 2016. doi:10.1007/978-3-662-53890-6_1.

[TM16] Yosuke Todo and Masakatu Morii. Bit-based division property and application to simon family. In Peyrin [Pey16], pages 357–377. doi:10.1007/978-3-662-52993-5_18.

[Tod15a] Yosuke Todo. Integral cryptanalysis on full MISTY1. In Gennaro and Robshaw [GR15], pages 413–432. doi:10.1007/978-3-662-47989-6_20.

[Tod15b] Yosuke Todo. Structural evaluation by generalized integral property. In Oswald and Fischlin [OF15], pages 287–314. doi:10.1007/978-3-662-46800-5_12.

[TW15] Biaoshuai Tao and Hongjun Wu. Improving the biclique cryptanalysis of AES. In Ernest Foo and Douglas Stebila, editors, *ACISP 15*, volume 9144 of *LNCS*, pages 39–56. Springer, Heidelberg, June / July 2015. doi:10.1007/978-3-319-19962-7_3.

[UDI$^+$11] Markus Ullrich, Christophe Decannière, Sebastiaan Indesteege, Ozgül Küçük, Nicky Mouha, and Bart Preneel. Finding Optimal Bitsliced Implementations of 4x4-bit S-boxes. Feb 2011.

[UMHA16] Rei Ueno, Sumio Morioka, Naofumi Homma, and Takafumi Aoki. A high throughput/gate AES hardware architecture by compressing encryption and decryption datapaths - toward efficient CBC-mode implementation. In Gierlichs and Poschmann [GP16], pages 538–558. doi:10.1007/978-3-662-53140-2_26.

[Vau98] Serge Vaudenay. Provable security for block ciphers by decorrelation. In Michel Morvan, Christoph Meinel, and Daniel Krob, editors, *STACS 98, 15th Annual Symposium on Theoretical Aspects of Computer Science, Paris, France, February 25-27, 1998, Proceedings*, volume 1373 of *Lecture Notes in Computer Science*, pages 249–275. Springer, 1998. URL: https://doi.org/10.1007/BFb0028566, doi:10.1007/BFb0028566.

[Vau02] Serge Vaudenay. Security flaws induced by CBC padding - applications to SSL, IPSEC, WTLS... In Lars R. Knudsen, editor, *EUROCRYPT 2002*, volume 2332 of *LNCS*, pages 534–546. Springer, Heidelberg, April / May 2002. doi:10.1007/3-540-46035-7_35.

[Wag99] David Wagner. The boomerang attack. In Knudsen [Knu99], pages 156–170. doi:10.1007/3-540-48519-8_12.

[Wan08] Meiqin Wang. Differential cryptanalysis of reduced-round PRESENT. In Serge Vaudenay, editor, *AFRICACRYPT 08*, volume 5023 of *LNCS*, pages 40–49. Springer, Heidelberg, June 2008.

[WHF03] D. Whiting, R. Housley, and N. Ferguson. Counter with CBC-MAC (CCM). RFC 3610 (Informational), September 2003. URL: http://www.ietf.org/rfc/rfc3610.txt.

[WN95] David J. Wheeler and Roger M. Needham. TEA, a tiny encryption algorithm. In Preneel [Pre95], pages 363–366. doi:10.1007/3-540-60590-8_29.

[WP19] Haoyang Wang and Thomas Peyrin. Boomerang switch in multiple rounds. *IACR Trans. Symm. Cryptol.*, 2019(1):142–169, 2019. doi:10.13154/tosc.v2019.i1.142-169.

[WSTP12] Meiqin Wang, Yue Sun, Elmar Tischhauser, and Bart Preneel. A model for structure attacks, with applications to PRESENT and Serpent. In Anne Canteaut, editor, *FSE 2012*, volume 7549 of *LNCS*, pages 49–68. Springer, Heidelberg, March 2012. doi:10.1007/978-3-642-34047-5_4.

[Wu08] Hongjun Wu. The stream cipher HC-128. In Matthew J. B. Robshaw and Olivier Billet, editors, *New Stream Cipher Designs - The eSTREAM Finalists*, volume 4986 of *Lecture Notes in Computer Science*, pages 39–47. Springer, 2008. URL: https://doi.org/10.1007/978-3-540-68351-3_4, doi:10.1007/978-3-540-68351-3_4.

[Wu16] Hongjun Wu. Acorn: a lightweight authenticated cipher (v3). *Candidate for the CAESAR Competition. See also https://competitions.cr.yp.to/round3/acornv3.pdf*, 2016.

[WW14] Yanfeng Wang and Wenling Wu. Improved multidimensional zero-correlation linear cryptanalysis and applications to LBlock and TWINE. In Willy Susilo and Yi Mu, editors, *ACISP 14*, volume 8544 of *LNCS*, pages 1–16. Springer, Heidelberg, July 2014. doi:10.1007/978-3-319-08344-5_1.

[WWJ16] Ning Wang, Xiaoyun Wang, and Keting Jia. Improved impossible differential attack on reduced-round LBlock. In Soonhak Kwon and Aaram Yun, editors, *ICISC 15*, volume 9558 of *LNCS*, pages 136–152. Springer, Heidelberg, November 2016. doi:10.1007/978-3-319-30840-1_9.

[WYWP18] Yongzhuang Wei, Tao Ye, Wenling Wu, and Enes Pasalic. Generalized nonlinear invariant attack and a new design criterion for round constants. *IACR Trans. Symm. Cryptol.*, 2018(4):62–79, 2018. doi:10.13154/tosc.v2018.i4.62-79.

[WZ11a] Wenling Wu and Lei Zhang. LBlock: A lightweight block cipher. In Javier Lopez and Gene Tsudik, editors, *ACNS 11*, volume 6715 of *LNCS*, pages 327–344. Springer, Heidelberg, June 2011. doi:10.1007/978-3-642-21554-4_19.

[WZ11b] Wenling Wu and Lei Zhang. LBlock: A lightweight block cipher *. Cryptology ePrint Archive, Report 2011/345, 2011. http://eprint.iacr.org/2011/345.

[XJHL15] Hong Xu, Ping Jia, Geshi Huang, and Xuejia Lai. Multidimensional zero-correlation linear cryptanalysis on 23-round LBlock-s. In Sihan Qing, Eiji Okamoto, Kwangjo Kim, and Dongmei Liu, editors, *ICICS 15*, volume 9543 of *LNCS*, pages 97–108. Springer, Heidelberg, December 2015. doi:10.1007/978-3-319-29814-6_9.

[XZBL16] Zejun Xiang, Wentao Zhang, Zhenzhen Bao, and Dongdai Lin. Applying MILP method to searching integral distinguishers based on division property for 6 lightweight block ciphers. In Cheon and Takagi [CT16], pages 648–678. doi:10.1007/978-3-662-53887-6_24.

[YI13] S. Yanagihara and T. Iwata. Improving the Permutation Layer of Type 1, Type 3, Source-Heavy, and Target-Heavy Generalized Feistel Structures. *IEICE Trans.*, 96-A(1):2–14, 2013.

[YWQ09] Lin Yang, Meiqin Wang, and Siyuan Qiao. Side channel cube attack on PRESENT. In Garay et al. [GMO09], pages 379–391.

[ZCGP19] Wenying Zhang, Meichun Cao, Jian Guo, and Enes Pasalic. Improved security evaluation of SPN block ciphers and its applications in the single-key attack on SKINNY. *IACR Trans. Symm. Cryptol.*, 2019(4):171–191, 2019. doi:10.13154/tosc.v2019.i4.171-191.

[ZDJ19] Boxin Zhao, Xiaoyang Dong, and Keting Jia. New related-tweakey boomerang and rectangle attacks on deoxys-bc including BDT effect. *IACR Trans. Symm. Cryptol.*, 2019(3):121–151, 2019. doi:10.13154/tosc.v2019.i3.121-151.

[ZDM$^+$19] Boxin Zhao, Xiaoyang Dong, Willi Meier, Keting Jia, and Gaoli Wang. Generalized related-key rectangle attacks on block ciphers with linear key schedule: Applications to SKINNY and GIFT. Cryptology ePrint Archive, Report 2019/714, 2019. https://eprint.iacr.org/2019/714.

[ZDW19] Rui Zong, Xiaoyang Dong, and Xiaoyun Wang. Related-tweakey impossible differential attack on reduced-round deoxys-bc-256. *Science China Information Sciences*, 62(3):32102, 2019.

[ZMI90] Yuliang Zheng, Tsutomu Matsumoto, and Hideki Imai. On the construction of block ciphers provably secure and not relying on any unproved hypotheses. In Gilles Brassard, editor, *CRYPTO'89*, volume 435 of *LNCS*, pages 461–480. Springer, Heidelberg, August 1990. doi:10.1007/0-387-34805-0_42.

[ZNS12] Liang Zhao, Takashi Nishide, and Kouichi Sakurai. Differential fault analysis of full LBlock. In Werner Schindler and Sorin A. Huss, editors, *COSADE 2012*, volume 7275 of *LNCS*, pages 135–150. Springer, Heidelberg, May 2012. doi:10.1007/978-3-642-29912-4_11.

[ZR17] Wenying Zhang and Vincent Rijmen. Division cryptanalysis of block ciphers with a binary diffusion layer. Cryptology ePrint Archive, Report 2017/188, 2017. http://eprint.iacr.org/2017/188.

[ZSW$^+$12] Wentao Zhang, Bozhan Su, Wenling Wu, Dengguo Feng, and Chuankun Wu. Extending higher-order integral: An efficient unified algorithm of constructing integral distinguishers for block ciphers. In Feng Bao, Pierangela Samarati, and Jianying Zhou, editors, *ACNS 12*, volume 7341 of *LNCS*, pages 117–134. Springer, Heidelberg, June 2012. doi:10.1007/978-3-642-31284-7_8.

[ZW15] Huiling Zhang and Wenling Wu. Structural evaluation for generalized Feistel structures and applications to LBlock and TWINE. In Alex Biryukov and Vipul Goyal, editors, *INDOCRYPT 2015*, volume 9462 of *LNCS*, pages 218–237. Springer, Heidelberg, December 2015. doi:10.1007/978-3-319-26617-6_12.

[ZWW$^+$14] Lei Zhang, Wenling Wu, Yanfeng Wang, Shengbao Wu, and Jian Zhang. LAC: a lightweight authenticated encryption cipher, March 2014. Submission to CAESAR. Available from http://competitions.cr.yp.to/round1/lacv1.pdf.

Résumé

Les travaux présentés dans cette thèse s'inscrivent dans le cadre du projet FUI PACLIDO qui a pour but de définir de nouveaux protocoles et algorithmes de sécurité pour l'Internet des Objets, et plus particulièrement les réseaux de capteurs sans fil. Cette thèse s'intéresse donc aux algorithmes de chiffrements authentifiés dits à *bas coût* ou également, *légers*, pouvant être implémentés sur des systèmes très limités en ressources. Une première partie des contributions porte sur la conception de l'algorithme léger LILLIPUT-AE, basé sur un schéma de Feistel généralisé étendu (EGFN) et soumis au projet de standardisation international *Lightweight Cryptography* (LWC) organisé par le NIST (National Institute of Standards and Technology). Une autre partie des travaux se concentre sur des attaques théoriques menées contre des solutions déjà existantes, notamment un certain nombre de candidats à la compétition LWC du NIST. Elle présente donc des analyses spécifiques des algorithmes SKINNY et SPOOK ainsi qu'une étude plus générale des attaques de type boomerang contre les schémas de Feistel.

Mots-clés: cryptographie, cryptanalyse, bas coût, authentification, chiffrement, Feistel, Internet des Objets

Abstract

The work presented in this thesis has been completed as part of the FUI PACLIDO project, whose aim is to provide new security protocols and algorithms for the Internet of Things, and more specifically wireless sensor networks. As a result, this thesis investigates so-called *lightweight* authenticated encryption algorithms, which are designed to fit into the limited resources of constrained environments. The first main contribution focuses on the design of a lightweight cipher called LILLIPUT-AE, which is based on the extended generalized Feistel network (EGFN) structure and was submitted to the *Lightweight Cryptography* (LWC) standardization project initiated by NIST (National Institute of Standards and Technology). Another part of the work concerns theoretical attacks against existing solutions, including some candidates of the NIST LWC standardization process. Therefore, some specific analyses of the SKINNY and SPOOK algorithms are presented, along with a more general study of boomerang attacks against ciphers following a Feistel construction.

Keywords: cryptography, cryptanalysis, lightweight, authentication, encryption, Feistel, Internet of Things.