Cryptanalysis of symmetric encryption algorithms

Colin Chaigneau

▶ **To cite this version:**

Colin Chaigneau. Cryptanalysis of symmetric encryption algorithms. Cryptography and Security [cs.CR]. Université Paris Saclay (COmUE), 2018. English. NNT : 2018SACLV086 . tel-02012149

HAL Id: tel-02012149
https://theses.hal.science/tel-02012149

Submitted on 8 Feb 2019

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

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

universite PARIS-SACLAY

UNIVERSITÉ DE VERSAILLES ST-QUENTIN-EN-YVELINES

université PARIS-SACLAY

NNT : 2018SACLV086

Cryptanalyse des algorithmes de chiffrement symétrique

Thèse de doctorat de l'Université Paris-Saclay
préparée à l'Université de Versailles Saint-Quentin-en-Yvelines

Ecole doctorale n°580 Sciences et Technologies de l'Information et de la Communication (STIC)
Spécialité de doctorat : Mathèmatiques et Informatique

Thèse présentée et soutenue à Versailles, le mercredi 28 novembre 2018, par

Colin Chaigneau

Composition du Jury :

M. Louis GOUBIN Professeur, Université de Versailles Saint-Quentin-en-Yvelines, France	Président
M. Thierry BERGER Professeur Émérite, Université de Limoges, France	Rapporteur
Mme. María NAYA-PLASENCIA Directrice de Recherche, INRIA Paris, France	Rapporteure
M. Patrick DERBEZ Maître de Conférence, Université de Rennes 1, France	Examinateur
Mme. Marine MINIER Professeure, Université de Lorraine, France	Examinatrice
M. Gilles VAN ASSCHE Senior Principal Cryptographer, STMicroelectronics, Belgique	Examinateur
M. Henri GILBERT Expert, ANSSI - Chercheur Associé, Université de Versailles Saint-Quentin-en-Yvelines, France	Directeur de thèse

Thèse de doctorat

ÉCOLE DOCTORALE

Sciences et technologies de l'information et de la communication (STIC)

Titre : Cryptanalyse des algorithmes de chiffrement symétrique

Mots clés : Cryptographie symétrique, cryptanalyse, chiffrement par bloc, chiffrement à flot, chiffrement authentifié

Résumé : La sécurité des transmissions et du stockage des données est devenue un enjeu majeur de ces dernières années et la cryptologie, qui traite de la protection algorithmique de l'information, est un sujet de recherche extrêmement actif. Elle englobe la conception d'algorithmes cryptographiques, appelée cryptographie, et l'analyse de leur sécurité, appelée cryptanalyse.

Dans cette thèse, nous nous concentrons uniquement sur la cryptanalyse, et en particulier celle des algorithmes de chiffrement symétrique, qui reposent sur le partage d'un même secret entre l'entité qui chiffre l'information et celle qui la déchiffre. Dans ce manuscrit, trois attaques contre des algorithmes de chiffrement symétriques sont présentées. Les deux premières portent sur deux candidats de l'actuelle compétition cryptographique CAESAR, les algorithmes AEZ et NORX, tandis que la dernière porte sur l'algorithme Kravatte, une instance de la construction Farfalle qui utilise la permutation de la fonction de hachage décrite dans le standard SHA-3. Les trois algorithmes étudiés présentent une stratégie de conception similaire, qui consiste à intégrer dans une construction nouvelle une primitive, i.e. une fonction cryptographique élémentaire, déjà existante ou directement inspirée de travaux précédents.

La compétition CAESAR, qui a débuté en 2015, a pour but de définir un portefeuille d'algorithmes recommandés pour le chiffrement authentifié. Les deux candidats étudiés, AEZ et NORX, sont deux algorithmes qui ont atteint le troisième tour de cette compétition. Les deux attaques présentées ici ont contribué à l'effort de cryptanalyse nécessaire dans une telle compétition. Cet effort n'a, en l'occurrence, pas permis d'établir une confiance suffisante pour justifier la présence des algorithmes AEZ et NORX parmi les finalistes.

AEZ est une construction reposant sur la primitive AES, dont l'un des principaux objectifs est d'offrir une résistance optimale à des scénarios d'attaque plus permissifs que ceux généralement considérés pour les algorithmes de chiffrement authentifié. Nous montrons ici que dans de tels scénarios il est possible, avec une probabilité anormalement élevée, de retrouver l'ensemble des secrets utilisés dans l'algorithme.

NORX est un algorithme de chiffrement authentifié qui repose sur une variante de la construction dite en éponge employée par exemple dans la fonction de hachage Keccak. Sa permutation interne est inspirée de celles utilisées dans BLAKE et ChaCha. Nous montrons qu'il est possible d'exploiter une propriété structurelle de cette permutation afin de récupérer la clé secrète utilisée. Pour cela, nous tirons parti du choix des concepteurs de réduire les marges de sécurité dans le dimensionnement de la construction en éponge.

Enfin, la dernière cryptanalyse remet en cause la robustesse de l'algorithme Kravatte, une fonction pseudo-aléatoire qui autorise des entrées et sorties de taille variable. Dérivée de la permutation Keccak-p de SHA-3 au moyen de la construction Farfalle, Kravatte est efficace et parallélisable. Ici, nous exploitons le faible degré algébrique de la permutation interne pour mettre au jour trois attaques par recouvrement de clé : une attaque différentielle d'ordre supérieur, une attaque algébrique "par le milieu" et une attaque inspirée de la cryptanalyse de certains algorithmes de chiffrement à flot.

Title : Cryptanalysis of symmetric encryption algorithms

Keywords : Symmetric cryptography, cryptanalysis, block cipher, stream cipher, authenticated encryption

Abstract : Nowadays, cryptology is heavily used to protect stored and transmitted data against malicious attacks, by means of security algorithms. Cryptology comprises cryptography, the design of these algorithms, and cryptanalysis, the analysis of their security.

In this thesis, we focus on the cryptanalysis of symmetric encryption algorithms, that is cryptographic algorithms that rely on a secret value shared beforehand between two parties to ensure both encryption and decryption. We present three attacks against symmetric encryption algorithms. The first two cryptanalyses target two high profile candidates of the CAESAR cryptographic competition, the AEZ and NORX algorithms, while the last one targets the Kravatte algorithm, an instance of the Farfalle construction based on the Keccak permutation. Farfalle is multipurpose a pseudo-random function (PRF) developed by the same designers' team as the permutation Keccak used in the SHA-3 hash function.

The CAESAR competition, that began in 2015, aims at selecting a portfolio of algorithms recommended for authenticated encryption. The two candidates analysed, AEZ and NORX, reached the third round of the CAESAR competition but were not selected to be part of the finalists. These two results contributed to the cryptanalysis effort required in such a competition. This effort did not establish enough confidence to justify that AEZ and NORX accede to the final round of the competition. AEZ is a construction based on the AES primitive, that aims at offering an optimal resistance against more permissive attack scenarios than those usually considered for authenticated encryption algorithms. We show here that one can recover all the secret material used in AEZ with an abnormal success probability. NORX is an authenticated encryption algorithm based on a variant of the so-called sponge construction used for instance in the SHA-3 hash function. The internal permutation is inspired from the one of BLAKE and ChaCha. We show that one can leverage a strong structural property of this permutation to recover the secret key, thanks to the designers' non-conservative choice of reducing the security margin in the sponge construction.

Finally, the last cryptanalysis reconsiders the robustness of the Kravatte algorithm. Kravatte is an efficient and parallelizable PRF with input and output of variable length. In this analysis, we exploit the low algebraic degree of the permutation Keccak used in Kravatte to mount three key-recovery attacks targeting different parts of the construction: a higher order differential attack, an algebraic meet-in-the-middle attack and an attack based on a linear recurrence distinguisher.

Université Paris-Saclay
Espace Technologique / Immeuble Discovery
Route de l'Orme aux Merisiers RD 128 / 91190 Saint-Aubin, France

REMERCIEMENTS

Ce sont sans doute les premières pages que liront beaucoup de personnes, mais elles sont surtout les dernières auxquelles je me serais consacré durant cette thèse, complétant ainsi trois années de belles aventures. Et quel chemin parcouru depuis les premiers pas foulés sur le sol parisien. J'ai eu l'occasion de rencontrer beaucoup de personnes intéressantes, à tous je vous remercie. Je m'excuse par avance pour les mots que je n'arriverai pas forcément à trouver pour exprimer toute la gratitude que je pourrais avoir pour vous, leur absence n'entame en rien ce que j'éprouve. Si par malchance votre nom serait omis et que vous arriveriez à prouver que cette absence n'est pas justifié je m'engage alors à me faire pardonner en vous offrant une boîte de mes délicieux cookies aux kinder maxi (ou alors tout simplement si vous en éprouvez l'envie et la gourmandise, mais il faudra alors me persuader que vous les méritez).

Je tiens tout d'abord à remercier mon directeur de thèse, Henri Gilbert, sans qui tout cela n'aurait pas été possible. Henri merci pour ta gentillesse, ta patience et ta sagesse durant ces trois années, ces qualités s'ajoutant aussi à l'extrême justesse dont tu fais preuve dans tes raisonnement. J'ai beaucoup appris à tes côtés et j'espère en avoir encore l'occasion, pour cela je te suis reconnaissant. Merci à toi.

J'aimerais ensuite remercier María Naya-Plasencia et Thierry Berger d'avoir accepté d'être mes rapporteurs.
Merci aussi aux membres de mon jury, Marine Minier, Louis Goubin, Patrick Derbez et Gilles Van Assche, pour avoir accepter d'être présent pour ma soutenance.

Je remercie le projet BRUTUS pour avoir financé ma thèse ainsi que tous les collaborateurs que j'ai eu l'occasion de rencontrer.
Je tiens aussi à remercier Thierry Berger, qui en plus d'avoir été mon professeur de cryptographie symétrique m'a aussi aiguillé vers les cryptographes de la région parisienne.

Merci à tous les membres de l'ANSSI, avec qui j'ai passé de très bon moments durant ces trois années et demie (depuis le début de mon stage à l'ANSSI), et ce n'est pas sans une certaine tristesse que je fermerais l'épicerie de mon bureau. Merci donc Henri, Thomas, Jean-René, Jérémy, Aurélie,

Guénaël, Jérôme, Yannick, Jean-Pierre, Mélissa, Adrian, Guillaume, Ryad, Boris, Emmanuel, Thomas, Louiza, David, Karim, Éliane, Christophe, José, Valentin, Philippe, Pierre-Michel. Vous contribuez tous à cette bonne ambiance qui caractérise si bien l'ANSSI (et ce malgré les innombrables moqueries que j'ai dû subir durant toutes ses années...).
Je remercie aussi en particulier mes co-auteurs, Henri, Thomas, Jérémy, Jean-René, avec qui j'ai eu l'occasion de collaborer et d'apprendre beaucoup.

Merci à l'équipe CRYPTO de m'avoir accueilli durant ma thèse : Louis, Christina, Ilaria, Luca, Michael, Valentin, Alexandre, Florent, Édouard, Axel et Élise tout récemment.
Louis merci pour les conseils, les discussions et l'aide apportée pour se défaire des démarches administratives. Merci Christina pour les conseils et les discussions que l'ont a pu avoir. Ilaria merci pour toutes ces conversations, verres et autres bon moments, bonne chance au pays des frites et du chocolat.

Je remercie aussi ceux qui me font confiance dans l'enseignement : Louis, Michael, Christina, Sandrine et Franck. J'ai découvert le plaisir d'enseigner avec ces TD. Merci aussi à tous les étudiants que j'ai pu avoir, j'en apprend encore tous les jours.
Merci à tous les membres du LMV, Christophe, Catherine, Nadège et ceux du département d'informatique, Sandrine, Franck, Thierry, Sébastien et Marie avec qui j'ai eu l'occasion de discuter et partager.
Merci aux doctorants de math avec qui j'ai eu l'occasion de passer quelques bons moments : Patricio, Antoine, Maxime, Camilla, Sibylle et Salim.

Merci aussi à tous ceux que j'ai pu rencontrer sur la fac et avec qui j'ai partagé de très bon moments : Alicia, Amélie, Angélo, Axelle, Axelle, Béni, Cassandre, Cylia, Feriel, Florian, Jérémy, Lucas, Lucie, Marius, Mathieu, Nicolas, Rémi, Ségolène, Théoo, Vincent, Zachary. J'espère avoir l'occasion de passer encore de bons moments.

Je remercie ensuite tout les doctorants et post-doc crypto que j'ai eu l'occasion de rencontrer : Anca, Dahmoun, Romain, Alain, Yann, Sébastien, Virginie, Léo, Isabella, Sarah, Lucas, Mathilde, André, Ferdinand, Julien, Baptiste. Un merci aussi à tout les cryptographes un peu plus vieux (j'ai dit un peu plus vieux) : Anne, Gaëtan, María, Pierre, Brice, Thomas, Patrick, Marine, Pierre-Alain, Mathieu. La communauté cryptographique est, je trouve, une belle famille.

Je remercie aussi chaleureusement les camarades du master CRYPTIS dont un certain nombre ont finalement rejoint Paris et ses environs : Élise et Tom

(dans cet ordre là), Nico, Anthony, Zoé, Adrien, Chloé, Maël et Paul. Une pensée particulière à Maël et Paul avec qui j'ai partagé plusieurs années d'université et que j'espère pouvoir revoir bientôt.

Puisqu'il est question des études, je tiens aussi à remercier tout les enseignants de l'Université de Limoges dont j'ai eu la chance d'assister aux cours, certains m'ont conforté dans les choix que je faisais, d'autres m'ont aiguillé, tous ont en tout cas participé à ce que je suis maintenant et je les en remercie.
Je remercie aussi les professeurs de math du lycée Bernard Palissy pour avoir entretenu cette attraction pour les mathématiques.

Si beaucoup de rencontres durant la thèse furent le fruit de cette passion pour la cryptographie, beaucoup d'autres en sont le résultat d'autres passions communes.
Merci donc aux membres de la B.A.B., Julie, Julie, Elisa et Emy. Qui aurait cru que cette soirée ratée il y a maintenant un peu plus de deux ans aurait aboutie à cette amitié. Merci aussi à Franky pour toutes ces soirées et ces bons moments (l'avenir approche !). Merci aussi à Ibrahim, Camille, Claire, Cynthia, Jessica, Manon, Marine, Julie, Alizée. Merci aussi à Magali pour l'organisation de ces soirées à la librairie. Seuls des moldus ne pourraient comprendre les raisons de cette passion.
J'ai aussi eu l'occasion de rencontrer des trolls plutôt affectueux et qui égayent maintenant mes mercredis soirs, merci donc à Arthur, Aurel, Arnaud, Guillaume, Sébastien et Jessica.
Merci aussi à tout ceux que je ne vois pas forcément souvent : Guix, Jean, Sarah, Nicolas, Xavier, Agathe.
Merci en particulier à Julie que je connais maintenant depuis quelques années, merci pour ces FaceTime épiques et tout ces délires, reste comme tu es.

Je termine enfin ces remerciements par les personnes les plus importantes. Même si je vis maintenant un peu loin d'eux, j'ai une pensée particulière pour toute ma famille. Je vous aime beaucoup et vous remercie pour tout ce que vous avez fait pour moi, et tout ce que vous m'avez apporté.

PUBLICATIONS

[CG16] Colin Chaigneau and Henri Gilbert.
Is AEZ v4.1 Sufficiently Resilient Against Key-Recovery Attacks?
IACR Transactions on Symmetric Cryptology, December 2016.

[CFG+17] Colin Chaigneau, Thomas Fuhr, Henri Gilbert, Jérémy Jean, and Jean-René Reinhard.
Cryptanalysis of NORX v2.0.
IACR Transactions on Symmetric Cryptology, March 2017.

[CFG+18a] Colin Chaigneau, Thomas Fuhr, Henri Gilbert, Jian Guo, Jérémy Jean, Jean-René Reinhard, and Ling Song.
Key-Recovery Attacks on Full Kravatte.
IACR Transactions on Symmetric Cryptology, March 2018.

[CFG+18b] Colin Chaigneau, Thomas Fuhr, Henri Gilbert, Jérémy Jean, and Jean-René Reinhard.
Cryptanalysis of NORX v2.0.
Journal of Cryptology, 2018. To appear.
https://www.springerprofessional.de/en/cryptanalysis-of-norx-v2-0/15826164.

CONTENTS

1 INTRODUCTION 1
 1.1 What is Cryptography? 2
 1.2 Symmetric Encryption Algorithms 3
  1.2.1 Basic Security Notions 5
  1.2.2 Symmetric Primitives 6
  1.2.3 Primitives are Used in Larger Constructions 10
 1.3 Beyond Confidentiality, Authenticity 15
  1.3.1 Message Authentication 16
  1.3.2 Formalisation 16
  1.3.3 Security of AE Schemes 17
  1.3.4 Constructions and Modes of Operations to Achieve Authentication 18
 1.4 How to Build Strong Cryptographic Primitives? 19
  1.4.1 Attack Success 19
  1.4.2 Context of Cryptanalysis 20
  1.4.3 Cryptanalysis Strategies 21
 1.5 Contributions 21
2 CAESAR COMPETITION 25
 2.1 Context and Goals 25
 2.2 Timeline 27
 2.3 Finalists 27
  2.3.1 ACORN v1.2 28
  2.3.2 Ascon v1.2 29
  2.3.3 AEGIS v1.1 30
  2.3.4 MORUS v2 32
  2.3.5 OCB v1.1 33
  2.3.6 COLM v1 35
  2.3.7 Deoxys-II v1.41 36
3 CRYPTANALYSIS OF AEZ 39
 3.1 Description of AEZ 42
  3.1.1 Tweaked Instances of AES4 and AES10 Used in AEZ 43
  3.1.2 AEZ-hash universal hashing 44
  3.1.3 PRF Function 44
  3.1.4 AEZ Core 44
  3.1.5 Tweaks from AEZ v3 45
 3.2 Attacks on AEZ 47
  3.2.1 Birthday Attacks 47
  3.2.2 AES4 Cryptanalysis 53
  3.2.3 Results of Our Attack 62

3.3 Conclusion 63
4 CRYPTANALYSIS OF NORX 65
4.1 Specifications of NORX 68
4.1.1 Description of NORX v2.0 68
4.1.2 Security Claims 72
4.1.3 NORX Variants 72
4.2 Cryptanalysis of NORX v2.0 73
4.2.1 Non-Random Properties of F 74
4.2.2 Ciphertext-Only Forgery of NORX v2.0 Without Padding 76
4.2.3 Forgery Attack Against NORX v2.0 77
4.2.4 Adversarial Model Discussion 78
4.2.5 Key-Recovery Attack Against NORX v2.0 79
4.3 Pseudo-code for the Ciphertext-only Forgery and Key-Recovery Attack 80
4.4 Application to Other Variants of NORX 80
4.5 Discussion About NORX Security Claims 83
5 CRYPTANALYSIS OF KRAVATTE 87
5.1 Specifications of Farfalle and Kravatte 90
5.1.1 The Farfalle Construction for Permutation-Based PRFs 90
5.1.2 The Kravatte Pseudo-Random Function 91
5.1.3 Round Function of the Keccak-p Permutation 93
5.2 Algebraic Cryptanalysis of Full Kravatte 94
5.2.1 Meet-in-the-Middle Algebraic Attack 94
5.2.2 Cancellation of Monomials Using a Linear Recurrence . 97
5.3 Higher Order Differential Cryptanalysis of Full Kravatte . . . 101
5.3.1 Construction of Affine Spaces in the Accumulator 102
5.3.2 Higher Order Differential Attacks Against Kravatte . . 103
5.3.3 Last-Round Attacks 104
5.4 Optimization Techniques for the Cryptanalysis 105
5.4.1 Minimizing the Number of Variables for Two Inverse Rounds 105
5.4.2 Super Structure of Input Messages 108
5.4.3 Counters 110
5.4.4 Optimizing the Attacks 113
5.5 Concluding Remarks and Discussion 115

BIBLIOGRAPHY 117

LIST OF FIGURES

Figure 1 Albus sending encrypted data to Barty. 2
Figure 2 Combination of symmetric and asymmetric encryption to send data. 4
Figure 3 SPN and Feistel networks. 7
Figure 4 Stream Cipher Construction. 10
Figure 5 ECB mode of operation. 13
Figure 6 CBC, CFB, OFB and CTR modes of operation. 14
Figure 7 Sponge construction. 14
Figure 8 monkeyDuplex construction. 15
Figure 9 Generic compositions of an encryption scheme and a MAC. 18
Figure 10 ACORN algorithm - encryption process. 29
Figure 11 Ascon algorithm - encryption session. 31
Figure 12 AEGIS state update (up) and encryption (down) process. 32
Figure 13 MORUS state update (up) and encryption (down) process. 34
Figure 14 OCB encryption process without padding. 35
Figure 15 COLM encryption process without plaintext padding. . . 36
Figure 16 Deoxys-II encryption process with 4 plaintext blocks and 3 AD blocks. Tag generation (up) and ciphertext computation (down). 38
Figure 17 AEZ-core scheme. 46
Figure 18 Difference propagation in the birthday attack to retrieve I. 52
Figure 19 AES4 scheme. 54
Figure 20 Differential path. 55
Figure 21 Bytes numbering in AES state. 55
Figure 22 Difference propagation within AEZ-core. 56
Figure 23 Other possible differential characteristics. 58
Figure 24 NORX v2.0 mode: the padded bit-strings of $12w$-bit blocks $A = A_0||\cdots||A_{a-1}$, $M = M_0||\cdots||M_{m-1}$ and $Z = Z_0||\cdots||Z_{z-1}$ are processed by the monkeyDuplex sponge construction. 70
Figure 25 Function G applies on state columns. 71
Figure 26 Function G applies on state diagonals. 71
Figure 27 Forgery first step: assume the capacity is symmetric (probability 2^{-2w}). 77
Figure 28 Forgery second step: attempt forgery with rotated ciphertext and tag. 77

Figure 29 NORX v3.0 serial mode. 82
Figure 30 The KRAVATTE primitive. The input message M is padded and split into the b-bit blocks m_i. The function $\circlearrowleft n$ refers to the linear function $x \rightarrow roll^n(x)$. 93
Figure 31 Meet-in-the-middle algebraic attack on KRAVATTE, with n_1 forward and n_2 backward rounds, $n_1 + n_2 = n_e$. 95
Figure 32 Linear recurrence in the KRAVATTE branches: the sequence $(\mathbf{y}_i^j)_j$ of highlighted bits at a prescribed Position i across the branches $j = 0, \ldots, \ell_o - 1$ follows a linear recurrence described by the polynomial $(X + 1) \cdot P_{roll}$. 98
Figure 33 Higher order differential distinguisher on KRAVATTE. Summing over the whole affine space $\texttt{Acc}(\mathcal{S})$ the states obtained after application of $\ell = n_d + n_e - \epsilon$ rounds to the blocks X_i of the affine space, i.e., summing along every bold line, yields zero. 103
Figure 34 Notations used in Section 5.4.1. 105
Figure 35 Example of structure/message membership matrix, with $(n, t) = (2, 2)$. 110

LIST OF TABLES

Table 1 Summary of the CAESAR competition finalists 28
Table 2 Main cryptanalyses results on Ascon. 31
Table 3 Main cryptanalyses results on Deoxys. 37
Table 4 AEZ attacks complexities. 41
Table 5 Birthday attacks complexities. 53
Table 6 AES4 attack complexities. 62
Table 7 Full attack complexities. 62
Table 8 Rotation constants in the permutation G. 71
Table 9 Key-recovery attacks against Kravatte instantiations for several (n_d, n_e) values. All attacks are independent of n_b and n_c, and $\star$ means that n_d can take any value. The reference points to the section describing the attack type. The complexity figures are obtained after the selection of optimizations described in Section 5.4. 90
Table 10 Number of monomials in input (resp. output) variables after n rounds of Keccak-p or Keccak-p^{-1} for $b = 1600$ ($\log_2$ scale). 96
Table 11 Degree and computation time of recurrence polynomial for all monomials in $\mathbf{y}$ after n_1 rounds of Keccak-p, and attack complexity against Kravatte-(n_d, n_e), for any n_d and $n_e = n_1 + n_2$. For optimized attacks, see Section 5.4. 102

1 INTRODUCTION

"Paypal notification: you have paid 133.7€ to Steam Marketplace". This notification on the lock screen of your phone has at least two implications. You spent money on games you will probably never use. And by the time this notification reached you, several cryptographic protocols and algorithms have been involved to ensure a secure transaction, from transmission to authentication and payment.

This is an insignificant example of what nowadays, in our ubiquitous world that depends heavily on the Internet, cryptography became: predominant in our daily lives, and vital in many domains. The application spectrum of cryptography is broad but whether it is intended to protect our private information and our communications, or to ensure safe transactions and many other uses, cryptography needs to be fast, reliable and secure. Building a cryptographic system with these qualities is uneasy, but can be achieved with relevant choices and analyses and the knowledge of previous works.

Cryptanalysis is the study by cryptanalysts of cryptographic algorithms from the point of view of an adversary who is trying to break the security of the cryptographic systems. Informally, the longer an algorithm has been analysed the more reliable it will be considered by the cryptographic community. The new proposals can be inspired by previous cryptographic systems which gained trust from the cryptographic community and benefit from the previous security analysis, or they can be built from scratch, offering new design strategies that improve performance, security, or both. In either case, they need to be carefully examined. It is only when confidence, gathered by strong design rationale and cryptanalysis and spread among the cryptographic community that a cryptographic algorithm can start being used in real-world situations. Even if an algorithm does not exhibit critical flaws, the discovery of undesirable properties can justify untrustworthiness. When some flaws are exhibited after a widespread implementation, consequences can be harmful. It can lead to the exploitation of these vulnerabilities by dishonest people and requires quick fixes to prevent these potential attacks. The WEP protocol is a perfect example of what cryptanalysis is dedicated to avoid. While the WEP protocol was broadly adopted in secure transmission, in Wi-Fi networks for instance, an attack[1] focusing on its underlying cryptographic algorithm led to a complete and practical security

1 By Fluhrer et al. [FMS01]

breakdown of this protocol. Hence, cryptanalysis is primordial in the process of designing and adopting new cryptosystems and justifies investing a sufficient amount of time in analysis and studies.

1.1 What is Cryptography?

Cryptology is the study of secure communications, and addresses the question of how to efficiently protect data from untrusted parties. Cryptology encompasses the design, cryptography, and the analysis, cryptanalysis. We depict the fundamentals of cryptography in the first sections and focus on cryptanalysis in Section 1.4.

The process of concealing data is achieved with a so-called *encryption* algorithm, and the reverse operation is performed by a *decryption* algorithm. Both algorithms are parametrized by an *encryption*, resp. a *decryption key*. An encryption algorithm takes as input unprocessed data called *plaintext* and returns a *ciphertext*. The high-level process of encryption is depicted in Figure 1, where Albus wants to send a message to Barty while preventing any third party from deciphering it[2].

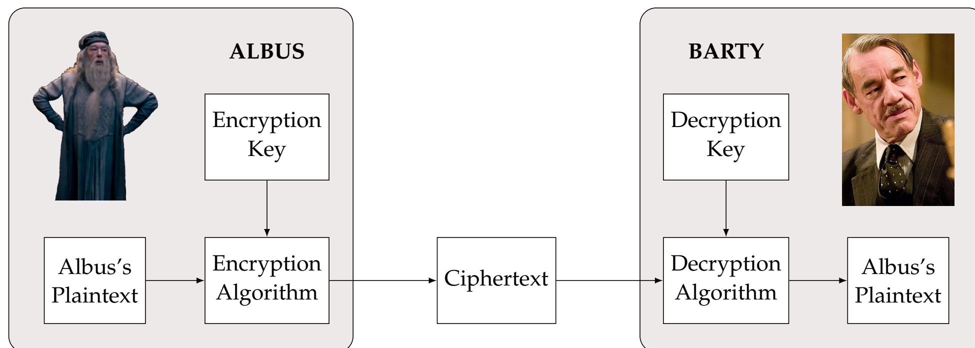

Figure 1: Albus sending encrypted data to Barty.

With this encryption scheme, we can distinguish two algorithm families: those that use the same key for the encryption and the decryption, and those that use two different keys, one for the encryption and another one, impossible to derive from the former one, for the decryption. The family of algorithms involving a unique key to encrypt and decrypt messages belong

2 They use cryptography because the owl postal service is assuredly a non-secure communication channel, which does not prevent malicious people to intercept messages.

to *symmetric* cryptography. When a different key is used for encryption and decryption, the encryption/decryption algorithm belongs to the *asymmetric* cryptography family.

Using a symmetric or an asymmetric algorithm implies profound differences on the design of these algorithms. Asymmetric cryptography is built around hard mathematical problems, e.g. factorization of large numbers, discrete logarithm on finite fields or elliptic curves, lattice problems, etc. Symmetric cryptography, on the other hand, uses basic arithmetic, logical operations and/or look-up tables, and its security is based on Shannon's confusion and diffusion principles [Cla45]. The use of basic operations in symmetric encryption enables efficient software and hardware implementations, but this scheme assumes that the secret key required for encryption and decryption has been securely exchanged beforehand. Asymmetric cryptography is much slower since mathematical objects lay in structures that are more complex to run and implement in software and hardware. But there is no key exchange required in asymmetric cryptography, Albus and Barty both have a public and a private key. To perform encryption, Albus encrypts his plaintext with Barty's public key and sends the computed ciphertext to Barty, who just has to decrypt the ciphertext with his private key. Anyone can encrypt messages and send them to Barty, but he is the only one who knows the private key that can decipher them.

In practice, both cryptographic families are complementary and used in a combined way: asymmetric cryptography ensures that a secret key is safely shared, and symmetric algorithms use this shared key to efficiently protect large amounts of data. An illustration of how the two sides of cryptography are jointly used to achieve secure encryption is depicted in Figure 2.

The following section focuses on symmetric cryptography and provides an overview of this family. Hence, we now exclusively consider symmetric algorithms parametrized by a secret key only shared between two parties.

1.2 Symmetric Encryption Algorithms

Symmetric encryption, as said before, relies on a unique secret key used both for encryption and decryption. Symmetric encryption can be formalized as follows: an encryption algorithm *Enc* takes as input a secret key of length k bits denoted by K and a plaintext string P of p bits. The encryption algorithm returns a ciphertext C of $c \geq p$ bits.

$$\begin{aligned} Enc \ : \ \{0,1\}^k \times \{0,1\}^p &\longrightarrow \quad \{0,1\}^c \\ (K, P) \qquad\quad &\longmapsto C = Enc_K(P). \end{aligned}$$

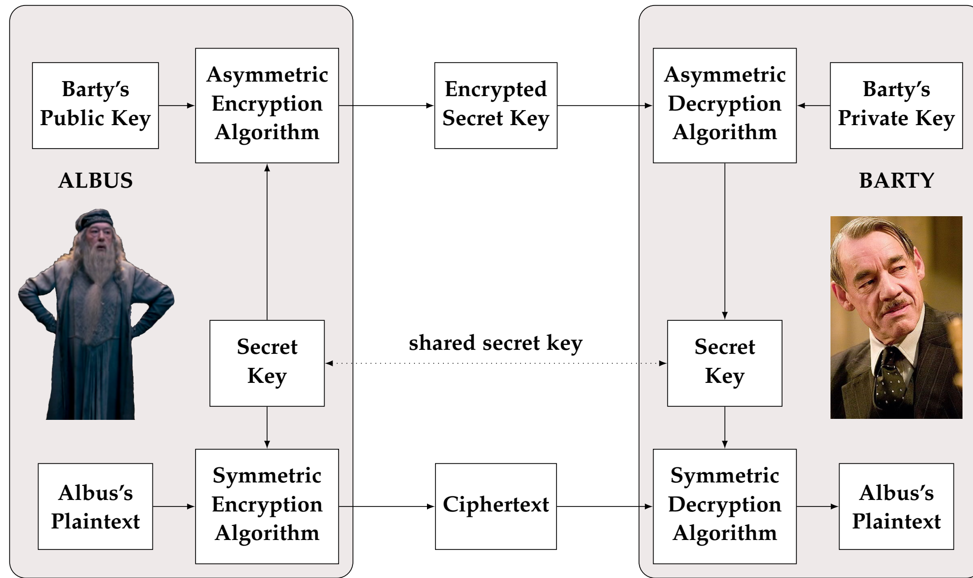

Figure 2: Combination of symmetric and asymmetric encryption to send data.

Its reverse operation *Dec*, for decryption, takes as input a ciphertext and the secret key to recover the initial plaintext,

$$\begin{aligned} Dec \ : \ \{0,1\}^k \times \{0,1\}^c &\longrightarrow \{0,1\}^p \\ (K, C) &\longmapsto P = Dec_K(C), \end{aligned}$$

with Enc and Dec verifying $\forall P,\ \mathrm{Dec}_K(\mathrm{Enc}_K(P)) = P$.

Confidentiality

Since cryptography is used to conceal messages, the main property targeted by a symmetric encryption algorithm is *confidentiality*. Confidentiality means that no unauthorized party can recover secret information on the system and no information can be retrieved from previous encryptions in order to predict the future behaviour of the algorithm. For instance, the knowledge of some previously encrypted plaintext/ciphertext pairs must reveal no information on a ciphertext which could be used to recover any information on its associated plaintext.

To achieve this property, designers ensure that the output of their algorithm is *unpredictable*. The notion of unpredictability is specific to the considered symmetric cryptographic object.

1.2.1 Basic Security Notions

Before introducing the main families of symmetric primitives we formalize three following theoretical objects: a *Pseudo Random Function* (PRF), a *Pseudo Random Permutation* (PRP) and a *Pseudo Random Number Generator* (PRNG). These definitions are useful for capturing the security requirements on symmetric primitives.

Pseudo Random Function - PRF

We denote f_K a function of n bits to m bits parametrized by a secret key K taken from the set $\mathcal{K}$ of all k-bit secret keys under the uniform distribution. We denote f^* a random function from the space of all possible functions of n bits to m bits. We denote A a probabilistic algorithm that given a function f outputs 1 or 0. The goal of the algorithm A is to distinguish a function f_K with a random secret key K from f^*.

We say that the collection $F = \{f_K \mid K \in \mathcal{K}\}$ is a PRF if the advantage of any probabilistic algorithm A for distinguishing f_K, a random instance of F, from f^*, defined by

$$\mathrm{Adv}_F^{PRF}(A) = \left|\Pr[A(f_K) = 1] - \Pr[A(f^*) = 1]\right|$$

is small.

Pseudo Random Permutation - PRP

The definition of a PRP derives naturally from the definition of a PRF. We denote π_K a permutation of n bits parametrized by a secret key K taken from the set $\mathcal{K}$ of all k-bit secret keys under the uniform distribution. We denote π^* a random permutation from the space of all possible permutation of n bits to n bits. We denote A a probabilistic algorithm that given a permutation π outputs 1 or 0. The algorithm A is able to distinguish a random permutation π_K from π^*.

We say that the collection $\Pi = \{\pi_K \mid K \in \mathcal{K}\}$ is a PRP if the advantage of any probabilistic algorithm A for distinguishing π_K, a random instance of Π, from π^*, defined by

$$\mathrm{Adv}_\Pi^{PRP}(A) = \left|\Pr[A(\pi_K) = 1] - \Pr[A(\pi^*) = 1]\right|$$

is small.

Pseudo Random Number Generator - PRNG

We denote s a function that takes as input a secret key K of k bits and outputs a binary string of m bits. We denote s^* a random binary string of m bits. We denote A a probabilistic algorithm that given a binary string outputs 1 or 0. The algorithm A is able to distinguish $s(K)$, with K random, from s^*.

We say that s is a PRNG if the advantage of any probabilistic algorithm A for distinguishing $s(K)$ from s^*, defined by

$$\mathrm{Adv}_s^{PRNG}(A) = \left|\Pr[A(s(K)) = 1] - \Pr[A(s^*) = 1]\right|$$

is small.

1.2.2 Symmetric Primitives

A primitive is a low-level cryptographic algorithm that performs operations on data. Primitives are not encryption algorithms on their own, they are the building blocks of wider constructions.

In symmetric cryptography, we traditionally distinguish two families of primitives, block ciphers and stream ciphers. As their names suggest, the former comprises encryption algorithms that process blocks of data, while the latter comprises encryption algorithms that perform encryption by generating a keystream, i.e. a binary string which is combined with the plaintext to generate the ciphertext. Recently, while block ciphers were widely used as primitives, several cryptographic algorithms were proposed relying on a permutation instead. In the following we provide an overview of these families.

Block Cipher

By definition, a block cipher primitive is an algorithm that performs encryption of blocks. Formally, it can be defined as a family of n bits to n bits permutations parametrized by a secret key. The expected security of a block cipher primitive is the PRP criterion. Thus, we can see a block cipher as a PRP that takes as input a secret key and returns a permutation of n-to-n bits.

The first standardized block cipher algorithm was the Data Encryption Standard (DES) in 1977 [DES77]. Currently, the main standardized block cipher is the *Advanced Encryption Standard* (AES) [AES01], which replaced the DES in 2001. The design of a block cipher relies on a round function parametrized by a round key. The round keys are derived from the secret

key with a key-schedule algorithm. The encryption of a block of data is performed by successive iterations of this round function, with a sufficient number of rounds chosen to guarantee comfortable security.

The structure of the two main types of block cipher designs, *Substitution Permutation Networks* (SPN) and *Feistel networks*, is depicted in Figure 3. SPN make use of a non-linear substitution layer and a linear diffusion layer while Feistel networks rely on a single non-linear function.

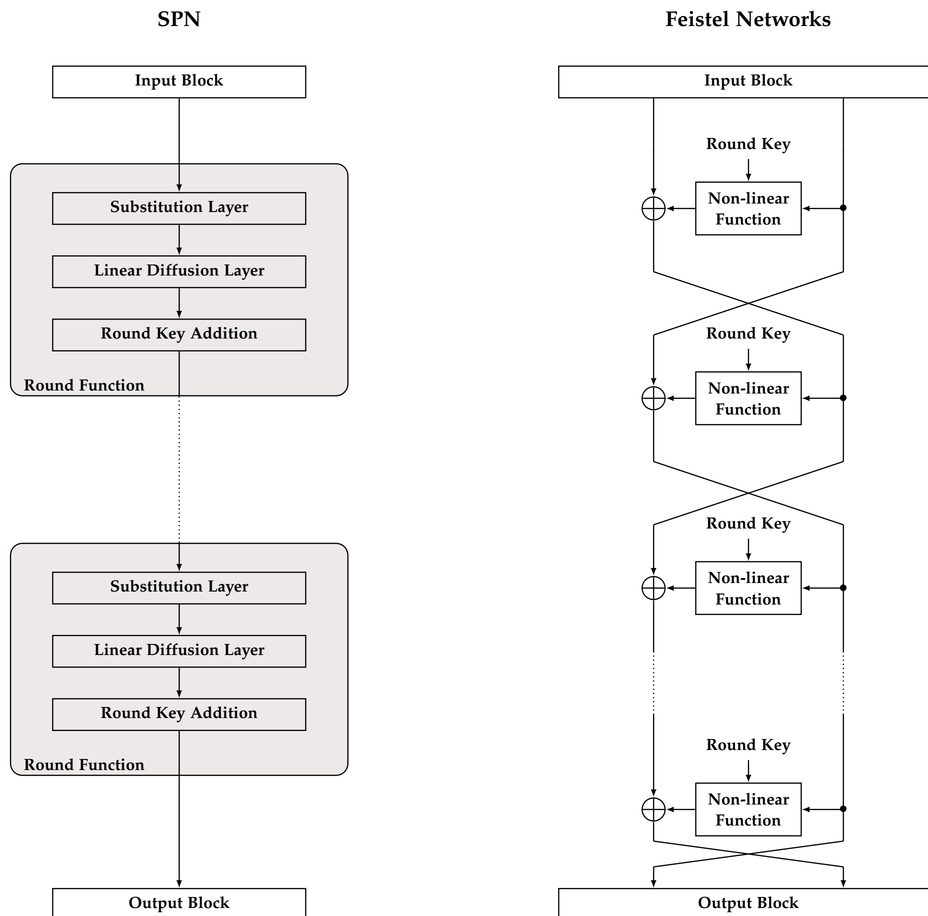

Figure 3: SPN and Feistel networks.

AES Block Cipher

Currently, the state-of-the-art block cipher primitive is the broadly adopted *Advanced Encryption Standard* (AES) algorithm [AES01]. AES is a block cipher designed to encrypt blocks of 128 bits with a 128, 192 or 256-bit secret key.

AES is built on a 10, 12 or 14-round[3] SPN construction. The round function is the composition of a non-linear substitution layer with two linear diffusion layers and a round-key addition layer. The current state value is viewed as a 4×4 matrix of bytes. With this representation the four layers are operating as follows:

- SubBytes, each byte of the matrix is processed through the same S-Box. A S-Box is a n-bits to m-bits non-linear map that is efficient and easy to implement, for instance through its lookup table representation. The AES S-Box is a 8-to-8 bits non-linear map.
- ShiftRows, that applies left circular rotations of 1, 2 and 3 bytes on the last three rows of the matrix.
- MixColumns, a linear mixing operation performed on the columns of the matrix.
- AddRoundKey, where the current round key derived from the secret key is xored with the state matrix.

Note that the first round is preceded by an AddRoundKey operation and the last round is performed without the MixColumn operation.

Tweakable Block Cipher and XEX Construction

A *tweakable block cipher* (TBC) is another primitive, first proposed by Liskov et al. [LRW11]. A TBC is a block cipher that takes, in addition of a secret key and a block of data, a third input denoted *tweak*. The role of the tweak is to modify the behaviour of the block cipher, that is two TBC calls with the same secret key and input block but different tweak values will return two "independent" output block values. Several algorithms relying on a TBC are depicted in Section 2.3, with OCB, COLM and Deoxys, and in Chapter 3 with AEZ.

The TBC used in the algorithms mentioned above are all based on the so-called *XOR-Encrypt-XOR* (XEX) construction [Rog04]. In this construction two offsets are derived from the input tweak value. To process a block of data, a first offset is xored to the input block, followed by the block cipher primitive and a xor with the second offset. Remark that the offset can also depend of the secret key, this is the case in the AEZ algorithm.

3 Depending on the key size.

Stream Cipher

A stream cipher is an encryption algorithm built around an internal state initialized from a secret key and an *Initial Value* (IV). This state is iteratively updated to generate a keystream. The plaintext is xored with the generated keystream to compute the ciphertext. Figure 4 depicts a classical stream cipher construction.

The IV is an important element in a stream cipher. Without an IV a stream cipher generates a keystream that only depends from the key. Thus, with the knowledge of one plaintext/ciphertext pair one can deduct the generated keystream computed from the secret key and be able to further decrypt messages without the knowledge of the secret key value, compromising the security. Hence, encryption of several plaintext without an IV must be performed with different secret keys, which is not convenient. Using an IV allows the encryptions of several plaintext without key replacement, assuming that the IV is changed for each encryption.

A stream cipher construction with IV can be seen as a family of functions parametrized by a secret key K that take as input an IV of n bits and generate a keystream of variable length. The expected security of a stream cipher with IV is the PRF criterion. Without an IV the stream cipher construction can be seen as a family of number generators taking a secret key K of k bits as the seed, and generating a keystream of variable length. The expected security of a stream cipher without IV is the PRNG criterion. The latter construction, without an IV is rarely used.

Remark that the plaintext and/or ciphertext can also be injected in the internal state to produce the keystream. We refer to a *synchronous* stream cipher when the keystream depends only on the key/IV pair and to an *asynchronous* stream cipher when the keystream also depends on the plaintext and/or ciphertext.

Permutations

As said before, the use of permutations as primitives instead of block cipher is a recent trend. Several propositions follow this idea, as the two cryptographic algorithms analysed latter in this manuscript (cf. Chapter 4 and Chapter 5). A permutation is defined as a bijection mapping n bits to n bits with n the width of the permutation. Seen as a parameter, the width of a permutation can be modified to suit specific implementations allowing versatility in the use of a permutation.

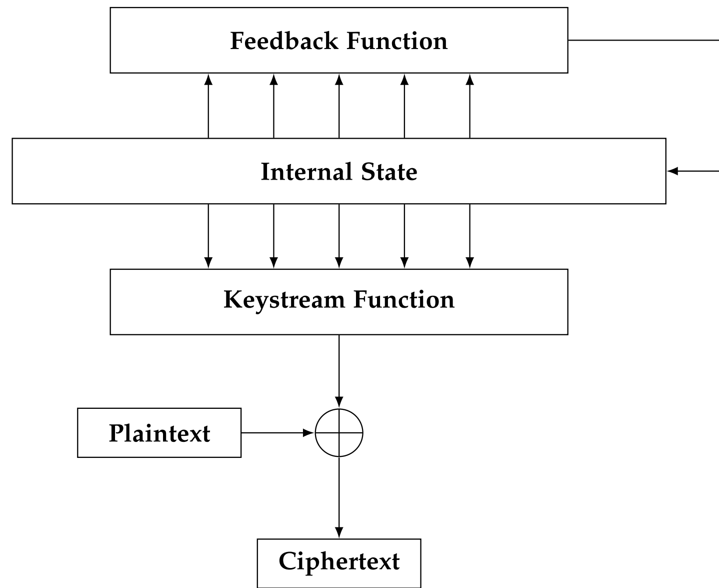

Figure 4: Stream Cipher Construction.

The security expected from a permutation is not easy to formalize and depend on the use of the permutation. Heuristically, a good permutation should avoid any structural property that can be satisfied for given input and output pairs. In Chapter 4, where we analyse the NORX algorithm, we provide a compelling example of such a structural property and exploit it to mount an attack.

1.2.3 Primitives are Used in Larger Constructions

Stream ciphers, by definition, can generate keystreams of arbitrary length. Therefore, encryption of a plaintext can be done with only one call to the primitive, that generates a keystream combined with the plaintext to produce the ciphertext. Hence, the stream cipher primitive is generally used on its own and not embedded in a wider construction to perform encryption.

Block ciphers, however, can only encrypt one block at the time. To perform encryption of plaintexts of arbitrary length a block cipher has to be incorporated in a larger construction named *mode of operation*. A mode of operation is an algorithm which specifies plaintexts of variable length how should be encrypted with a primitive. This definition also comprises the recent use of permutations as primitives, embedded in a more involved construction.

Before detailing the design of some modes of operation, we must define the expected security for a mode.

Security Notion for Encryption Algorithms

The main security notion sought for an encryption algorithm is the notion of *indistinguishability* [BDJR97]. The definition of indistinguishability can be viewed as a game between an adversary and an oracle, that is an independent third party that will receive queries and send non-deterministic responses. Three main versions are usually considered, IND-CPA for indistinguishability under *Chosen Plaintext Attack*, IND-CCA (or IND-CCA1) for indistinguishability under chosen ciphertext attack[4] and IND-CCA2 for indistinguishability under adaptive chosen ciphertext attack. They differ from the adversary freedom with IND-CCA2 allowing the largest variety of queries by the adversary. IND-CCA2 security implies IND-CCA1 security which in turns implies IND-CPA security.

We denote $\mathcal{SE} = (\mathcal{E}, \mathcal{D}, \mathcal{K})$ a symmetric encryption scheme defined by an encryption algorithm $\mathcal{E}$, a decryption algorithm $\mathcal{D}$ and a set of secret keys $\mathcal{K}$. The adversary is denoted A.

The IND-CPA security can be formalized with the following Guess game:

1. The oracle randomly computes a value $b \overset{\$}{\leftarrow} \{0,1\}$, and chooses a random secret key $K \overset{\$}{\leftarrow} \mathcal{K}$;
2. the adversary A can query encryption of arbitrary plaintexts to the oracle, then he chooses two plaintexts of equal length M_0 and M_1, and sends them to the oracle;
3. the oracle encrypts the plaintext M_b under the secret key K and sends the ciphertext $C = \mathcal{E}_K(M_b)$ to A;
4. A returns a value $b' = \mathrm{Guess}_{\mathcal{SE}}(A)$ to the oracle and wins if $b' = b$, that is he correctly guessed which plaintext was encrypted by the oracle.

We define the IND-CPA advantage of A for the symmetric scheme $\mathcal{SE}$ by

$$\mathrm{Adv}_{\mathcal{SE}}^{IND-CPA}(A) = |\Pr[\mathrm{Guess}_{\mathcal{SE}}(A) \Rightarrow win] - \Pr[\mathrm{Guess}_{\mathcal{SE}}(A) \Rightarrow loss]| \ .$$

A symmetric encryption scheme is considered secure with respect to the IND-CPA security notion or equivalently semantically secure if this advantage remains small.

4 Non-adaptive chosen ciphertext attack.

The IND-CCA1 and IND-CCA2 securities can be formalized as follows:

1. The oracle randomly computes a value $b \xleftarrow{\$} \{0, 1\}$, and chooses a random secret key $K \xleftarrow{\$} \mathcal{K}$;
2. the adversary A can query encryption or decryption of arbitrary plaintexts and ciphertexts to the oracle, then he chooses two plaintexts M_0 and M_1, and sends them to the oracle;
3. the oracle encrypts the plaintext M_b under the secret key K and sends the ciphertext $C = \mathcal{E}_K(M_b)$ to A;
4. in the IND-CCA2 setting, the adversary is allowed to query encryption or decryption of more plaintexts or ciphertexts to the oracle, but obviously cannot query the ciphertext returned by the oracle in the previous step;
5. A returns a value $b' = \text{Guess}_{\mathcal{SE}}(A)$ to the oracle and wins if $b' = b$.

We define the IND-CCA1 (resp. IND-CCA2) advantage of A for the symmetric scheme $\mathcal{SE}$ by

$$\text{Adv}_{\mathcal{SE}}^{IND-CCA*}(A) = \left| \Pr[\text{Guess}_{\mathcal{SE}}(A) \Rightarrow \ win] - \Pr[\text{Guess}_{\mathcal{SE}}(A) \Rightarrow \ loss] \right| \ .$$

A symmetric encryption scheme is considered secure with respect to the IND-CCA1 or CCA2 security notion if this advantage remains small. What differs in the IND-CCA2 notion in comparison with the IND-CCA1 notion is the ability for the adversary to ask the decryption of a ciphertext C' built from C in order to distinguish M_b.

Modes of Operation

The simplest mode of operation is the *Electronic Code Book* (ECB) [Dw001] and simply consists of the independent encryption of each plaintext block with the same block cipher instance, as seen in Figure 5.

Although being efficient and simple to implement, the ECB mode offers no semantic security or IND-CPA security. Indeed, two blocks with the same value will give the same ciphertext output blocks independently of their positions in the plaintext. Thus, the IND-CPA game can be correctly guessed by providing to the oracle two messages with a specific structure, for instance

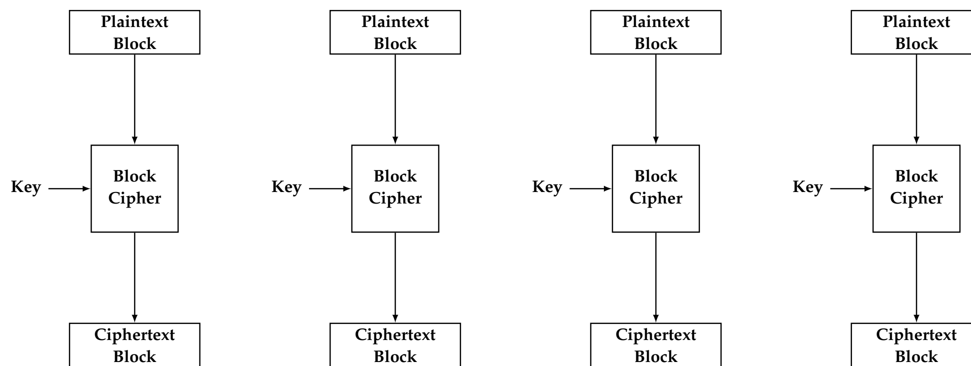

Figure 5: ECB mode of operation.

$M_0 = (B_0, B_1)$ and $M_1 = (B_1, B_1)$ where B_0 and B_1 are two arbitrary different blocks[5]. Such behaviour is not desirable, especially when encrypting large files such as pictures, where encryption with the ECB mode does not "randomize" the output, and provides no confidentiality.

Fortunately, other modes of operation exist [CSA16], such as Cipher Block Chaining (CBC), Cipher FeedBack (CFB), Output Feedback (OFB) and Counter (CTR) modes, all displayed in Figure 6. All these modes are IND-CPA secure but not IND-CCA secure. Remark that OFB and CTR modes can be viewed as stream ciphers with a keystream generated in blocks instead of single bits.

Sponge Construction Based on an Unkeyed Permutation

The sponge construction is a mode of operation relying on an unkeyed permutation instead of a block cipher primitive that was initially introduced to derive a hash functions, another symmetric cryptography tool [BDPA11b]. The concept of the sponge construction is very simple: an internal state split into two parts, the *rate* and the *capacity*, is updated with a permutation through an absorbing and a squeezing phase. In the absorbing phase, blocks of data are injected in the rate part of the sponge, leaving the capacity part of the sponge unaltered. In the squeezing phase, the value of the rate part is returned after each state update. The processed input and output blocks are separated by a state transition performed by the unkeyed permutation as seen in Figure 7.

Formalizing the security expected from a sponge function is not trivial, proofs of security for the sponge construction are achievable in an idealized

5 If the oracle encrypts M_0 then the two blocks of the ciphertext will be different while the two blocks of the ciphertext will be equal if the oracle encrypts M_1.

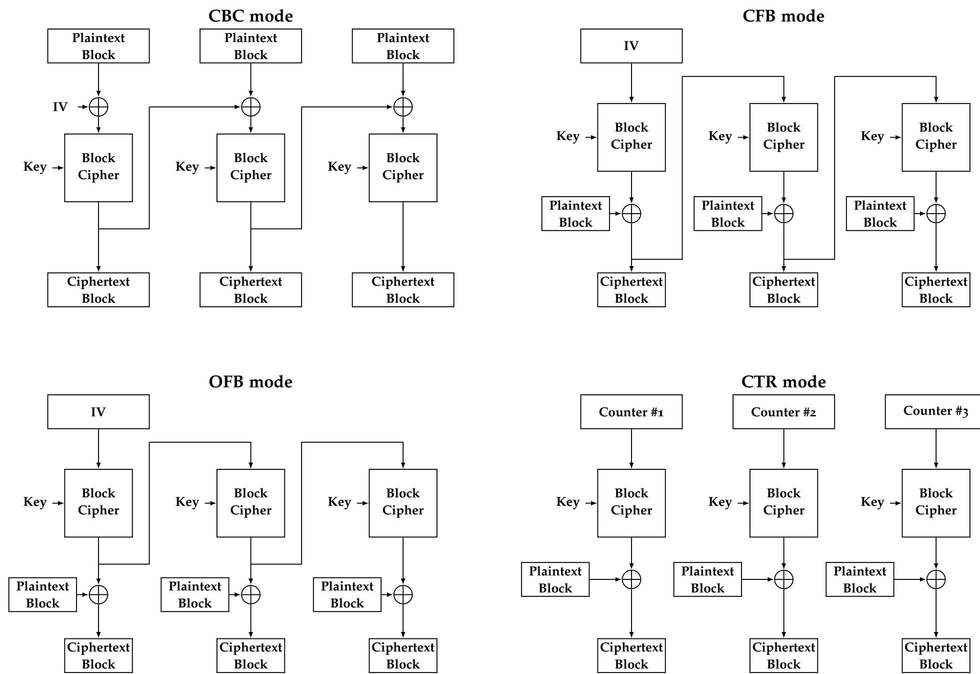

Figure 6: CBC, CFB, OFB and CTR modes of operation.

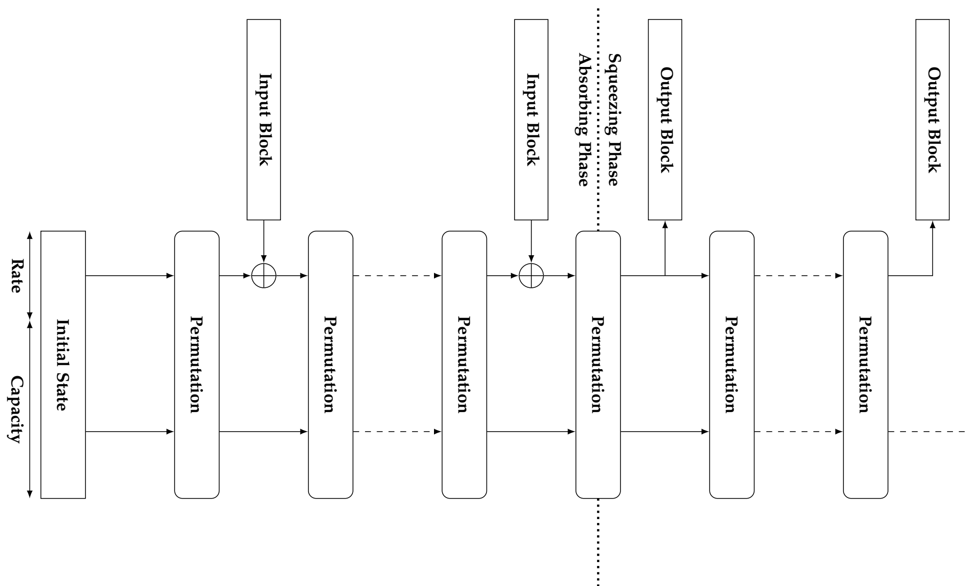

Figure 7: Sponge construction.

model where the actual and fully instantiated permutation is replaced by an ideal permutation [ADMA15, DMA17].

While initially introduced to construct hash functions, the sponge construction can also be used for further applications. The *duplex* version of this design [BDPVA11] enables additional encryption features. *Duplexing* the sponge consists in mixing the absorbing and squeezing mechanisms, that is input and output blocks are processed by pairs between two state transitions as depicted in Figure 8. The block of ciphertext is generated by combination of the input with the output block.

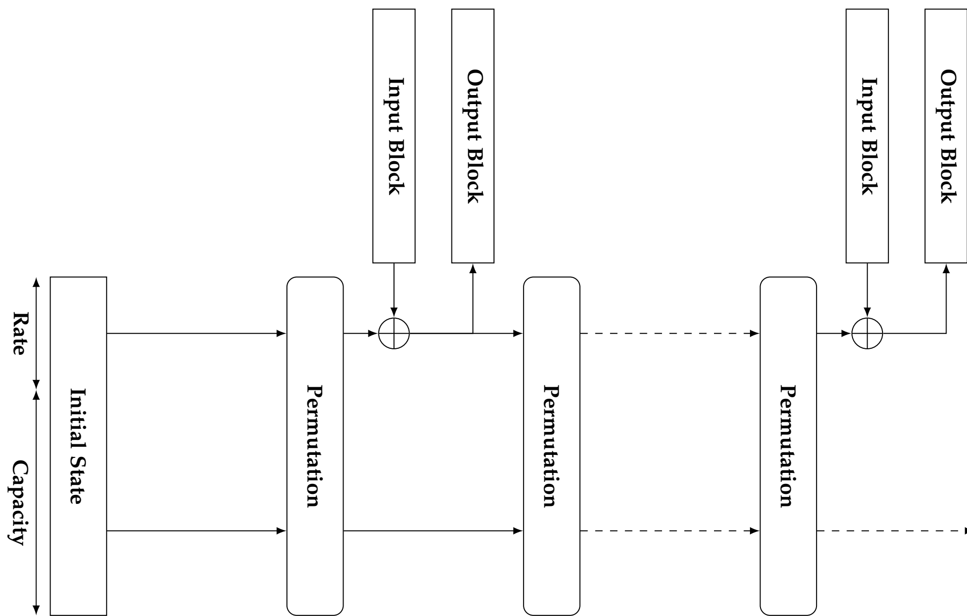

Figure 8: monkeyDuplex construction.

More Than Confidentiality ?

We saw that constructions and modes based on strong cryptographic primitives such as block ciphers or permutations allow encryption of arbitrary long messages while providing the confidentiality of the data. But these constructions can also be used to target further cryptographic purposes, like message authentication, described in the next section.

1.3 Beyond Confidentiality, Authenticity

Transmission of encrypted data on a public channel can be subject to many alterations. Senders and receivers want to protect their messages against

events that come from the channel itself, like packet loss or poor signal, or from the data manipulation by malicious people. This is where *authenticated encryption* algorithms (AE), come into play with further security objectives than mere confidentiality.

1.3.1 Message Authentication

Beyond confidentiality, an authenticated encryption algorithm aims at providing *message authentication*, that is, combining the ability to verify the identity of the sender and the *integrity* of the data. Verifying the identity of the sender means that the receiver can verify the source of the data while integrity ensures that the ciphertext was not modified during the transmission. Authentication is obtained by a *ciphertext expansion*, that is the output of an AE algorithm is larger than the original message, which is necessary to guarantee the two notions simultaneously.

1.3.2 Formalisation

The encryption part of an AE algorithm can be defined as the following map,

$$\begin{array}{ccc} \{0,1\}^k \times \{0,1\}^p \times \{0,1\}^n \times \{0,1\}^{ad} & \longrightarrow & \{0,1\}^{c+\tau} \\ (K,P,N,AD) & \longmapsto & C = Enc_K(K,P,N,AD), \end{array}$$

where K and P are key and plaintext as for a block cipher, and where AD are *associated data* (also sometimes referred to as a *header*) and N a *nonce* or an *Initializing Vector* (IV). The associated data are optional and not encrypted, this is a feature to allow authentication of additional data which do not require encryption. When allowing associated data, we refer to an *authenticated encryption with associated data* (AEAD) algorithm. The nonce is by definition a value that can be used only once. One role of the nonce is to make the encryption non-deterministic, using two different nonces to encrypt the same message will result in two different ciphertexts. Note that some AE constructions omit the nonce. The ciphertext expansion mentioned above involves an expansion of τ bits. The decryption function is defined as follows,

$$\begin{array}{ccc} \{0,1\}^k \times \{0,1\}^{c+\tau} \times \{0,1\}^n \times \{0,1\}^{ad} & \longrightarrow & \{0,1\}^p \cup \perp \\ (K,C,N,AD) & \longmapsto & Dec_K(C,N,AD) = P \text{ or } \perp. \end{array}$$

One can note that the output of the decryption algorithm can be an error message, denoted $\perp$. To protect the confidentiality of the plaintext no information must be revealed before the authentication is verified. If the verification of the authentication fails then an error message is returned. In most cases, the plaintext needs to be recovered before the verification of the validity of the ciphertext.

1.3.3 Security of AE Schemes

The expected security property of an AE scheme in addition to the ciphertext indifferentiability requirement of a classic encryption scheme is to provide *integrity of ciphertexts*, as defined by Bellare and Namprempre in [BN00]. The integrity of ciphertexts means that the probability for an adversary to successfully produce a valid ciphertext of a plaintext not previously encrypted by the sender must remain negligible without the knowledge of the secret key.

The first solution to convert an encryption system into an AE scheme is the use of an additional dedicated algorithm, denoted *Message Authentication Code* (MAC). A MAC is an algorithm parametrized by a secret key that takes a string of data to authenticate and returns an *authentication tag* of fixed length. The security expected for a MAC is the *non-forgeability*. The probability for an adversary to produce a valid authentication tag for a message without prior knowledge of the secret key must be negligible[6]. Note that two secret keys are required to ensure security, one for the encryption algorithm and one for the MAC, and must be distinct to avoid security issues. Three generic composition methods can be found in the literature:

- *Encrypt-and-MAC*, the plaintext is encrypted to produce a ciphertext and a MAC of the plaintext is concatenated to the ciphertext;
- *MAC-then-Encrypt*, a MAC of the plaintext is produced first, following by the encryption of both the plaintext and the MAC;
- *Encrypt-then-MAC*, the plaintext is encrypted to produce a ciphertext and a MAC of the ciphertext is appended.

These three compositions are considered secure when used with a nonce, and only the Encrypt-then-MAC composition can be considered secure if the nonce is omitted. Figure 9 depicts each composition.

Misuse Resistance [RS06] (MRAE algorithms) is a stronger security notion for AE schemes. MRAE addresses the possibility for an adversary to violate some of the requirements for an AE algorithm. For instance, a MRAE algorithm allows an adversary to repeat the public nonce to encrypt different messages with the same key (*nonce misuse*). MRAE also encompasses the release of unverified plaintexts, that is plaintexts are released even if the tag verification fails instead of a resulted error message. Remark that this security notion is not attainable by an *Online* AE scheme, that is an algorithm where the encryption of the $n+1$-th block depends only on the last n-th block ciphertext value. Online AE schemes are used for the encryption of large messages for instance.

6 At most $2^{-\tau}$ for a tag of length τ.

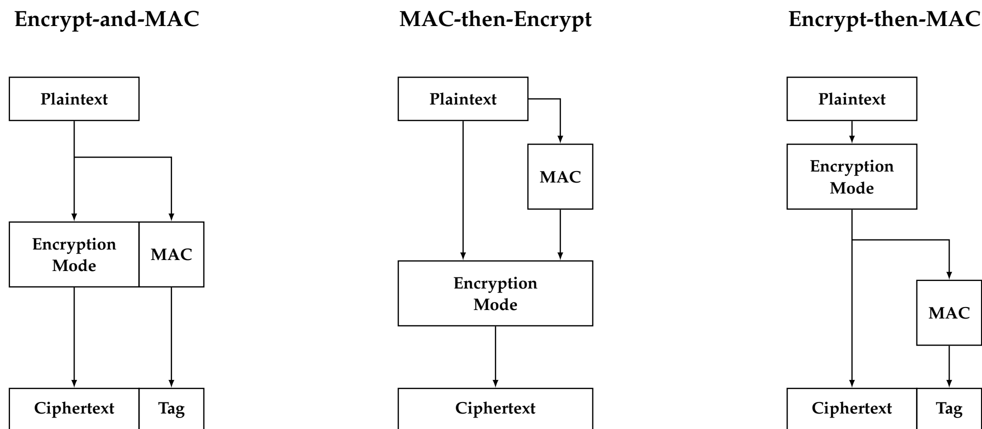

Figure 9: Generic compositions of an encryption scheme and a MAC.

1.3.4 Constructions and Modes of Operations to Achieve Authentication

The first modes of encryption were designed to encrypt long messages with block ciphers and provide confidentiality, but lacked of authentication properties. When the need for message authentication began to spread, the first response was to use an additional algorithm dedicated to authentication, leading to CBC-MAC, a MAC based on the CBC construction. But CBC-MAC was not flawless[7], so HMAC[8] [BCK96] and CMAC[9] [BR00], two message authentication modes, were adopted.

But these MAC modes only provide message authentication, meaning that performing encryption and authentication with one of these modes should require two distinct secret keys in two dedicated algorithms, in order to avoid security issues. Thus, the community began to design modes of encryption combining both confidentiality and authenticity under the same key. This led to the first authenticated encryption modes of operation like Counter with CBC-MAC (CCM) [DD07] or Galois/Counter Mode (GCM) [MV04a]. But these modes of operation do not suit all modern cryptographic purposes, like security in constrained resource environments or improved performance on high-end system, and the cryptographic community continues to seek for more secure and/or efficient AE algorithms. In Chapter 2 we provide an overview of CAESAR, an open competition aiming at selecting the next AE algorithms dedicated for these new cryptographic purposes.

7 It requires for instance to input only constant-length messages since enabling variable-length messages leads to simple attacks.
8 Hash-based Message Authentication Code.
9 Cipher-based Message Authentication Code.

1.4 How to Build Strong Cryptographic Primitives?

We saw in the previous sections that primitives are used in wider constructions named modes of operation. To achieve secure encryption or authentication with these modes their underlying primitives must comply with the adequate security. Hence, while modes can rely on security proofs, how can primitives and *ad hoc* constructions that do not rely on a trusted primitive and security proof be considered secure?

The main answer is *cryptanalysis* of cryptographic objects, that is the study of the security claimed by their designer. By analysing the structure of a cryptosystem, cryptanalysts try to exhibit flaws that could be turned in attacks and lead to a security breakdown. Before detailing the main cryptanalysis strategies, we need to define under which criteria a cryptanalysis can be considered strong and efficient.

1.4.1 Attack Success

In symmetric cryptography, the use of secret keys of fixed length implies that every cryptosystem can be broken within a finite amount of time. Indeed, since the secret key has a fixed length, recovering the secret key can be done by exhausting all the possible values, the verification requiring only one (or a few)[10] known plaintext/ciphertext pairs. Once the secret key is revealed an adversary can then encrypt and authenticate any messages of his choice, and confidentiality and authenticity collapse.

Hence, no encryption scheme parametrized by a secret key is unconditionally secure. This method is called *exhaustive-key search*, and involves only computational power. Assuming that the length of a secret key is k bits, anyone can recover the value of this secret key with one (or few) plaintext/ciphertext pairs and 2^k encryptions. When designing a symmetric algorithm, the authors calimed a security level measured in bits. This means they assume that no attack can be performed with a time complexity whose binary logarithm is lower than the claimed security level.

For instance, the designers of the `AES-128` block cipher claim 128 bits of security. This means that no attack can be performed to break the `AES` security with a complexity lower than 2^{128} operations. The complexity of an attack is measured in basic operations performed by a computer and encompasses time, memory and data complexity. Nowadays, on a regular computer with a basic `AES` implementation, one can verify approximatively 2^{30} keys by

10 If the plaintext size is lower than the key size.

minute[11], it implies that the recovery of an `AES-128` secret key would take more than 10^{23} years, which is highly non-practical. The current minimal key size recommendation to ensure a sufficient /medium/long term security of a strong block cipher is 128 bits [ANS14].

The efficiency of an attack relies on two factors:

– its complexity, that is how much the security claimed by the designer is deteriorated; a practical cryptanalysis will have a heavier impact than a cryptanalysis that breaks the claim security by a small margin;

– the criticality of the attack, that is which security is damaged; a distinguisher is less harmful than a full-key recovery.

Yet, we must emphasize that every cryptanalysis is relevant, since it is a direct contradiction with the security claimed by the designers. Furthermore, a small attack can be the vector for new ideas and be extended with further analyses to a more relevant one.

1.4.2 Context of Cryptanalysis

The impact of a cryptanalysis also depends on the context of the attack. By context, we mean what data is required for an adversary to perform the attack. We can distinguish four categories of attacks, sorted from less constrained to most constrained.

– *ciphertext-only attack*: the adversary only knows the value of some ciphertext outputs and may have some knowledge about the plaintext distribution;

– *known-plaintext attack*: the adversary has access to plaintext/ciphertext pairs encrypted under the same key;

– *chosen-plaintext attack*: the adversary has access to an encryption oracle to retrieve ciphertexts corresponding to plaintexts of his choice;

– *chosen-ciphertext attack*: the adversary has access to a decryption oracle, in addition of an encryption oracle, and can retrieve the plaintext value for ciphertexts of his choice.

Note that in the case of a chosen-plaintext (resp. chosen-ciphertext) attack, the set of the required plaintexts (resp. ciphertexts) may have to be specified before processing the attack. If the adversary has the ability to choose the

11 2^{30} keys by minute performance is a naive approximation which does not take into account of possible software optimization, parallelization and other techniques to speed-up the exhaustive search.

value of the processed plaintext (resp. ciphertext) from the knowledge of previous encryptions (resp. decryptions), the attack is known as an adaptive chosen plaintext (resp. adaptive chosen-ciphertext) attack.

1.4.3 Cryptanalysis Strategies

We can group the strategies used to target symmetric encryption algorithm in three main categories: statistical attacks, algebraic attacks and structural attacks.

Statistical attacks are performed by exploiting statistical biases in the output of a cipher or some intermediate values, and/or abnormal correlation with the inputs. With the appropriate input, the anomalous statistical behaviour can be detected and used to recover secret information on the original plaintext or the secret key. These statistical attacks usually require a large data complexity to exploit this undesired statistical behaviour. Differential [BS91] and linear [TG91] cryptanalysis of block cipher such as DES represent the most famous statistical attacks.

Algebraic attacks recover secret information by generating and solving multivariate algebraic systems over a finite field, expressed in the secret key or a plaintext. The structure and solvability of the system depends on the internal design of a cipher. To thwart algebraic attacks, designers try to increase the non-linearity of their system and improve the linear diffusion throughout the cipher. For instance, Courtois published in [Cou03] algebraic attacks on a family of LFSR-based stream ciphers.

The last category, *structural attacks*, relies on the specific design of system to exhibit vulnerabilities. They do not take account internal properties of the elementary components, like statistical or algebraic properties, and focus on the construction itself. For example, the *square attack* that first targeted the Square cipher [DKR97] and also applies to a round-reduced version of AES, is exploiting structural properties of the ciphers.

1.5 Contributions

This thesis focuses on the cryptanalysis of symmetric encryption algorithms. After an introduction in Chapter 1 of the fundamental notions encountered in symmetric cryptography along with a motivation for cryptanalysis, Chapter 2 provides an overview of the CAESAR competition that aims at selecting a portfolio of new AE algorithms. The main results of this thesis are three cryptanalyses, that target two high profile candidates of the CAESAR com-

petition and one PRF with variable input and output length proposed by the SHA-3 designers.

In Chapter 3, we describe a cryptanalysis of the CAESAR candidate AEZ v4.1 [CG16]. AEZ is an authenticated encryption construction that involves a mixture of a 4-round reduced version of AES-128 and the full 10-round version of AES-128. These two underlying primitives are encompassed in the XEX construction to build a tweakable block cipher primitive. In AEZ, three sub-keys are derived from the secret key and used to compute the input and output offsets in the tweakable block cipher. The attack is a combination of attacks with birthday-bound complexity that target the tweakable block cipher and a differential attack that focuses on the 4-round reduced version of AES. The combination of these attacks allows to recover the full secret key with overwhelming probability more efficiently than a generic exhaustive-key search.

In Chapter 4, we analyse the security of another CAESAR candidate, namely NORX v2. NORX is an authenticated encryption scheme based on the so-called Duplex sponge construction. The permutation underlying NORX is inspired from the permutation used in BLAKE [AHMP10] or ChaCha [Ber08]. We exploit a strong structural distinguisher of this permutation. Due to the non-conservative design choice of increasing the rate part of the sponge as compared with the initial version of NORX, we can leverage the structural distinguisher to mount a ciphertext-only forgery attack and a key-recovery attack with complexities far below the claimed security.

In Chapter 5, we depict a cryptanalysis of the recent Kravatte algorithm, an instance of Farfalle, a pseudo-random function with input and output of variable length. Farfalle is an efficient and parallelizable construction that relies on so-called rolling functions and a set of permutations based on the Keccak round function. We take advantage of the low algebraic degree of the underlying Keccak permutation to mount three key-recovery attacks targeting different parts of the construction: a higher order differential attack, an algebraic meet-in-the-middle attack and an attack based on a linear recurrence distinguisher. All the depicted attacks exhibit far below security claims complexities.

These three cryptanalyses have been published in the journal *Transaction on Symmetric Cryptology* and presented at the companion conferences FSE 2017 and FSE 2018 under the following titles:

– *"Is AEZ v4.1 Sufficiently Resilient Against Key-Recovery Attacks?"* [CG16];

– *"Cryptanalysis of NORX v2.0"* [CFG+17];

- *"Key-Recovery Attacks on Full Kravatte"* [CFG+18a].

Note that an extended version of *"Cryptanalysis of NORX v2.0"* will also be published in *Journal of Cryptology* [CFG+18b].

2 CAESAR COMPETITION

Throughout the history of cryptography, needs for standardization rose several times, to replace an ageing algorithm, to fulfill a specific requirement or to prevent attacks from forthcoming adversaries. These needs led to various cryptographic competitions aiming at specifying new standardized algorithms.

The most famous example is the AES algorithm [AES01], which replaced the previously standardized block cipher DES [DES77]. The need for a more resilient block cipher arose with the increase of computational power provided by computers, especially against brute-force attacks, since the 56-bit key of the DES was becoming small. To find the successor of the DES algorithm the NIST established a competition, namely the AES competition, in order to select the next standardized block cipher. The performance and security of fifteen initial submissions were scrutinized and confronted between each other. Finally, the NIST declared the Rijndael algorithm, designed by Vincent Rijmen and Joan Daemen, as the winner. In November 2001, AES was officially announced by NIST as the new standardized FIPS PUB 197 algorithm.

After the AES competition, many others have been initiated, and surely more will be in the future. Different topics were targeted by these competitions such as hash functions for the SHA-3 competition, stream ciphers for eSTREAM, or more recently, quantum-resistance in public-key cryptography algorithms with the Quantum-Safe Cryptography call for proposals. Along them the CAESAR competition is on-going, and this chapter will provide a detailed overview of this competition.

2.1 Context and Goals

CAESAR is the acronym for *Competition for Authenticated Encryption: Security, Applicability, and Robustness*. As its name suggests, this competition aims at selecting new authenticated encryption algorithms. It is supervised by a secretary, Daniel J. Bernstein, and monitored by the CAESAR committee, a panel of cryptographers.

The CAESAR competition is motivated by the need for dedicated, trustworthy and more resilient than current AE algorithms that fit the needs of

modern applications. As explained in Section 1.3, most of the current systems rely on two dedicated algorithms: one for encryption and one for authentication. Hence, implementation of these dedicated algorithms is prone to errors and must be handled carefully to avoid security breaches. An algorithm providing both confidentiality and data authentication is expected to be more user-friendly, with only one algorithm to implement. Moreover, the algorithms which will succeed in the competition are expected to benefit from the confidence of the cryptographic community towards a broad adoption.

Currently, the available state-of-the-art AE algorithm is AES-GCM [MV04b]. Although providing good performance, it does not suit all modern implementations. For instance, with the exponential growth of devices connected to the internet, the so-called Internet of Things, confidentiality and data authentication are required inside constrained environments. When embedded in tiny hardware like RFID tags or small electronic devices, secure algorithms have to deal with limited computational resources, e.g., a low area, and still provide a required level of security.

Furthermore, AES-GCM strongly requires a nonce-respecting utilisation, and is vulnerable to key-recovery attacks if the nonce is repeated, as shown by Joux in [Jou06]. Hence, algorithms more resilient against misuse scenarios are in the scope of the CAESAR competition.

Unlike the AES process where only one winner had been selected, a portfolio of algorithms will be released at the end of the CAESAR competition as the winners. To be selected in this final portfolio, candidates require to provide equal or better performances than AES-GCM; more resilience against misuse scenarios; and/or to exhibit suitability for resource-constrained environments. These criteria are covered by the three following use cases[1] (which are not mutually exclusive):

- use case 1 : lightweight applications, that is constrained environments with few resources like IoT[2] or RFID tags;
- use case 2 : high-performance applications, that is efficiency on high-end platforms like computers, servers or smartphones;
- use case 3 : defence in depth, that is resilience and robustness in misuse scenarios like nonce repetition or release of unverified plaintexts.

1 Defined in the Google group of the competition at https://groups.google.com/forum/#!topic/crypto-competitions/DLv193SPSDc.

2 IoT stands for *Internet of Things* and comprises all devices connected to internet, such as clock, electrical appliances, etc.

Candidates which exhibit suitability for one or more of the above use cases, along with secure, reliable, and efficient authenticated encryption will likely be part of the final portfolio.

2.2 Timeline

The competition has been announced at the Early Symmetric Crypto workshop in January 2013 with a deadline for the call for submissions set to March 2014. Fifty-seven candidates have been submitted for the initial round.

With block cipher, stream cipher, or sponge oriented constructions, and novel designs or modes of operation with existing cryptographic primitives, a wide variety of designs have been submitted to the first round of the competition. Sixteen months later, in July 2015, the second-round candidates were announced. At this time, nine submissions had been withdrawn due to critical cryptanalyses. The weakest candidates did not reach the second round and twenty-eight candidates remained in July 2015, then further reduced to fifteen in August 2016 for the third round.

From the fifty-seven initial submissions, seven were selected on March 2018 for the final round. If no new relevant cryptanalysis is found, all the finalists should be in the final portfolio. Winners of the competition should be announced by the end of the year 2018. Below is the list of the selected finalists along with their targeted use case[3]:

- `ACORN` (v3) [Wu16], use case 1,
- `Ascon` (v1.2) [DEMS], use case 1,
- `AEGIS` (v1.1) [WP16], use case 2,
- `MORUS` (v2) [HW16], use case 2,
- `OCB` (v1.1) [RBBK01], use case 2,
- `COLM` (v1) [ABD+16b], use case 3,
- `Deoxys-II` (v1.41) [JNPS16], use case 3.

2.3 Finalists

As said before, a wide variety of algorithms have been submitted to the CAESAR competition. The seven remaining algorithms are still a reflection of this variety, and Table 1 provides a summary of the parameters, the construction

3 As defined by the CAESAR committee, not by the designers.

and the data limitations of each finalist. For some algorithms, several sets of recommended parameters are defined by the designers.

Table 1: Summary of the CAESAR competition finalists

Algorithm	**Key** (bit)	**Nonce** (bit)	**Tag** (bit)	**Construction**	**Data Cap Limit**[b]
ACORN	128	128	128	Asynchronous Stream Cipher	2^{64} bits[*]
Ascon	128[1,2]	128[1,2]	128[1,2]	MonkeyDuplex	2^{64} blocks
AEGIS	128[1,2], 256[3]	128[1,2], 256[3]	128[1,2,3]	Asynchronous Stream Cipher[a]	2^{64} bits[*]
MORUS	128[1,2], 256[3]	128[1,2,3]	128[1,2,3]	Asynchronous Stream Cipher[a]	2^{64} bits[*]
OCB	128, 192, 256	128	64, 96, 128	AES Mode of Operation	2^{48} blocks
COLM	128[1,2]	64[1,2]	128[1,2]	AES Mode of Operation	2^{64} bits[*]
Deoxys-II	128[1], 256[2]	120[1,2]	128[1,2]	TBC + Mode of Operation	2^{128} bytes[**]

[a] Blocks of keystream are generated instead of a single output bit.
[b] The amount of data that can be processed under the same key/nonce pair.
[*] Up to 2^{64} bits of associated data and up to 2^{64} bits of plaintext.
[**] Total size of associated data and plaintext. Up to 2^{64} messages processed under the same key.
[1,2,3] Primary, secondary and tertiary recommendation.

An overview of the finalists is provided in the following.

2.3.1 ACORN v1.2

ACORN [Wu16] is an AEAD algorithm targeting lightweight applications designed by Hongjun Wu with a structure close to the one of an asynchronous stream cipher. ACORN is based on an 293-bit internal state updated at each state transition to generate a single keystream bit. ACORN differs from a traditional stream cipher by the injection of the plaintext inside the internal state. The internal state is built from the concatenation of several linear feedback shift registers (LFSR). Two quadratic functions, namely the feedback and keystream function, complete the construction. Note that since the feedback function depends on the keystream bit, updates of the internal state depend on the previously encrypted bits. ACORN makes use of a simple operation set, XOR, AND, NOR, and rotation, whose hardware implementation costs are low.

Encryption of a plaintext requires four different steps: initialization of the state, associated data loading, encryption with keystream, and tag generation. Each step consists of an iteration of the state update function with slight variations depending on the current step. The state update function combines the LFSR update mechanism and the computation of a feedback bit injected in the internal state, with a keystream bit generated only during the encryption step. The whole process is depicted in Figure 10.

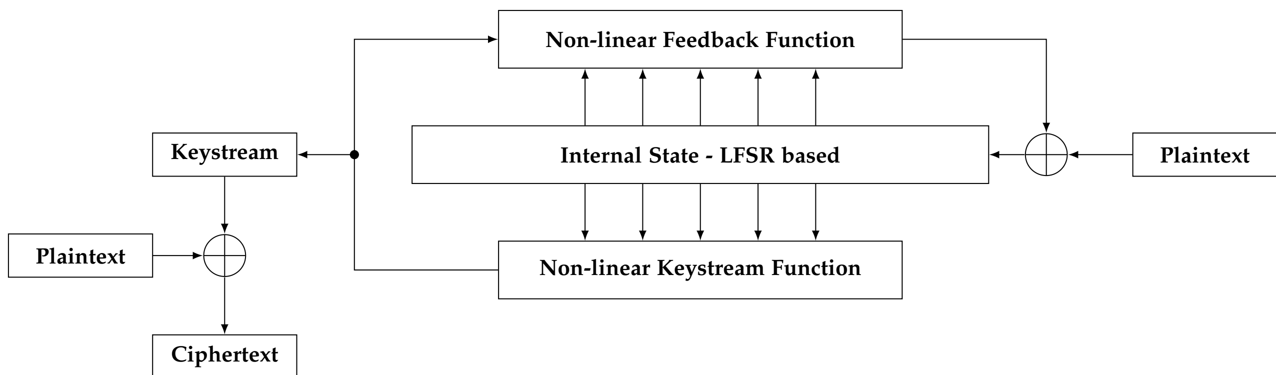

Figure 10: ACORN algorithm - encryption process.

The feedback and keystream function, both quadratic and lightweight, along with the LFSR structure make the ACORN design really suitable for lightweight applications of use case 1, implementation on constrained environments and low computational cost. While the construction of ACORN is quite aggressive, several cryptanalyses[4] [LLMH16, JS15, SWB+15, CFG15, LL14] concluded that its security level remains unaltered as soon as the nonce is not repeated under the same key. To confirm the security degradation under nonce repetition, we showed in a joint work with Thomas Fuhr and Henri Gilbert made before my thesis [CFG15] that the repeated use of the same couple key/nonce pair to encrypt messages with specific difference patterns allows to recover the internal state in practical time, but this requires to abuse the nonce-respecting requirement of the author and thus does not contradict the security claims of ACORN. Recovering the internal state in the first version of ACORN was equivalent to a successful forgery or a key recovery since each state update operation could be computed backwards or forwards. The author thwarted this potential key recovery in the second version by injecting the secret key in the initialization phase. In the current third version, internal state recovery can still allow tag forgeries.

2.3.2 Ascon v1.2

Ascon is a family of AEAD designs submitted by Dobraunig et al. [DEMS] that targets use case 1. Ascon is built from the MonkeyDuplex sponge construction [BDPVA11], with a 320-bit internal state. The rate and capacity sizes are defined for two instances, a primary recommendation with 64-bit

4 See https://groups.google.com/forum/#!topic/crypto-competitions/dzzNcybqFP4 for a list of the cryptanalyses along with the comments of the author.

rate and 256-bit capacity, and a secondary recommendation with 128-bit rate and 192-bit capacity.

The permutation used in the sponge mode of Ascon is based on a substitution-permutation network. Viewed as a 5 by 64 matrix, the internal state is processed vertically by a substitution layer of 64 parallel 5-bit S-boxes, and horizontally by a linear diffusion layer with 64-bit rotations and xor operations. A constants addition completes the substitution and linear layers to form one round of the permutation. Each round is repeated 6, 8 or 12 times, depending on the position of the permutation in the encryption process (cf. Figure 11).

Four steps are required to encrypt a message with Ascon: initialization, associated data, encryption, and tag generation. Initialization consists of setting a value for the initial state and applying 12 rounds of the permutation. The key is xored in the capacity of the state after the first permutation to avoid the backward computation of the initial state from an internal state recovery. Associated data and plaintext are split in blocks of 128 bits and xored with the rate; between consecutive blocks a 6-or 8-round permutation is processed (resp. for the primary and secondary instance of Ascon). Ciphertext blocks are generated from the rate value after the plaintext xor. Finally, to generate the authentication tag, the key is xored to the capacity, 12 rounds of the permutation are processed and the first 128 bits of the capacity are used as tag after a final key addition in the capacity. The whole process is described in Figure 11.

The permutation used in Ascon can be easily implemented in small devices, as done in [GWDE15] where the authors describe an efficient hardware implementation, thanks to the parallel application of the same 5-bit S-box and the use of rotation in the linear diffusion layer. Some cryptanalyses had targeted Ascon [Tez16, DEMS15, GRW16, LDW17], but none succeeded to break the full 12-round version and the best results achieve a successful attack on the 7-round reduced version. The complexities of the cryptanalyses mentioned above are displayed in Table 2.

2.3.3 AEGIS v1.1

AEGIS is an AEAD algorithm using the AES round as the underlying primitive, developed by Hongjun Wu and Bart Preneel [WP16] and selected for use case 2. AEGIS can be seen as an asynchronous stream cipher built

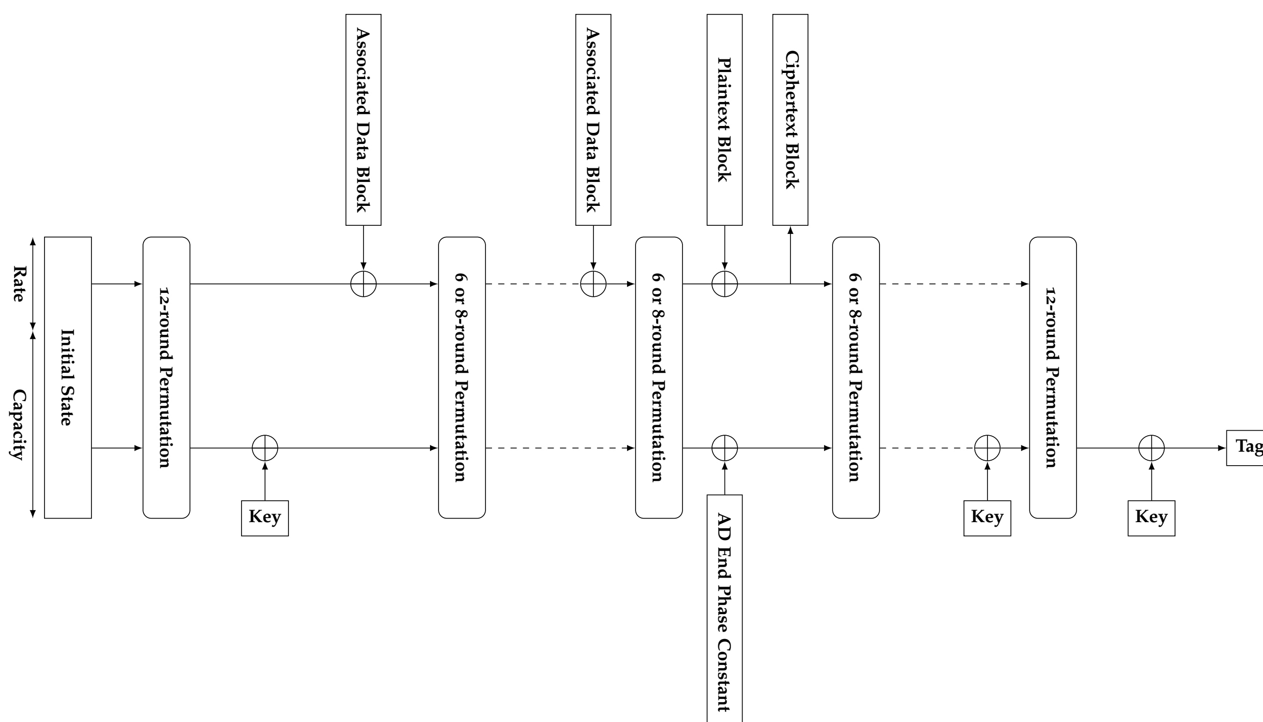

Figure 11: Ascon algorithm - encryption session.

Table 2: Main cryptanalyses results on Ascon.

Reference	Targeted version	Method	Time complexity
[Tez16]	5/12 rounds	Truncated/Impossible Diff.	2^{58*}or 2^{128}
[Tez16]	6/12 rounds	Cube-like	2^{66}
[LDW17]	7/12 rounds	Cube-like	2^{77**}or $2^{103.9}$

* For a set of 2^{64} weak keys.
** For a set of 2^{117} weak keys.

around an internal state of 1024, 640 or 768 bits[5] which generates a block of keystream at each state update.

Encryption of a message with AEGIS goes by initialization, associated data injection, encryption and tag generation, all performed by updating the state iteratively. The state update relies on a single round of the AES primitive. At each update the state is divided in 128-bit words, then, the value of each

5 Resp. for primary, secondary and tertiary recommendation.

word processed through the AES round is xored to its next right neighbour. Note that, like ACORN, blocks of plaintext are also injected in the internal state. The keystream block used to perform encryption is computed from a non-linear combination of internal words of the state. The whole process is depicted in Figure 12.

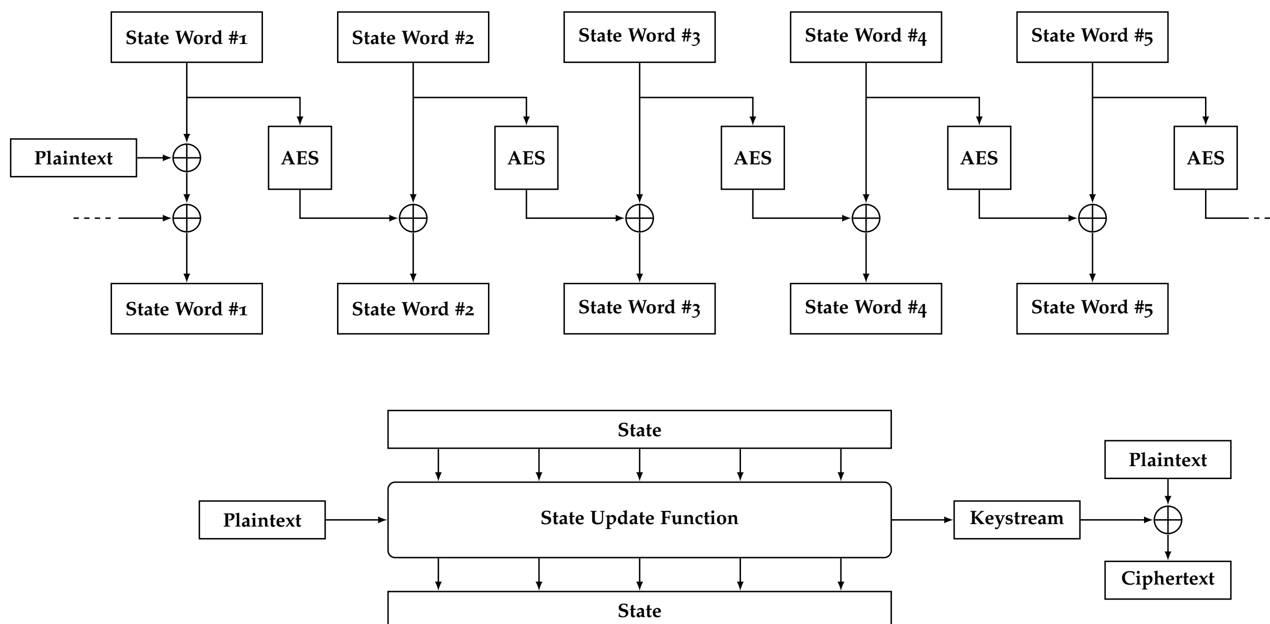

Figure 12: AEGIS state update (up) and encryption (down) process.

AEGIS exploits the support of the AES-NI instruction set available in recent CPUs since 2008, along with a simple and parallelizable stream-cipher-like processing, to provide suitability for software implementations with high efficiency. In [Min14] Minaud exhibits linear biases on AEGIS keystream. Since the biases discovered in the keystream of AEGIS do not depend on the key the data complexities of the attacks are not limited by the usage cap limit. However, only the tertiary recommendation could arguably be considered weakened. No other cryptanalysis has been performed against AEGIS.

2.3.4 MORUS v2

MORUS is an AEAD algorithm designed by Hongjun Wu and Tao Huang [HW16], that targets high-end platforms. MORUS shares many similarities about its internal design or its encryption process with AEGIS.

MORUS is built around a 1280-bit state (640 bits for the secondary recommendation) which is iteratively updated to perform the same encryption process as AEGIS. The main difference between MORUS and AEGIS is the state update function. While both perform operations on 128-bit words, instead of using the AES round function as a primitive, MORUS uses AND, XOR, and rotations. Since the encryption process is the same for AEGIS and MORUS, it will not be recalled here. Figure 13 depicts the construction used in MORUS.

The use of a *simple* function combined with an efficient design, similar to AEGIS, allows MORUS to achieve high performance in both software and hardware implementations [AA16]. The similarities between MORUS and AEGIS do not stop at design specifications. In a recent cryptanalysis [AEL+18] Ashur et al. described linear biases on the keystream output of MORUS, that allows an adversary to distinguish a keystream with 2^{152} encryptions. While it does not threaten the security claims for the primary and secondary recommendation, the tertiary recommendation in which a 256-bit key is used, can be targeted by this attack. Note that since the bias is in the keystream, it does not depend on the key and the limitation of the data encryption capability without key update imposed by the author does not prevent this attack.

2.3.5 OCB v1.1

OCB [RBBK01] is a mode used in AEAD settings developed by Ted Krovetz and Phillip Rogaway that was selected as a finalist for use case 2. Nine instances are described in the OCB submission, parametrized by the use of the underlying block cipher AES with a 128-bit, 192-bit or 256-bit key and the size of the generated tag, 128, 96 or 64 bits.

The encryption process with OCB is easy to describe. First an offset is derived from the nonce using a so-called incremental function, the number of offsets generated is equal to the number of data blocks to process, that is associated data and plaintext, plus a final offset for the tag generation. A checksum, equals to the xor of all plaintext blocks, is computed to generate the authentication tag. Each block of plaintext and the checksum are then xored with its corresponding offset, encrypted with the AES, and ultimately xored with the same offset to generate the ciphertext block and the authentication tag. The encryption process of OCB is depicted on Figure 14[6]. The offset-encrypt-offset construction can be seen as a tweakable block cipher based on the XEX construction. In the OCB construction, the position of the current block is the tweak value resulting in a different offset at each block position. This feature provide security by ensuring that the same block of

6 Only the case with no padding is depicted, refer to [RBBK01] for more details.

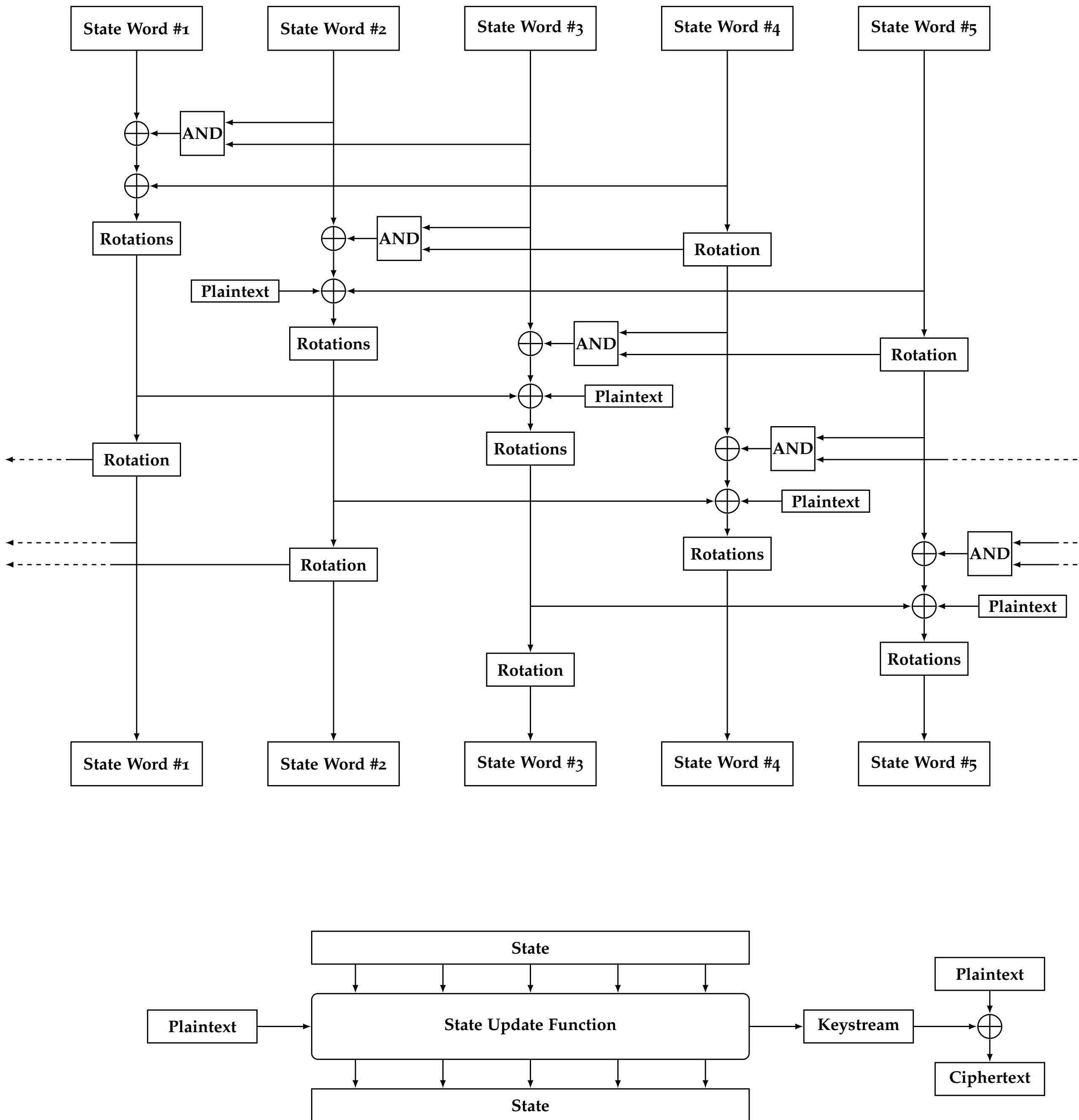

Figure 13: MORUS state update (up) and encryption (down) process.

data processed at different positions through the message does not generate the same output block, adding randomness to the ciphertext.

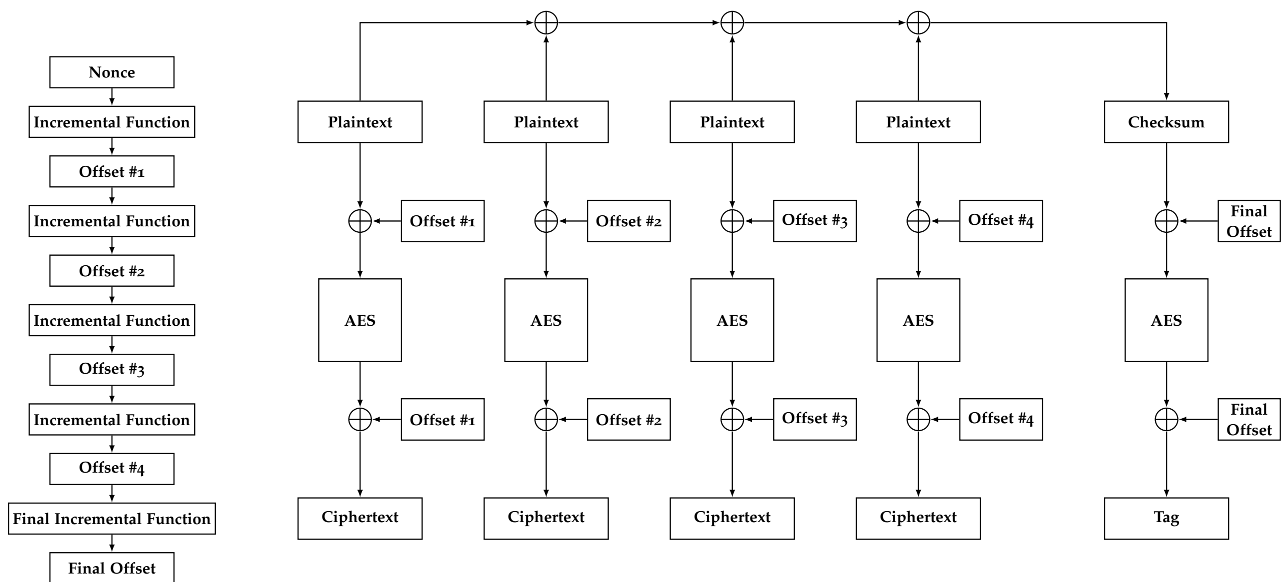

Figure 14: OCB encryption process without padding.

We can see in Figure 14 that only one call to the underlying block cipher is required to produce one ciphertext block, and one more to produce the final tag[7]. This makes OCB really efficient in both software and hardware implementation. The only drawback is that OCB is not resilient against a birthday paradox collision attack as pointed out by Niels Ferguson in [Fer02]. To thwart this attack, the authors limit the number of blocks of data processed under the same key to 2^{48}.

2.3.6 COLM v1

COLM is an Encrypt-LinearMix-Encrypt mode developed by Andreeva et al. [ABD+16b] and selected for use case 3 (defence in depth). This mode is based on an underlying block cipher, the AES with 128-bit key and ensures confidentiality and integrity even in misuse scenarios

COLM relies on the AES primitive, comprised in a mode involving layers of linear mixing and offset additions to perform encryption. The encryption process is depicted in Figure 15. The linear mixing is a linear function that outputs two different blocks computed from linear combinations of the input blocks and the offset values depend on the position of the current block of data. Remark that COLM is an online algorithm, meaning that the ciphertext and tag generation only requires one pass of the associated data and the plaintext data to be computed and that the plaintext can be processed

7 It also requires one additional call to the AES function to generate the first offset value.

intermittently. Indeed, additional blocks of plaintext can be appended in the flow since the generation of a new ciphertext block only depend of the linear mixing function output, which can be computed from the last ciphertext block.

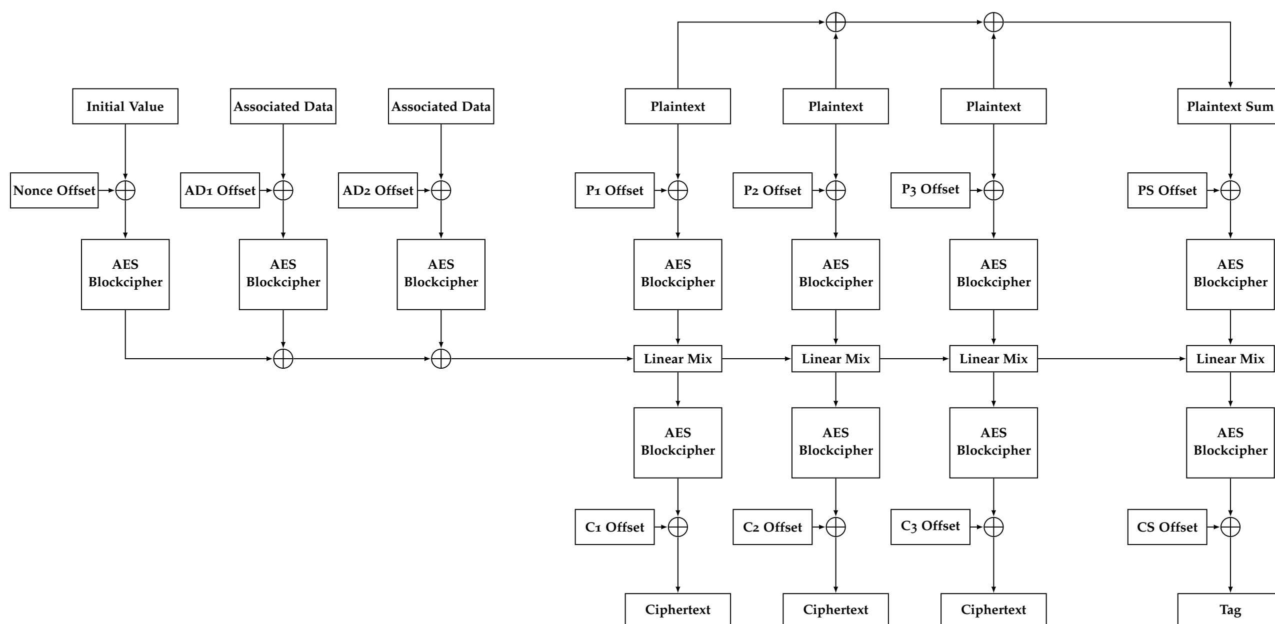

Figure 15: COLM encryption process without plaintext padding.

COLM takes its internal design from the best of the COPA [ABD+16a] and ELmD [DN16] designs, thus its security claims rely on the security analysis conducted on COPA and ELmD [ABD+]. Currently, no cryptanalysis threatens the security of COLM.

2.3.7 Deoxys-II v1.41

Deoxys [JNPS16] is an AEAD algorithm proposed by Jérémy Jean, Ivica Nikolić, Thomas Peyrin and Yannick Seurin that was selected for use case 3. Deoxys relies on a tweakable block cipher Deoxys-BC inspired by the AES and incorporated in two modes proposed by the authors. The first one, Deoxys-I, is an instantiation of θCB and must be used without nonce repetition, the second mode, Deoxys-II, is designed to target resilience in misuse scenarios. Only the latter, Deoxys-II, has been selected as finalist for defence-in-depth applications. Two instances of Deoxys-II are described by the authors that differ by the TBC used, Deoxys-BC-256 with a 128-bit key or Deoxys-BC-384 with a 256-bit key.

The mode used in Deoxys-II draws its inspiration from the *Synthetic Counter-in-Tweak* mode, SCT [PS15], which uses a tweakable block cipher to provide authenticated encryption with nonce and associated data. In Deoxys-II, the authors preserved the design of the encryption part of SCT, but modified the tag computation part "*in order to provide graceful degradation of security for authentification with the maximal number of repetitions of nonces*" [JNPS16]. Two steps are required to encrypt a message. First, a tag is generated from the nonce, plaintext and associated data. Then, this tag is used as a tweak of the TBC Deoxys-BC to compute the keystream blocks used to encrypt the plaintext. The encryption process of Deoxys-II is depicted in Figure 16. Remark that Deoxys is not online since two passes are required to produce the ciphertext and the tag.

While AES-GCM provides 64-bit security against nonce-misuse, Deoxys-II offers full 128-bit authentication security in nonce-misuse scenarios, making it really suitable for defence in depth, use case 3 of the CAESAR competition. Several cryptanalyses have been presented, targeting Deoxys with reduced variants of the TBC. The results are depicted in Table 3, note that only the results targeting variants of the TBC without modifying the key size are depicted, the main results of these cryptanalyses target TBC variants with a modified key size.

Table 3: Main cryptanalyses results on Deoxys.

Reference	TBC	Variant	Method	Time compl.	Data Compl.	Attack Mode
[CHP+17]	Deoxys-BC-256	9/14	Boomerang attack	2^{118}	2^{117}	Related-Key
[CHP+17]	Deoxys-BC-384	12/16	Boomerang attack	2^{127}	2^{127}	Related-Key
[mMS18]	Deoxys-BC-256	8/14	Impossible diff.	2^{118}	2^{118}	Single-Key
[mMS18]	Deoxys-BC-256	9/14	Impossible diff.	2^{118}	2^{118}	Related-Key
[ZDW18]	Deoxys-BC-256	9/14	Impossible diff.	$< 2^{128}$	2^{124}	Related-Key

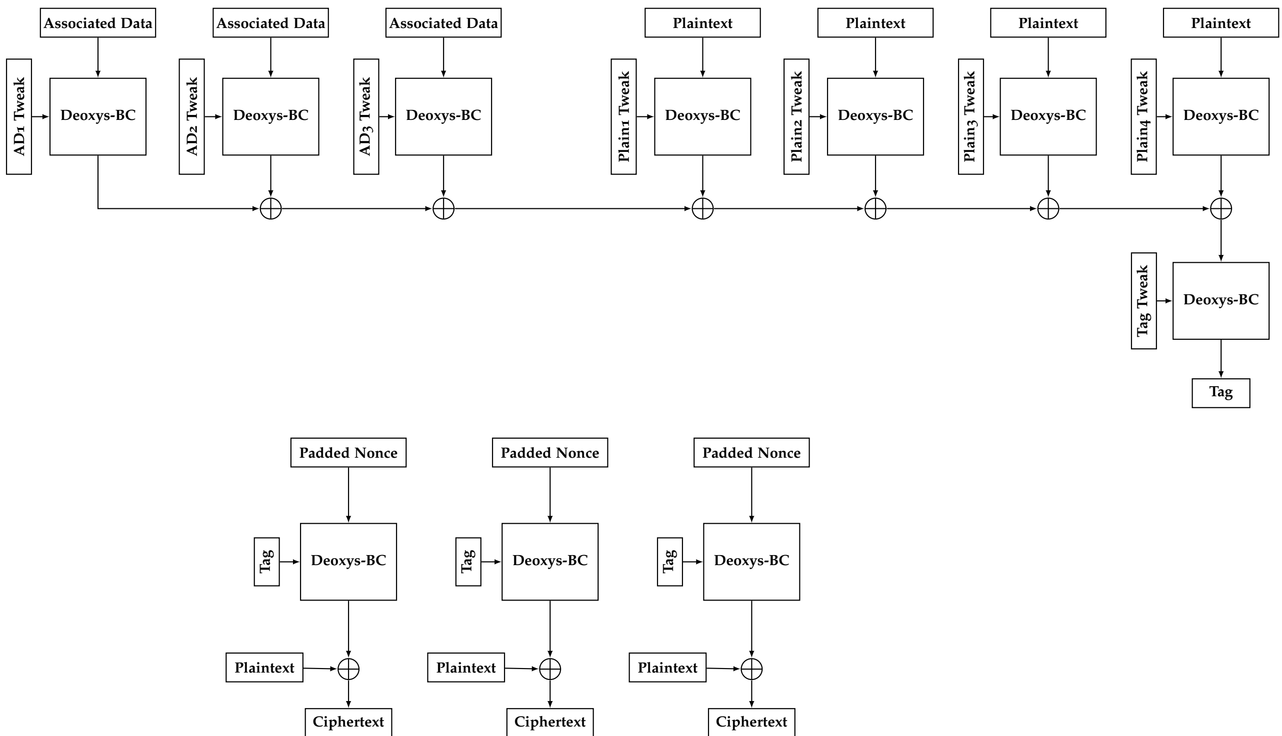

Figure 16: `Deoxys-II` encryption process with 4 plaintext blocks and 3 AD blocks. Tag generation (up) and ciphertext computation (down).

3 CRYPTANALYSIS OF AEZ

This chapter focuses on the AEZ candidate of the CAESAR competition, and presents a cryptanalysis developed in a joint work with Henri Gilbert that targets the latest version of the algorithm at that time, namely AEZ v4.1. Our results were published in the paper *Is AEZ v4.1 Sufficiently Resilient Against Key-Recovery Attacks?* [CG16] and presented at FSE 2017.

AEZ [HKR15b, HKR15a] is an AES-based AE scheme designed by Hoang, Krovetz, and Rogaway. AEZ is a high-profile CAESAR candidate and was selected in August 2016 for the third round of the competition. The AEZ construction can be viewed as a mode of operation of an underlying block cipher – more precisely of a mixture of AES versions with 4 and 10 rounds denoted AES4 and AES10. AEZ uses secret offsets and round keys derived from the authenticated encryption key K. AEZ is parallelizable and particularly well suited for software implementations on processors equiped with the AES-NI instruction set. On such environments, its computational cost is lower than the one of AES-GCM and close to the one of OCB [RBBK01]. AEZ aims at providing an unusually strong nonce and decryption misuse resistance and more generally best achievable security given the selected amount of plaintext expansion. These security properties are captured by the notion of *robust authenticated encryption* (RAE), a demanding security notion that comprises the MRAE notion defined in Section 1.3.3, not attainable by *online* AE schemes, i.e. AE schemes allowing a single-pass blockwise plaintext encryption with constant memory [HRRV15]. The RAE security notion and the security arguments underlying the AEZ construction were detailed in the Eurocrypt 2015 paper [HKR15b].

Related Work and Our Contributions. In this chapter, we analyze the resilience of AEZ v4.1 (October 2015) [HKR15a] that was selected for the third round of CAESAR, against key-recovery attacks. We show that the AEZ modifications introduced in 2015, that were partly motivated by thwarting a key derivation attack with birthday complexity against AEZ v3 published at Asiacrypt 2015 by Fuhr, Leurent and Suder [FLS15], do not prevent the existence of key derivation attacks of birthday complexity against AEZ v4.1. In both cases, the attack rests on the fact that AEZ was designed as to be po-

tentially usable either without nonce[1] or with a ptentially repeated nonce values without other security impact than the detectability of repeated (associated data, message) pairs. Unlike the attack of [FLS15], our most efficient key recovery attack relies on the use of AES4 in AEZ[2].

Neither the AEZ v3 attack of [FLS15] nor our attack on AEZ v4.1 violates the security claims of AEZ since the designers made no claim for beyond-birthday security. It should also be noted that if one takes into account the limitation of the amount of data processed under the same key to 2^{48} bytes required by the designers, their success probability becomes relatively low.

We nevertheless believe that the vulnerability of AEZ v4.1 to a key derivation attack of birthday complexity[3] represents an undesirable property, particularly for an algorithm that otherwise aims at satisfying a very strong notions of security and at being exceptionally resilient in various misuse situations. Indeed, even though the existence of distinguishers of birthday complexity against modes of operation of block ciphers is not so unusual, the existence of full key derivation attacks of birthday complexity is far less frequent and raises in the case of AEZ v4.1 the following resilience questions (exactly the same as those raised by the attack of [FLS15] in the case of AEZ v3). First, our attack allows to recover the whole key material with a much higher success probability than the one that would result from generic attacks for typical key sizes, e.g. 128, 256, or 384 bits even if the below-birthday data limitation of 2^{48} bytes imposed by the designers is respected. Second, this probability can become arbitrarily close to 1 in the algorithm misuse case where the data limitation of 2^{48} bytes cannot be enforced and "birthday" amounts of data can be processed.

Our results are summarized in Table 1. Our most efficient attack essentially consists of two phases. In a first phase, 128 bits of key material used for pre-whitening the inputs to some AES4 and AES10 computations are derived, using a birthday attack. We show that the universal hashing part of the AEZ computations, on which the attack of [FLS15] concentrated, can still be targeted by some birthday attacks. However the key material information this provides is less suited for continuing the attack than a 128-bit sub-key that can be derived by targeting instead the encipherment part of the AEZ computations. This sub-key determines the pre-whitening of some AES4 computations also involved the encipherment procedure. In a second phase

1 This is allowed by the specification, with the warning that "a nonce must be used unless one has certitude that, even in the presence of the adversary, all encrypted [(associated data, message)] pairs will be distinct[...]" [HKR15a].

2 It can therefore not be transposed to the more conservative but less efficient scaled up version of AEZ where only AES10 is used instead of a mixture of AES10 and AES4.

3 And to a resulting below-birthday attack of abnormally high success probability, as discussed below.

of the attack, we encrypt particular plaintext structures and detect plaintext pairs leading to a special differential behaviour in the last three rounds of these AES4 computations. This allows to recover the remaining of the key material.

Table 4: AEZ attacks complexities.

AEZ version	Data complexity (blocks)[4]	Success prob.	Ref.
AEZ v4.1	$2^{66.5}$	0.5	**Our work**
AEZ v4.1	2^{44}	$2^{-45.7}$	**Our work**
AEZ v3	$2^{66.6}$	1	[FLS15]
AEZ v3	2^{44}	$2^{-45.2}$	[FLS15]

Subsequent versions of AEZ. Two versions of AEZ have been introduced since AEZ v4.1:

– AEZ v4.2 (September 2016) the authors acknowledge the existence of key-recovery attacks with birthday-bound complexities. No algorithmic change.

– AEZ v5 (March 2017) has been introduced to rework the secret offsets computation after Leurent et al. [BDD+17] pointed out a bug in the tweakable block cipher definition. Two distinct tweak values may could result in the same offset value, destroying the authenticity of the AEZ construction. The modifications introduced in AEZ v5 do not attempt to thwart our attack.

Organization. The chapter is organized as follows. Section 3.1 outlines the parts of the AEZ v4.1 specifications that are useful for our attack and the main differences between AEZ v4.1 and AEZ v3. Section 3.2 first describes partial attacks of birthday complexity allowing to recover a 128-bit piece of the key material (Section 3.2.1). The combination of these partial attacks can be viewed as a suboptimal key derivation attack of birthday time and data complexity. Then we detail our most efficient key-recovery attack on AEZ v4.1 (Section 3.2.2), that exploits the use of AES4 in AEZ. The attack of Section 3.2.2 has the property (not shared by the suboptimal key-recovery attack of Section 3.2.1) that its success probability remains abnormally high

4 Chosen plaintexts mode.

if the amount of data processed under the same key is limited to the below-birthday threshold of 2^{48} bytes.

3.1 Description of AEZ

The following input and output arguments are used in AEZ:

– a plaintext P of *plen* bits;

– a key K of arbitrary length *klen* bits. The default value of *klen* is 384 bits and *klen* values of at least 128 bits are recommended;

– a nonce N of length *nlen* bits. The use of nonce values of length at most 128 bits is recommended and $nlen = 0$ is allowed, as well as the use of several nonce lengths for authenticated encryptions under the same key;

– a string-valued or more generally vector-valued associated data $A = (A_1, \cdots, A_m)$ of m strings, of total length *alen* bits. A string-valued associated data can be viewed as a vector with $m = 1$ components;

– a ciphertext C of *clen* bits.

Although their lengths are defined in bits, all these arguments are required to consist of an integer number of bytes.

AEZ is also parametrized by the authenticator byte length `Abytes` of default value 16. The corresponding number of bits $\tau = 8 \times \texttt{Abytes}$ represents the plaintext expansion $clen - plen$ and also the number of zero bits that shall be appended to the plaintext P before encipherment if P is not the empty string. The augmented plaintext $(P \,||\, 0^\tau)$ is denoted by $\overline{P}$ in the sequel and the binary representation of τ as a 128-bit word is denoted by $[\tau]_{128}$.

The AEZ authenticated encryption process can be viewed as follows. First a vector-valued tweak $T = ([\tau]_{128}, N, A_1, \cdots, A_m)$, that encodes the triplet (τ, N, A) is derived. Then, depending on the the plaintext length *plen*, differnet encipherment functions are applied:

– $\texttt{AEZ-prf}(K, T, \tau)$ is returned if $plen = 0$,

– $\texttt{AEZ-tiny}(K, T, \overline{P})$ is returned if $0 < plen < 256 - \tau$,

– $\texttt{AEZ-core}(K, T, \overline{P})$ is returned if $256 - \tau \leq plen$.

The way the tweak argument T is processed in all these functions consists of deriving an associated universal hash value $\Delta = \texttt{AEZ-hash}(K, T)$ of length

128 bits and then using Δ as an offset in some parts of the encipherment computations.

Since we do not use `AEZ-tiny` in our attack, we only describe `AEZ-prf` and `AEZ-core`.

3.1.1 Tweaked Instances of AES4 and AES10 Used in AEZ

AEZ uses AES-based tweakable block ciphers [LRW11] (TBC) using the XE and XEX constructions. Three sub-keys I, J, and L, of length 128 bits each, are used in these TBC, which are derived from the key K in a way that depends of the key length $klen$:

- if $klen = 384$, then $I \mid\mid J \mid\mid L = K$;
- if $klen \neq 384$, then $I \mid\mid J \mid\mid L = \mathtt{BLAKE2b}(K)$, using an instance of the cryptographic hash function BLAKE2b [AHMP10] that produces 384-bit hash values.

Given two input tweaks i, j, the TBC $E_K^{i,j}$ is defined as follows:

i	j	$E_K^{i,j}$	k
-1	$\mathbb{N}$	$E_K^{i,j} = \mathtt{AES10}_k(X \oplus jJ)$	$(0, I, J, L, I, J, L, I, J, L, I)$
0	$\mathbb{N}$	$E_K^{i,j} = \mathtt{AES4}_k(X \oplus jI)$	$(0, J, I, L, 0)$
1	$\mathbb{N}$	$E_K^{i,j} = \mathtt{AES4}_k(X \oplus \delta_j I)$	$(0, J, I, L, 0)$
2	$\mathbb{N}$	$E_K^{i,j} = \mathtt{AES4}_k(X \oplus \delta_j I)$	$(0, L, I, J, L)$
≥ 3	0	$E_K^{i,j} = \mathtt{AES4}_k(X \oplus \delta_i L) \oplus \delta_i L$	$(0, J, I, L, 0)$
≥ 3	$\mathbb{N}^*$	$E_K^{i,j} = \mathtt{AES4}_k(X \oplus \delta_i L \oplus \delta_j J) \oplus \delta_i L \oplus \delta_j J$	$(0, J, I, L, 0)$

where $\delta_i = 2^{i-3}$ and $\delta_j = 2^{3+\lfloor (j-1)/8 \rfloor} + (j-1) \bmod 8$. In the former table, AES4 (resp. AES10) are AES variants that consist of 4 (resp. 10) full AES rounds parametrized by 5 (resp. 11) independent sub-keys. Thus, if we denote the composition of `SubBytes`, `ShiftRows`, and `MixColumns` by `aesr` we get

$$\mathtt{AES4}_k(X) = \mathtt{aesr}(\mathtt{aesr}(\mathtt{aesr}(\mathtt{aesr}(X \oplus k_0) \oplus k_1) \oplus k_2) \oplus k_3) \oplus k_4),$$

with $k = (k_0, k_1, k_2, k_3, k_4)$, AES10 can be defined in the same way.

3.1.2 `AEZ-hash` universal hashing

To describe $\texttt{AEZ-hash}(K, T)$, we assume that the tweak $T = (\tau, N, A_1, \ldots, A_m)$, where $(A_1, \ldots, A_m)$ is a m-component vector. This allows to cover both the cases of string- and vector-valued associated data. Let us rewrite T as $T = (T_1, \ldots, T_t)$, where $t = m + 2$. For each m_i-block component $T_i = B_{i,1} \ldots B_{i,m_i}$ of T, whose last block B_{i,m_i} can be complete or incomplete, a partial hash value Δ_i is computed as follows:

$$\Delta_i = \begin{cases} E_K^{i+2,1}(B_{i,1}) \oplus E_K^{i+2,2}(B_{i,2}) \oplus \cdots \oplus E_K^{i+2,m_i-1}(B_{i,m_i-1}) \oplus E_K^{i+2,m_i}(B_{i,m_i}) & \text{if } |B_{m_i}| = 128 \\ E_K^{i+2,1}(B_{i,1}) \oplus E_K^{i+2,2}(B_{i,2}) \oplus \cdots \oplus E_K^{i+2,m_i-1}(B_{i,m_i-1}) \oplus E_K^{i+2,0}(B_{i,m_i} \,||\, 10^*) & \text{if } |B_{m_i}| < 128 \end{cases}.$$

Finally, $\texttt{AEZ-hash}(K, T) = \Delta \stackrel{\text{def}}{=} \Delta_1 \oplus \cdots \oplus \Delta_t$.

3.1.3 PRF Function

`AEZ-prf` is designed with the purpose to provide a PRF of settable output length τ, that can be viewed as an encipherment of the empty plaintext. The output of `AEZ-prf` is the τ-bit string given by

$$\texttt{AEZ-prf}(K, T, \tau) = (E_K^{-1,3}(\Delta) \,||\, E_K^{-1,3}(\Delta \oplus [1]_{128}) \,||\, E_K^{-1,3}(\Delta \oplus [2]_{128}) \,||\, \ldots)[1..\tau],$$

with $\Delta = \texttt{AEZ-hash}(K, T)$.

3.1.4 `AEZ` Core

`AEZ-core` is the encipherment function used to process augmented plaintexts of at least 256 bits. It takes as input the key K, the tweak vector T and the augmented plaintext $\overline{P}$. The vector T is first preprocessed by computing the universal hash value:

$$\Delta = \texttt{AEZ-hash}(K, T),$$

which will be used as an offset value at some subsequent steps of the encipherment computation.

Then, the augmented plaintext is split as follows into (in)complete 128-bit blocks:

$$\overline{P} = P_1 P_1' \,||\, P_2 P_2' \,||\, \cdots \,||\, P_m P_m' \,||\, P_u P_v \,||\, P_x P_y,$$

where $|P_*| = |P_*'| = 128$ except for at least one of the values P_u and P_v, that satisfy $|P_u| + |P_v| < 256$. In detail, to split $\overline{P}$ one needs to

1. Take the 256 last bits of $\overline{P}$ to form $P_x P_y$. This is always possible since $|\overline{P}| \geq 256$;

2. For every remaining pair of entire blocks if any form $P_i P_i'$ (starting from the beginning of $\overline{P}$);

3. Letting $r = plen + \tau \bmod 256$, if $r \neq 0$, the remaining bits form
   – P_u if $r < 128$,
   – $P_u P_v$ with an empty block P_v if $r = 128$
   – $P_u P_v$ with $|P_v| = r \bmod 128$ if $128 < r < 256$.

The ciphertext blocks are then computed as shown in Figure 17, up to the fact that in the first case ($r < 128$), the v-column is omitted and the paddings and compressions represented by trapezoids on the v-column are moved to the u-column.

For a more detailed description of `AEZ-core` and more generally on `AEZ` v4.1, we refer to the `AEZ` v4.1 specification [HKR15a].

3.1.5 Tweaks from `AEZ` v3

In a nutshell, the main differences between `AEZ` v3 and `AEZ` v4.1 are the following:

– the procedure for deriving the subkeys I, J, and L from the key K was entirely modified. The `AEZ` v3 derivation procedure, that did not involve the BLAKE2b hash function, had indeed the undesirable property that for key lengths such as $| K |= 128$ bits, the knowledge of one of the subkeys implied the knowledge of the key K. Moreover, while a key length of at least 128 bits is recommended in both `AEZ` v3 and `AEZ` v4.1, a default key length of 384 bits was introduced in `AEZ` v4.1;

– the tweakable block ciphers involved in the `AEZ-hash` universal hashing use the XEX construction in `AEZ` v4.1, whereas they were using the XE construction in `AEZ` v3. Moreover the offset values used in the definition of the various tweakable block ciphers $E_K^{i,j}$ used in AEZ were modified.

One of the motivations for these changes was to thwart the birthday attack on `AEZ` v3 introduced by Fuhr, Leurent, and Suder in 2015 [FLS15]. This attack indeed recovered one of the subkeys (namely J) by leveraging its use in the pre-whitening keys of the XE construction of the `AEZ-hash` computation underlying the `AEZ-prf` function. It then took advantage from the undesirable property of the `AEZ` v3 subkey derivation procedure mentioned above to recover the key K.

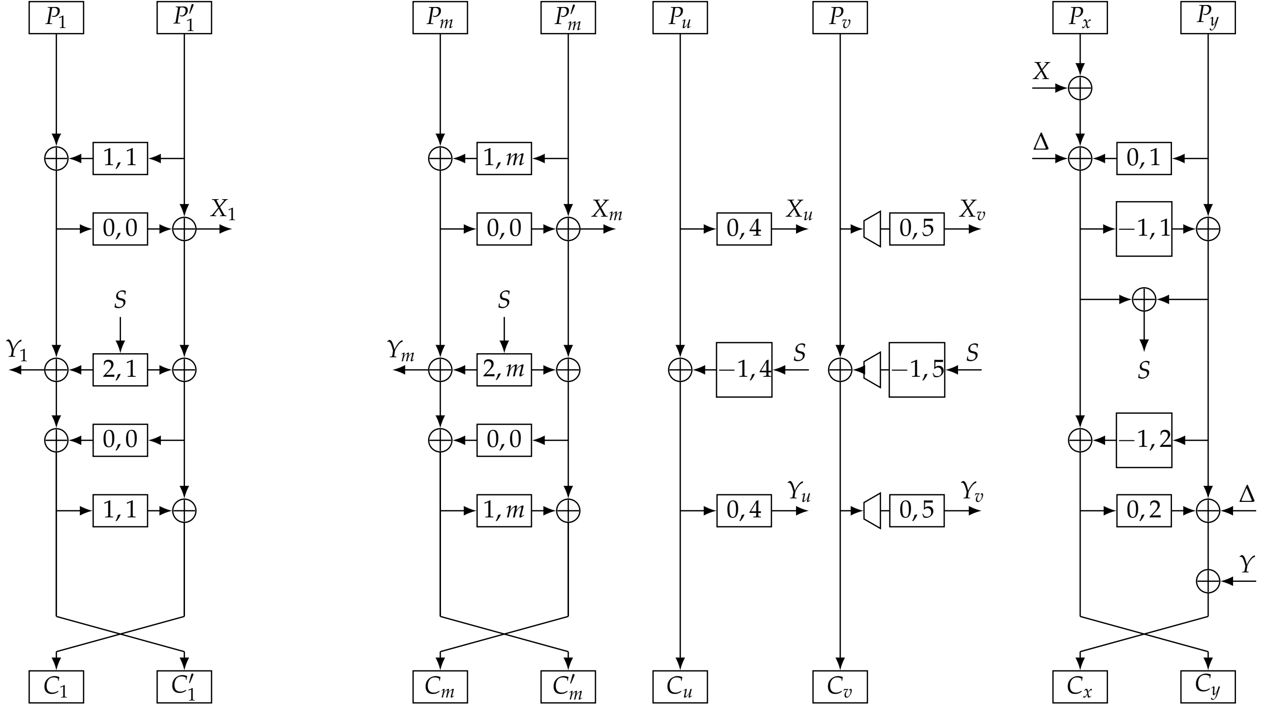

Figure 17: AEZ-core scheme.

$$
\begin{aligned}
\boxed{i,j} &= E_K^{i,j}(X) \\
X_i &= E_K^{0,0}(P_i \oplus E_K^{1,1}(P_i')) \ i \in [1..m] \\
X_u &= E_K^{0,4}(P_u) \\
X_v &= E_K^{0,5}(P_v 10^*) \\
X &= X_1 \oplus \cdots \oplus X_m \oplus X_u \oplus X_v \\
S &= \Delta \oplus X \oplus E_K^{0,1}(P_y) \oplus E_K^{-1,1}(\Delta \oplus X \oplus E_K^{0,1}(P_y)) \oplus P_y \\
Y_i &= P_i \oplus E_K^{1,1}(P_i') \oplus E_K^{2,1}(S) \ i \in [1..m] \\
Y_u &= E_K^{0,4}(P_u \oplus E_K^{-1,4}(S)) \\
Y_v &= E_K^{0,5}(P_v \oplus E_K^{-1,5}(S)[1..|P_v|] \ || \ 10^*) \\
Y &= Y_1 \oplus \cdots \oplus Y_m \oplus Y_u \oplus Y_v
\end{aligned}
$$

While the attack described by Fuhr et al. does not work anymore on AEZ v4.1, we will see in Section 3.2.1 that the use of the XEX construction in

AEZ-hash does not prevent birthday attacks, and Section 3.2.2 will show that the knowledge of I can be leveraged for recovering the other subkeys J and L.

3.2 Attacks on AEZ

In this section, we describe two key derivation attacks:

– First, a combination of three independent birthday attacks allowing to retrieve one of the sub-keys I, J, and L each. One limitation of this combined attack comes from the fact that the amount of data that can be processed under one single key is limited to 2^{48} bytes, below the 2^{64} blocks birthday bound. Its success probability, equal to the product of the success probabilities of the underlying birthday attacks, becomes in the case of a 128-bit key lower than the one of a generic attack.

– Second, a more efficient attack that consists of two phases. In the first phase, of birthday complexity, one of the three former partial attacks is applied to retrieve the value of I. In the second phase, the knowledge of I is leveraged to mount a differential attack against some of the AES4 instances of the encipherment computations. For any reasonable key length, e.g. at least 128 bits, its success probability remains abnormally high (i.e. higher than the one of a generic attack) if the amount of data that can be processed under one single key is limited to 2^{48} bytes.

3.2.1 Birthday Attacks

We describe in this subsection three partial attacks of birthday complexity each allowing to recover one of the three sub-keys.

All these partial attacks are based on the following informal observation. Let F and G denote two one-block to one-block functions parametrized by secret keys, δ_1 and δ_2 denote two secret one-block offset values and n denote the block length. Let us assume that an adversary is able to access $H(x) = G(F(x \oplus \delta_1) \oplus F(x \oplus \delta_2))$ for sufficiently many chosen block values x. Let us show that if a small multiple of $2^{\frac{n}{2}}$ values of x are tried this allows (under mild conditions on F and G that we will not detail here) to determine the secret offset difference $\delta_1 \oplus \delta_2$ with an overwhelming probability. Indeed, with overwhelming probability, there exists a pair (x, x') such that $x \oplus x' = \delta_1 \oplus \delta_2$. It is easy to see that for such a pair, $H(x) = H(x')$ since the single difference between the computations of $H(x)$ and $H(x')$ is that the entries of the first and second invocations of F are swapped. Conversely, if $H(x) =$

$H(x')$, $x \oplus x'$ provides a candidate value for $x \oplus x' = \delta_1 \oplus \delta_2$ that is easy to test using a few extra H computations.

Note that all attacks presented below can be conducted under the assumption of a fixed nonce length, i.e. $nlen = 128$. We emphasize that the attacks can be transposed, with slight adjustements, to a situation where the nonce is omitted (i.e. $nlen = 0$).[5]

Collisions in `AEZ-hash`

Detecting suitable collisions in the `AEZ-hash` function allows to recover the two sub-keys J and L by birthday attacks. While `AEZ-hash` is an internal procedure whose ouput is not directly available to the adversary, such collisions on `AEZ-hash` can nevertheless be detected by collisions they induce in some `AEZ-prf` output blocks.[6] Let Δ be the output of `AEZ-hash` under an unknown key and a chosen entry

$$\Delta = \texttt{AEZ-hash}(K, T) \text{ with } T = (\tau, N, A)$$

where we assume that $|A| = 128$. For simplicity, we also assume $\tau = 128$ bits, but the attack also applies to others value of τ.

Following the description of `AEZ-hash`, we get:

$$\Delta = E_K^{3,1}(\tau) \oplus E_K^{4,1}(N) \oplus E_K^{5,1}(A).$$

By replacing $E_K^{i,j}(X)$ by its expression we obtain:

$$\Delta = \texttt{AES4}_K(\tau \oplus L \oplus 8J) \oplus \texttt{AES4}_K(N \oplus 2L \oplus 8J) \oplus \texttt{AES4}_K(A \oplus 6L \oplus 8J) \oplus 8J \oplus 7L.$$

If we restrict ourselves to (A, N) pairs of blocks such that $A = N$, the former expression becomes

$$\Delta = \texttt{AES4}_K(\tau \oplus L \oplus 8J) \oplus \texttt{AES4}_K(N \oplus 2L \oplus 8J) \oplus \texttt{AES4}_K(N \oplus 4L \oplus 8J) \oplus 8J \oplus 7L.$$

5 The assumption that $nlen = 0$ was used in the `AEZ` v3 attack of [FLS15].

6 Collisions on `AEZ-hash` could alternatively be detected by collisions they induce on some `AEZ-core` output blocks. We will however not detail this slight variant here.

With this expression, we are able to create a collision on hash values Δ associated to (N, N) pairs and this can be used to retrieve the difference $6L$ between the first and second offset values applied to N. Indeed, if $N' = N \oplus 6L$, let us denote by Δ' the associated hash value. We have:

$$
\begin{aligned}
& \mathtt{AES4}_K(N' \oplus 2L \oplus 8J) \oplus \mathtt{AES4}_K(N' \oplus 4L \oplus 8J) \\
= \quad & \mathtt{AES4}_K(N \oplus 6L \oplus 2L \oplus 8J) \oplus \mathtt{AES4}_K(N \oplus 6L \oplus 4L \oplus 8J) \\
= \quad & \mathtt{AES4}_K(N \oplus 4L \oplus 8J) \oplus \mathtt{AES4}_K(N \oplus 2L \oplus 8J).
\end{aligned}
$$

Hence, when $N' = N \oplus 6L$, we have $\Delta = \Delta'$. Note that this can be viewed as a direct consequence from the former observation. Indeed, in the former expressions of Δ, N is added with the offsets $\delta_1 = 2L \oplus 8J$ and $\delta_2 = 4L \oplus 8J$, of difference L, before being input to the function $F = \mathtt{AES4}_K$.

Recovering the Sub-key L

The former remark allows to build the following birthday attack:

1. Collect $H(N) = \mathtt{AEZ\text{-}prf}(K, T, \tau)$ with $T = (\tau, N, N)$ for 2^{64} values of N,
2. If a collision occurs, this implies $H(N') = H(N)$ since `AEZ-prf` is just an overencryption of the `AEZ-hash` output and therefore $6L$ is likely to be equal to $N \oplus N'$.
3. L can be computed easily from $6L$ in $\mathbb{F}_{2^{128}}$

For this attack we need about $2^{64.2}$ pairs $(N, A = N)$ of input blocks to succeed with a probability of about 0.5. If the amount of input data is restricted to the limit of 2^{44} blocks imposed by the designers, the success probability drops to 2^{-43}.

Recovering the Sub-key J

The previous method can be used to retrieve J in a nonce-misuse scenario where a fixed nonce value N is repeated (which should result in no security degradation if the other `AEZ` input data are not repeated since an optimal nonce-misuse resistance is claimed). Indeed one can remark that using the previous notation except letting

$$T \quad = \quad (\tau, N, A, A),$$

where N is a fixed nonce value of length 128 bits, A is a variable one-block string, and the default value of 128 bits is assumed for τ. Using the description of AEZ-hash one can write

$$\begin{aligned}\Delta &= \mathtt{AES4}_K(\tau \oplus L \oplus 8J) \oplus \mathtt{AES4}_K(N \oplus 2L \oplus 8J) \oplus \mathtt{AES4}_K(A \oplus 4L \oplus 8J) \\ &\oplus \mathtt{AES4}_K(A \oplus 4L \oplus 9J) \oplus 3L \oplus J.\end{aligned}$$

.

Hence if $A' = A \oplus J$, one obtains

$$\begin{aligned}&\mathtt{AES4}_K(A' \oplus 4L \oplus 8J) \oplus \mathtt{AES4}_K(A' \oplus 4L \oplus 9J) \\ = \; &\mathtt{AES4}_K(A \oplus J \oplus 4L \oplus 8J) \oplus \mathtt{AES4}_K(A \oplus J \oplus 4L \oplus 9J) \\ = \; &\mathtt{AES4}_K(A \oplus 4L \oplus 8J) \oplus \mathtt{AES4}_K(A \oplus 4L \oplus 9J),\end{aligned}$$

which implies $\Delta = \Delta'$. One can then build the following attack to recover J.

1. Collect $H(A) = \mathtt{AEZ\text{-}prf}(K, T, \tau)$ with $T = (\tau, N, A, A)$ for 2^{64} values of A,
2. If a collision happens, this implies $H(A') = H(A)$ and therefore J is likely to be equal to $A \oplus A'$.

To reach $2^{64.2}$ queries and a probability of success of about 0.5 we need to run the algorithm with $2^{65.8}$ blocks of data. If the amount of input data is restricted to 2^{44} blocks, the recovery of J succeeds with probability $2^{-44.2}$.

3.2.1.1 Collision in AEZ-core

The last sub-key that remains unknown is I. We show how to recover it using a birthday attack within the AEZ-core function.

Let us consider 6-block augmented plaintexts:

$$\overline{P} = \overbrace{0^{128} \,||\, B}^{P_1, P_1'} \,||\, \overbrace{0^{128} \,||\, B}^{P_2, P_2'} \,||\, \overbrace{0^{128} \,||\, 0^{\tau}}^{P_x, P_y},$$

where B denotes a one-block string and $\tau = 128$ bits. In this partial attack, we assume that a fixed nonce value N and no associated data are used. Note that plaintext messages of length only five blocks are being used since the last block 0^τ corresponds to the ciphertext expansion.

Next, we denote by X the intermediate value associated with the two first pairs of blocks, that is used as an offset in the P_x, P_y part. We remind that $X = X_1 \oplus \cdots \oplus X_m$ in general. In our case, we have $X = X_1 \oplus X_2$ which, once developed, becomes

$$X = E_K^{0,0}(E_K^{1,1}(B)) \oplus E_K^{0,0}(E_K^{1,2}(B)) \oplus B \oplus B.$$

We can rewrite this expression as

$$X = \texttt{AES4}_K(\texttt{AES4}_K(B \oplus 8I)) \oplus \texttt{AES4}_K(\texttt{AES4}_K(B \oplus 9I)),$$

and notice that if $B' = B \oplus I$ then $X = X'$. This leads to a similar attack to the one on J and L. Indeed one can remark that collisions on the value of X induce collisions in the value of C_y since the single difference affecting the (P_x, P_y) part of the computation is the introduction of distinct values Y and Y', that only affect the value of C_x, not the value of C_y.

In summary, we search for collisions on the value of C_y to detect collisions on X (as shown in Figure 18).

The following steps describe the attack exploiting the preceding remark.

1. Collect $C_{y,B}$ from the encipherments $\texttt{AEZ-core}(K, T, \overline{P})$ associated to 2^{64} values of B.

2. If a collision occurs, i.e. $C_{y,B} = C_{y,B'}$, then I is likely to be equal to $B \oplus B'$ and this is easy to test using another value of B.

We need to encrypt $2^{66.3}$ blocks of data to expect a collision with probability 0.5. If the amount of input data is restricted to 2^{44} blocks, the recovery of I succeeds with probability $2^{-45.6}$.

3.2.1.2 Summary of Birthday Attacks

We have presented three partial attacks, each allowing to recover one of the three sub-keys I, J, and L. The following table summarizes the data complexity required by each attack for a success probability of 0.5 and their success probabilities if the amount of input data is limited to 2^{44} blocks. The time complexity of these attacks is equal to the time complexity for a single query multiplied by the query complexity.

The former partial attacks can be combined to recover the three sub-keys. However, when restricted to the encryption of 2^{44} blocks, this combined

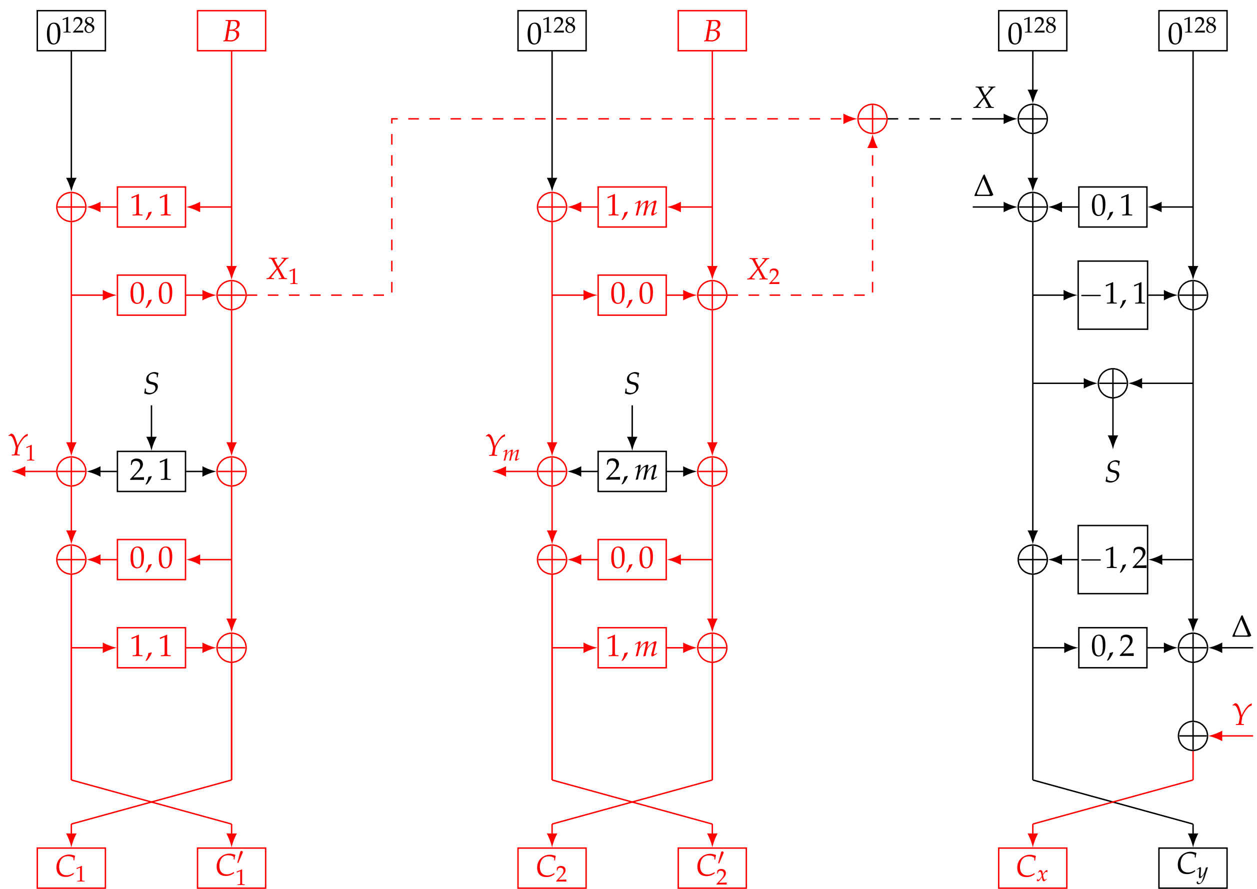

Figure 18: Difference propagation in the birthday attack to retrieve I.

attack succeeds, in the case of a 128-bit key, with a lower probability than a classical brute-force attack.

Table 5: Birthday attacks complexities.

Retrieved key	Data complexity (blocks)[a]	Queries	Success probability
I	$2^{66.5}$	$2^{64.2}$	0.5
I	2^{44}	$2^{41.7}$	$2^{-45.6}$
J	$2^{65.8}$	$2^{64.2}$	0.5
J	2^{44}	$2^{42.4}$	$2^{-44.2}$
L	$2^{65.2}$	$2^{64.2}$	0.5
L	2^{44}	2^{43}	2^{-43}
I, J, L[b]	$2^{68.1}$	$2^{65.8}$	0.5
I, J, L	2^{44}	$2^{42.5}$	$2^{-142.4}$

[a] Chosen plaintexts.
[b] In this row, the data and query complexities were derived as the sum of the data (resp. query) complexities for recovering I, J, and L with a probability of $0.5^{1/3}$ in three birthday attacks, as to ensure that the success probability is 0.5 for the combined attack.

3.2.2 `AES4` Cryptanalysis

We previously described partial attacks allowing to recover one of the three sub-keys used in `AEZ` and a resulting combined attack. We now describe an attack which, assuming the sub-key I has been retrieved using the last partial attack presented before, allows to efficiently recover the two others sub-keys J and L.

As in the partial attack allowing to recover I, we assume that $\tau = 128$ and we use a fixed nonce value N and no associated data in all considered encryptions.

3.2.2.1 Conducting Idea

Mounting a differential attack that targets the first `AES4` encryption of the P_u part of the `AEZ-core` function allows to leverage the knowledge of I to recover J and L. Since I is known, this eventually boils down to attacking only three AES rounds instead of four. Although the differential cryptanalysis of 3-round AES is simple and well studied, the context of the attack for `AEZ` is

more constrained and requires dedicated analysis. We therefore describe in some detail how to take these constraints into account.

Let

$$\overline{P} = P_u \mid\mid \underbrace{0^{128} \mid\mid 0^{\tau}}_{P_x, P_y},$$

where P_u denotes a 128-bit block. Since $(plen + \tau) \bmod 256 = 128$, an empty block P_v is introduced in the computation of X, that we denote by $R = E_K^{0,5}(1 \mid\mid 0^{127})$.The resulting offset value X is:

$$\begin{aligned} X &= E_K^{0,4}(P_u) \oplus R \\ &= \texttt{AES4}_K(P_u \oplus 4I) \oplus R. \end{aligned}$$

The detailed computation of X is summarized in Figure 19.

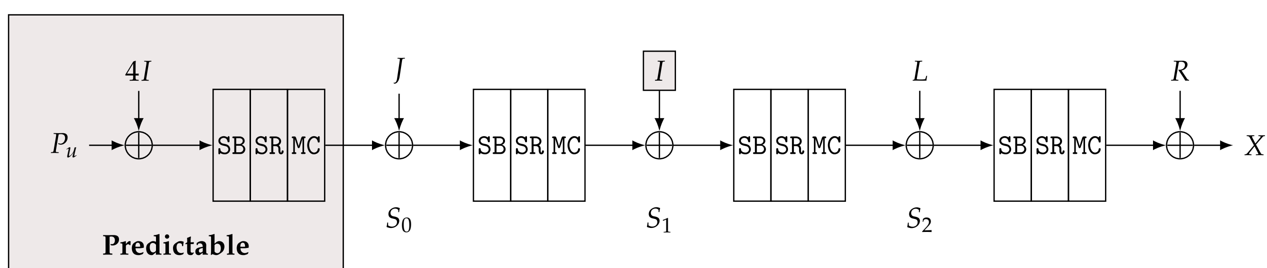

Figure 19: AES4 scheme.

To obtain information on J and L, one can search pairs of P_u values whose differential behaviour in the three last rounds of the AES4 computation follows a 4-1-4 differential characteristic. In other words, at the input to the second, third, and fourth rounds, we want the associated state values to only differ on 4 bytes, resp. 1 and 4 bytes, as shown in Figure 20.[7]

We are using the numbering convention of Figure 21 for the AES state bytes.

7 We would like to acknowledge the work of Jérémy Jean with his repository “TikZ for Cryptographers” [J´] for providing inspiration for the AES figures.

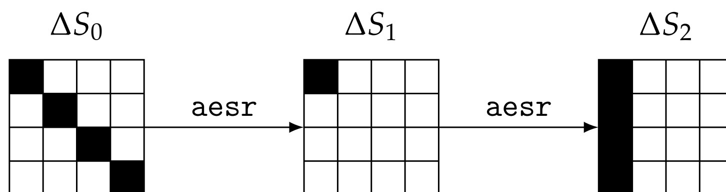

Figure 20: Differential path.

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

Figure 21: Bytes numbering in AES state.

Let us denote by $\Delta(x_1, \ldots, x_l)$ the vector space of difference values equal to zero everywhere outside from the byte positions $x_1, \ldots, x_l$. The expected differential behaviour at rounds 2, 3, and 4 is the following.

$$
\begin{aligned}
\text{AES round 2}: &\quad \Delta(0,5,10,15) \xrightarrow{\texttt{SB SR}} \Delta(0,1,2,3) \xrightarrow{\texttt{MC}} \Delta(0) \\
\text{AES round 3}: &\quad \Delta(0) \xrightarrow{\texttt{SB SR MC}} \Delta(0,1,2,3) \\
\text{AES round 4}: &\quad \Delta(0,1,2,3) \xrightarrow{\texttt{SB SR}} \Delta(0,7,10,13) \xrightarrow{\texttt{MC}} \\
\text{Output}: &\quad \texttt{MC}(\Delta(0,7,10,13)).
\end{aligned}
$$

Let (S_0, S_0') denote a pair of chosen second round input values before the addition of J, of difference $S_0 \oplus S_0' \in \Delta(0,5,10,15))$ and δ_x denote a difference value from $\texttt{MC}(\Delta(0,7,10,13))$. (S_0, S_0') can be derived from the chosen pair of P_u values ($P_u = \texttt{aesr}^{-1}(S_0) \oplus 4I, P_u' = \texttt{aesr}^{-1}(S_0')) \oplus 4I$) and one can test whether the differential behaviour of this pair is the desired one and the resulting AES4 output difference is equal to δ_x.

Indeed, let (P_x, P_x') be a pair of P_x blocks of difference $P_x \oplus P_x' = \delta_x$ and $P_y = 0$, and let

$$
\begin{aligned}
(C_v, C_x, C_y) &= \texttt{AEZ-core}(K, T, (P_u, P_x, P_y)) \\
(C_v', C_x', C_y') &= \texttt{AEZ-core}(K, T, (P_u', P_x', P_y)).
\end{aligned}
$$

Then we get $C_y = C'_y$ since the differences on the X offset values and on the P_x values cancel out. In Figure 22, the difference propagation is represented by the red pattern. Note that $C_y = C'_y$ happens with a negligible probability of about 2^{-128} if the tested condition is not met.

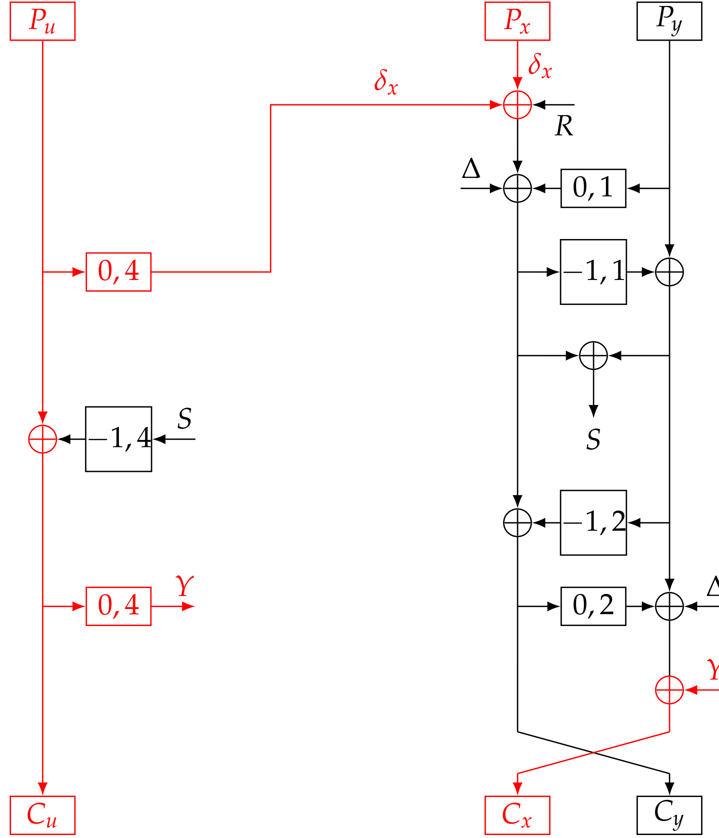

Figure 22: Difference propagation within `AEZ-core`.

We now show that the use of appropriate structures of (P_u, P_x) values – obtained as the cartesian product of smaller structures of P_u and P_x values – allows to efficiently get $C_y = C'_y$ collision.

3.2.2.2 Structure of P_u values

We want to test at least one pair (S_0, S'_0) of difference value $S_0 \oplus S'_0 \in \Delta(0, 5, 10, 15)$ that leads after the second round to a difference value that belongs to the set $\Delta(0)$. A simple heuristic argument indicates that testing about 2^{24} (S_0, S'_0) pairs should suffice for this to happen. Since picking S_0 and S'_0 from a subset of $\mathcal{S}$ of $\Delta(0, 5, 10, 15)$ of size 2^m, $m \leq 16$, allows to cover approximately 2^{2m-1} such pairs, selecting a structure $\mathcal{S}$ of $2^{12.5}$ such values

can be expected to suffice to obtain a good pair with a sufficient probability.

The resulting structure of P_u values that we use in the sequel is $\mathcal{U} = \texttt{aesr}^{-1}(\mathcal{S}) \oplus 4I$. We expect at least one pair of $\mathcal{U}$ elements to have the expected differential behaviour.

3.2.2.3 Structures of P_x and P'_x values

At the output of the AES4 function we want the difference to belong to the image of $\Delta(0,7,10,13)$ by MixColumns. Since MixColumns is a linear operation, such a difference, denoted δ_x, can be expressed as follows,

$$\delta_x = \delta_{x,(0,7)} \oplus \delta_{x,(10,13)}$$

with $\delta_{x,(0,7)} \in \texttt{MC}(\Delta(0,7))$ and $\delta_{x,(10,13)} \in \texttt{MC}(\Delta(10,13))$.

This decomposition, in combination with the previous structure, allows to reduce by a squared factor the sets of tested P_x and P'_x values. This is explained in the next section.

3.2.2.4 How to Find a Good Pair

We can use cartesian products of the structures defined above to find a collision with an improved data complexity.

We encrypt the plaintexts associated with the two following structures of (P_u, P_x) pairs:

$$\begin{aligned}(P_u, P_x) &\in \mathcal{U} \times \texttt{MC}(\Delta(0,7)) \\ (P'_u, P'_x) &\in \mathcal{U} \times \texttt{MC}(\Delta(10,13)).\end{aligned}$$

We call *observation* the block C_{y,P_u,P_x} resulting from the encryption of the plaintext $P_u \mid\mid P_x \mid\mid 0^{128}$. With the previous notations, one can remark that if $C_{y,P_u,P_x} = C_{y,P'_u,P'_x}$ then with overwhelming probability:

$$E_K^{0,4}(P_u) \oplus P_x = E_K^{0,4}(P'_u) \oplus P'_x$$

or equivalently

$$E_K^{0,4}(P_u) \oplus E_K^{0,4}(P'_u) = P'_x \oplus P'_x.$$

By construction, P_u and P'_u values are selected in such a way that the resulting round 2 input difference $\delta_{in} = \texttt{aesr}(P_u \oplus 4I) \oplus \texttt{aesr}(P'_u \oplus 4I)$ belongs to $\Delta(0,5,10,15)$, and P_x and P'_x values were selected in such a way that their difference $\delta_{out} = P_x \oplus P'_x$ can take any value from $\Delta(0,7) \oplus \Delta(10,13) = \Delta(0,7,10,13)$.

Therefore we can expect at least one equality $C_{y,P_u,P_x} = C_{y,P'_u,P'_x}$ to happen and with overwhelming probability the underlying (P_u, P'_u) pair is a good pair of second round input difference δ_{in} and fourth round output difference δ_{out}.

On can also note that this method can be extended to the following differential patterns.

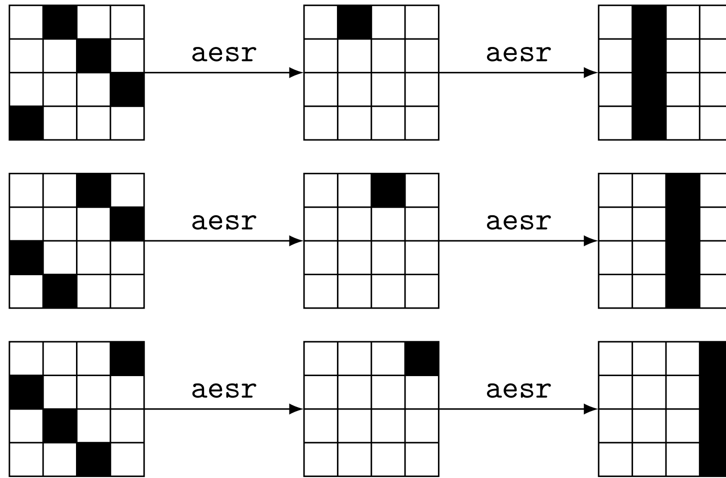

Figure 23: Other possible differential characteristics.

3.2.2.5 Sub-keys Recovery

Once a collision $C_{y,P_u,P_x} = C_{y,P'_u,P'_x}$ occurs, we obtain a good pair (P_u, P'_u) with a known AES4 output difference $\delta_{out} = P_x \oplus P'_x$. This can be used to retrieve information on the sub-keys J and L.

We know that the sub-key J has the property to allow a differential transition

$$\Delta(0,5,10,15) \xrightarrow{\texttt{SB SR MC}} \Delta(0)$$

in the second round. Let us denote by $\bar{J}$ a candidate value for J which leads to such a differential transition.

To each of the possible differences in $\Delta(0)$ we can associate a difference in $\Delta(0,5,10,15)$ by $\texttt{MC}^{-1} \circ \texttt{SR}^{-1}$. We denote such a difference by δ_{mid}. There are 255 possible values δ_{mid}.

Let $S_0 = \texttt{aesr}(P_u \oplus 4I)$, and $S'_0 = \texttt{aesr}(P'_u \oplus 4I) = S_0 \oplus \delta_{in}$. For a given trial value δ_{mid} we want to find $\overline{J}$ values that satisfy

$$\texttt{SB}(S_0 \oplus \overline{J} \oplus \delta_{in}) = \texttt{SB}(S_0 \oplus \overline{J}) \oplus \delta_{mid}.$$

With a variable substitution $X = S_0 \oplus \overline{J}$ this amounts to finding X such that

$$\texttt{SB}(X \oplus \delta_{in}) = \texttt{SB}(X) \oplus \delta_{mid}.$$

Let us denote by B_i the i-th byte of a block B and by sbox the AES S-box. The former conditions amount to finding X_0, X_5, X_{10}, X_{15} such that

$$\texttt{sbox}(X_i \oplus \delta_{in,i}) = \texttt{sbox}(X_i) \oplus \delta_{mid,i}, i = 0,5,10,15.$$

We know that at least one $\overline{J}$, the actual sub-key J, fulfils these conditions. But one can expect a larger set of candidates to fulfil these conditions, 275 in average (as shown in Section 3.2.2.6) and another step is thus required to retrieve the right candidate. We expect to test about $275^4 = 2^{32.4}$ values to find J.

Assuming that we have collected the candidates for the four 4-byte parts of J, namely $\overline{J_i}$, we can retrieve the right value of J by the following method (similar to the method used in the birthday attacks)

1. Compute all the $\Delta_i = \texttt{AEZ-prf}(K, (\tau, N, \overline{J_i}, \overline{J_i}), \tau)$ and the reference value $\Delta = \texttt{AEZ-prf}(K, (\tau, N, 0^{128}, 0^{128}), \tau)$.
2. Find the value Δ_m such that $\Delta_m = \Delta$. The right value for J is then $J = \overline{J_m}$. This is due to the former observation used for the birthday attacks.

Once J is recovered, one can apply a similar one-round differential technique to recover L. Indeed, for good pairs, the input values to the fourth round $S_2 = \texttt{aesr}(\texttt{aesr}(\texttt{aesr}(P_u \oplus 4I) \oplus J) \oplus I$ and $S'_2 = \texttt{aesr}(\texttt{aesr}(\texttt{aesr}(P'_u \oplus 4I) \oplus J) \oplus I$ are known, their difference $\delta_{mid} = S_2 \oplus S'_2 \in \Delta(0,1,2,3)$ is known, and the AES4 output difference δ_{out} is known. The latter difference

induces a known difference value $(\texttt{SR}^{-1} \circ \texttt{MC}^{-1})(\delta_{out})$ after the fourth round SubBytes. Since the differences before and after SubBytes are completely fixed, only about 16 candidates values in average will satisfy

$$\texttt{SB}(X \oplus \delta_{mid}) = \texttt{SB}(X) \oplus (\texttt{SR}^{-1} \circ \texttt{MC}^{-1})(\delta_{out}).$$

By using the other differential transitions the remaining 12 bytes of L can be completely recovered and the sub-key L is found by testing about 2^{16} candidates for L. A similar method to the one used to find the right value of J, based on the former observation used for the birthday attack, can be used in order to find the right value of L.

3.2.2.6 Computation of the Average Number of Candidates for a Quartet of Bytes of Sub-key J

If we let a, b represent two non-zero random one-byte differences, then the equation $\texttt{sbox}(X \oplus a) = \texttt{sbox}(X) \oplus b$ may have 0, 2 or 4 solutions (sbox corresponds to the AES S-box). These number of solutions stand with their respective probabilities which are:

$$\#\{X \mid \texttt{sbox}(X \oplus a) = \texttt{sbox}(X) \oplus b\} = \begin{cases} 0 & \text{with } p_0 = \frac{128}{255} \\ 2 & \text{with } p_2 = \frac{126}{255} \\ 4 & \text{with } p_4 = \frac{1}{255} \end{cases}.$$

For a random pair $\delta = (\delta_1, \delta_2, \delta_3, \delta_4), \delta' = (\delta'_1, \delta'_2, \delta'_3, \delta'_4)$ of quartets of non-zero difference bytes, the average number of solutions of

$$\texttt{sbox}(X \oplus \delta_i) = \texttt{sbox}(X) \oplus \delta'_i \text{ for } i = 1, 2, 3, 4$$

is given by

$$(2p_2 + 4p_4)^4 \simeq 1.015.$$

Hence, out of the 255 possible pairs $(\delta_{in}, \delta_{mid})$, we can expect 254 of them to bring an average of $254 \times 1.015 = 257.8$ candidates since they are not expected to exhibit the right guess on J. For the last one we know it will bring the right guess of J, at least one solution will be obtained. The previous expression for the average number of solutions has to be slightly modified and becomes

$$\left(2 \times \frac{126}{127} + 4 \times \frac{1}{127}\right)^4 \simeq 16.5.$$

The former heuristic reasoning shows that we can expect to have to test an average of about 275 candidates for each differential transition.

3.2.2.7 Algorithm and Complexity to Recover J and L

In summary, the following algorithm allows to find the values of J and L assuming that I is known.

1. Compute the observations C_{y,P_u,P_x} associated with all pairs $(P_u, P_x) \in \mathcal{U} \times \texttt{MC}(\Delta(0,7))$.
2. Compute the observations C_{y,P'_u,P'_x} associated with all pairs $(P'_u, P'_x) \in \mathcal{U} \times \texttt{MC}(\Delta(10,13))$.
3. Find (P_u, P'_u, P_x, P'_x) such that $C_{y,P_u,P_x} = C_{y,P'_u,P'_x}$ and compute $\delta_{in} = \texttt{aesr}(P_u \oplus 4I) \oplus \texttt{aesr}(P'_u \oplus 4I)$, $\delta_{out} = P_x \oplus P'_x$.
4. Repeat Steps 1,2 and 3 for the three other differential transitions as to finally either get $(\delta^1_{in}, \delta^1_{out}), (\delta^2_{in}, \delta^2_{out}), (\delta^3_{in}, \delta^3_{out})$ or $(\delta^4_{in}, \delta^4_{out})$ for each good pair.
5. For each good pair, compute the about 275 candidate quartets of J bytes that are compatible with δ^i_{in}.
6. Test with `AEZ-prf` all the candidate values to find J.
7. For each good pair, compute the candidate quartets of L bytes that are compatible with the δ^i_{out}.
8. Test with `AEZ-prf` all the candidate values to find L.

To compute the complexity of our attack, we need to compute the cost of each step

- **Step 1 & 2** : We need to go through $\mathcal{U} \times \texttt{MC}(\Delta(0,7))$ and $\mathcal{U} \times \texttt{MC}(\Delta(10,13))$ to compute all the observations. This costs $2 \times |\mathcal{U}| \times |\texttt{MC}(\Delta(0,7))| = 2 \times 2^{12.5} \times 2^{16} = 2^{29.5}$ queries of 2 blocks i.e. $2^{31.5}$ blocks have to be encrypted.
- **Step 3** : Finding a collision can be achieved with a time complexity of about $2^{33.3}$. This is a computational cost, so the query complexity is not affected.
- **Step 4** : $4 \times 2^{31.5} = 2^{33.5}$ blocks have to be encrypted.
- **Step 5** : With pre-computation of all solutions for $\texttt{SB}(X \oplus \delta_{in}) = \texttt{SB}(X) \oplus \delta_{mid}$ with any $\delta_{in}, \delta_{mid}$ the candidates are easily computed with a time complexity of 2^{24}. As for Step 3 this phase does not require additional queries.

- **Step 6** : We need to compute $\texttt{AEZ-prf}(K, (\tau, X, X), \tau)$ for $275^4 = 2^{32.4}$ values of X i.e. $2 \times 2^{32.4} = 2^{33.4}$ blocks have to be encrypted.

- **Step 7** : No more cost since the pre-computation used in Step 5 can be reused here.

- **Step 8** : We need to compute $\texttt{AEZ-prf}(K, (\tau, N, X, X), \tau)$ for 2^{16} values of X i.e. $3 \times 2^{16} = 2^{16.6}$ blocks have to be encrypted.

The final cost to find J and L is given in Table 6 below.

Table 6: AES4 attack complexities.

Data complexity (bytes)	Offline time complexity	Queries complexity
$2^{34.6}$	$2^{33.3}$	$2^{32.1}$

This part of the attack was successfully validated on the public implementation of AEZ v4.1. This allowed to confirm that J and L can be recovered once I has been recovered.

3.2.3 Results of Our Attack

As described our attack works in two phases : first, find the sub-keys I by a birthday attack, and then, recover the two other sub-keys J and L by attacking AES4. Since the number of queries needed in the attack is far greater than the offline time complexity, the latter is insignificant in comparison of other costs and so, not included in the complexity of our attack. The final cost of our attack, depending on whether the data limit of 2^{48} bytes is respected or not, is given in the following Table 7

Table 7: Full attack complexities.

Data complexity (blocks)[a]	Queries complexity	Success probability
2^{44}	$2^{41.7}$	$2^{-45.6}$
$2^{66.5}$	$2^{64.2}$	0.5

[a] Chosen plaintexts.

3.3 Conclusion

One of the purposes of the modifications between AEZ v3 and AEZ v4.1 was to fix an undesirable property allowing to recover the whole key from one of the sub-keys used in AEZ. Our work shows that this property remains despite the changes. We describe a key-recovery attack that allows to recover the three sub-keys from the knowledge of only one. These modifications were also partly motivated by thwarting an attack of birthday complexity allowing to recover one of the subkeys. We described three birthday attacks on AEZ v4.1 allowing to retrieve one of the three sub-keys.

Even though no claim for beyond birthday security has been made and our attack does not violate the security claims for AEZ, it raises some doubts regarding the resilience of AEZ against key-recovery attacks when the amount of processed data approaches the birthday bound.

4 CRYPTANALYSIS OF NORX

This chapter presents a cryptanalysis of the NORX algorithm developed jointly with Thomas Fuhr, Henri Gilbert, Jérémy Jean and Jean-René Reinhard. Our results were published in the paper *Cryptanalysis of NORX v2.0* [CFG+17] and presented at FSE 2017.

NORX is a family of AEAD algorithms designed by Aumasson, Jovanovic and Neves, and is one of the 15 CAESAR candidates that were selected in August 2016 for the third round of the competition. The overall structure of the NORX algorithm adopts the so-called monkeyDuplex construction, which is derived from the sponge construction and iterates a keyless permutation P of a large state [BDPV12]. The design of the permutation P used by NORX is partly inspired from those of the stream cipher ChaCha [Ber08], the SHA-3 finalist BLAKE [AHMP10] and its more efficient variant BLAKE2 [ANWW13]. This permutation operates over states that can be represented as 4×4 matrix of words whose size w is either 32 or 64 bits. It follows a design close to so-called ARX primitives, as it uses only "R" operations (circular rotations and shifts), "X" (exclusive or) operations, and modified "A" operations (modular additions, modified in that carries only propagate to one position to the left). The key length k, the default tag length t, and the claimed security level of NORX are all equal to $4w$, in other words either all equal to 128 bits or to 256 bits depending on the value of w.

Three consecutive versions of the NORX specification were published by the designers during the CAESAR competition : NORX v1.0 (March 2014), the initial submission to the CAESAR competition; NORX v2.0 (August 2015), the version that was evaluated and selected for the third round; NORX v3.0 (September 2016), a version published shortly after the beginning of the third round that served as a basis for the third-round evaluation. In all versions, the NORX family consists of two main sub-families of algorithms associated with the word sizes $w = 32$ and $w = 64$.

Lightweight variants of NORX [AJN15c], called NORX-8 and NORX-16, have also been proposed by the same designers apart from the CAESAR competition. They follow the same generic strategy as NORX v2.0, with word sizes $w = 8$ and $w = 16$.

Related Work. There exists a handful of papers that study the security of NORX, which we briefly recall here. First, the designers of NORX provided their own analysis of the permutation P in the specifications and [AJN15a]. They conclude that no high-probability differential exists in the primitive, that word-level rotational cryptalysis does not threaten the scheme, and that no structural distinguisher of the permutation can be used in an attack on the mode. Later in [DMM15], Das et al. describe statistical variants of zero-sum distinguishers that allow to distinguish 3.5 rounds of the permutation of NORX-32 and the full-round permutation of NORX-64 from random permutations. At FSE 2016, Bagheri et al. show in [BHJ+16] that the slow diffusion of G^{-1} can be leveraged into a state/key recovery for a reduced versions of NORX v2.0 where the underlying permutation applies half the rounds (two out of four). More recently, Dwivedi et al. [DKM+16] analyze the state-recovery resistance of several CAESAR candidates, including NORX, with respect to SAT solvers. About NORX, they conclude that state recovery is only possible on NORX-32 when the underlying permutation does not apply more than 1.5 round. Finally, throughout this chapter, we also refer to [JLM14] where Jovanovic et al. give a security proof of the NORX mode.

Our Contributions. Our main result is an attack on NORX v2.0 that shows that the security level of the NORX v2.0 algorithms is at most $2w + 2$ bits, i.e. about 66 or 130 bits depending whether $w = 32$ or $w = 64$, instead of the $4w$ bits claimed by the designers. The attack can be viewed in two ways:

1. as an *existential forgery attack* with success probability 2^{-2w-2}, for instance 2^{-66} if $w = 32$ bits, that requires to get the authenticated encryption of one single short chosen plaintext or,
2. as an existential forgery attack with success probability greater than 50% that requires the knowledge of 2^{2w+2} ciphertexts with their associated tags and the same number of forgery attempts.[1]

Both variants of the attack break the claim of the designers stating that NORX v2.0 offers a $4w$-bit level of security. We additionally observe that once a forgery attempt succeeds, a key-recovery attack can be easily mounted as the secret key is only injected during the initialization phase. Namely, a successful forgery can reveal the full internal state at the expense of an extra offline computation of about 2^{2w} operations. Then, same as was done in [DJ15] on the FIDES authenticated encryption scheme [BBK+13], the full sponge can be inverted, which reveals the initial state that contains the secret key. This can be achieved if we assume either chosen-plaintexts attacks, or that a ciphertext-only adversary interacts with a decryption oracle and gets

1 Enforcing a limitation of the amount of data handled with a single key does not thwart our attack as changing the key does not drop the marginal success probability of a single forgery attempt.

the plaintext corresponding to any successful forgery. We have implemented and experimentally verified the correctness of the attacks on a toy version of NORX v2.0, where the word size w is reduced to 8 bits.

The attack leverages an interaction between the two following properties of NORX v2.0, due to non-conservative designers choices.

– The *capacity* of the NORX sponge is low: only $4w$ bits, one fourth of the state size, i.e. 128 bits if $w = 32$ bits and 256 bits if $w = 64$ bits. This more aggressive choice than the $6w$-bit capacity that was selected for NORX v1.0 was motivated by performance considerations, as it allowed to increase the rate of the sponge construction by a factor 1.25. It was also supported by the security bounds derived from the security proofs of [JLM14] (substantiated in the security goals section of the algorithm specification [AJN15b]), up to the fact that the underlying permutation P does not behave like an ideal permutation.

– The permutation P used in the NORX sponge has strong structural properties that substantially deviate from the expected behavior of an ideal permutation. Our attack leverages the structural property that P commutes with a circular rotation of the columns of the internal state 4×4 matrix. This property has some connection with the weaker structural property of P already observed by the designers in [AJN15a] that the set of states whose four columns are equal is invariant under P.

The former attack can be viewed as a kind of rotational cryptanalysis at the state level rather than at the word level as considered in [AJN15a] on NORX or more generally in [KN10]. It also has some connection with the *invariant permutation attacks* sub-class of invariant subspace attacks introduced in [LMR15], since in both cases a permutation of the state words that commutes with a cryptographic state permutation is leveraged, one difference being that an invariant permutation property is used here in a keyless and constant-less context.

While the two non-conservative properties leveraged by the attack (the low capacity and the existence of a commuting rotation) still hold for NORX v3.0, one of the "tweaks" introduced in NORX v3.0 appears to thwart the former attack, namely the involvement of the key in the finalization of the tag computation, using an Even-Mansour-like construction. This finalization was also selected during the conception of another monkeyDuplex-based CAESAR candidate, namely Ascon [DEMS].

Finally, we also investigate the security claims of NORX v2.0 and NORX v3.0. We show that the generic success probability of a forgery attack has to be related to the cumulative length of forgery attempts (instead of the total

number of forgery attempts), as it is also the case for other AEAD schemes such as GCM. Moreover, we show that even if the total length of decryption queries is strongly limited, the authenticity bound of the proof does not guarantee the security level of 2^{4w} claimed for NORX.

Organization. The rest of this chapter is organized as follows. Section 4.1 gives detailed descriptions of NORX variants discussed in this chapter. In Section 4.2, we describe our attack on NORX v2.0. In Section 4.4, we study the applicability of our attack to other variants of NORX. Finally, in Section 4.5, we discuss the results of our attack and compare them to the security claims made by the designers of NORX and to the bounds derived from the security proofs.

4.1 Specifications of NORX

We provide in this section a description of the NORX family of *Authenticated Encryption with Associated Data* (AEAD) algorithms, through the description of NORX v2.0. We start by detailing in Section 4.1.1 the keyed-sponge mode and its core permutation. Then, in Section 4.1.2, we describe the security goals claimed by the designers. Finally, we outline in Section 4.1.3 the main differences between NORX v2.0 and the other members of the NORX family.

Notations. In the sequel, we use $x||y$ to denote the concatenation of two bit-strings x and y, and $|x|$ to represent the bit-length of the bit string x.

4.1.1 Description of NORX v2.0

We now describe NORX v2.0, which is the version our attack targets. It relies on w-bit words operations, with $w \in \{32, 64\}$. We note NORX-w when we consider NORX with a given w value.

Mode of Operation. NORX is based on the monkeyDuplex sponge construction [BDPVA11] and relies on a $16w$-bit permutation P that we describe later.

The monkeyDuplex sponge construction operates on an internal state S, which in the case of NORX v2.0 is divided into two distinct parts of respective bit-sizes $r = 12w$ and $c = 4w$ for a total size of $16w$ bits. We represent the

$16w$-bit internal state S of the construction as a 4×4 matrix of w-bit words as follows

$$S = \begin{bmatrix} s_0 & s_1 & s_2 & s_3 \\ s_4 & s_5 & s_6 & s_7 \\ s_8 & s_9 & s_{10} & s_{11} \\ s_{12} & s_{13} & s_{14} & s_{15} \end{bmatrix}.$$

The value r is called the *rate* of the sponge, and denotes the amount of data that can be processed by each call to permutation P. The rate part S^r of the state consists of its first 12 words. The value c is called the *capacity* and informally represents the security level expected from the construction. The capacity part S^c of the state consists of its last four words. The internal state S can then be written as $S = S^r || S^c$.

The encryption algorithm Enc takes as inputs a k-bit key K, an n-bit nonce N, a plaintext M and associated data in the form of a header A and a trailer Z. The header, plaintext and trailer are three optional strings. The encryption algorithm computes a t-bit authentication tag T, and a ciphertext C of same bit-length as the plaintext. Similarly, the decryption algorithm Dec takes as inputs (K, N, A, C, Z, T) and returns either $\perp$ or M depending on the validity of the authentication tag T.

Encryption and decryption algorithms begin by an initialization phase that sets the internal state to S_{init}: it consists in storing the $4w$-bit key $K \stackrel{\text{def}}{=} k_0||k_1||k_2||k_3$, the $2w$-bit nonce $N \stackrel{\text{def}}{=} n_0||n_1$ and some initialization constants (u_i) in the internal state, as follows:

$$S_{init} = \begin{bmatrix} n_0 & n_1 & u_2 & u_3 \\ k_0 & k_1 & k_2 & k_3 \\ u_8 & u_9 & u_{10} & u_{11} \\ u_{12} & u_{13} & u_{14} & u_{15} \end{bmatrix}.$$

After this step, some parameters of the cipher are XORed to s_{12}, s_{13}, s_{14} and s_{15}. Finally, P is applied to the full state.

The processing of the header, plaintext and trailer are similar. We assume that header, plaintext and trailer are split in blocks of bit-length $12w$. To achieve this, any non-empty field A, M or Z is padded using the so-called multi-rate padding function pad, which works as follows:

$$\mathsf{pad}(X) = 10^{f(X,w)}1,$$

where $f(X, w)$ is the smallest nonnegative integer such that $12w$ divides the total bit-length of $X||\mathsf{pad}(X)$. Header, plaintext and trailer blocks are then

processed iteratively. The whole mode of operation is depicted in Figure 24. Each block B is handled as follows.

1. A domain separation constant is first XORed to the last word of the internal state, namely s_{15}. Its value depends on the type of data being processed: `0x01` for the header, `0x02` for the plaintext, and `0x04` for the trailer.

2. The permutation P updates the internal state S; that is: $S \leftarrow \mathsf{P}(S)$.

3. The header, plaintext, or trailer block B is XORed in the rate part of the state; that is: $S^r \leftarrow S^r \oplus B$.

4. If B is a plaintext block M_i, the rate part (after XOR with B) is used as ciphertext block C_i. Note that if M_i is the last plaintext block, the part of C_i obtained from the padding is not returned as part of the ciphertext.

The last step is the tag generation. To compute the tag, first domain separation constant `0x08` is XORed to s_{15}. Then, P is applied twice to S. The t-bit tag T (where $t = 4w$) is extracted as the 4-tuple of state words (s_0, s_1, s_2, s_3).

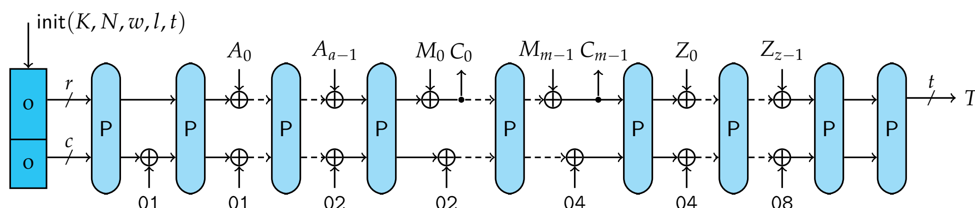

Figure 24: NORX v2.0 mode: the padded bit-strings of $12w$-bit blocks $A = A_0||\cdots||A_{a-1}$, $M = M_0||\cdots||M_{m-1}$ and $Z = Z_0||\cdots||Z_{z-1}$ are processed by the monkeyDuplex sponge construction.

The permutation P. The permutation P consists of l consecutive applications of a round function F, i.e. $\mathsf{P} = \mathsf{F}^l$. The function F in turns consists of two layers of a smaller permutation denoted G, which acts on $4w$ bits. The permutation G is first computed in parallel on the four columns of S, then on its four diagonals, as depicted in Figure 25 and Figure 26. The pseudocodes for both functions F and G are given in Algorithm 1 and Algorithm 2, respectively.

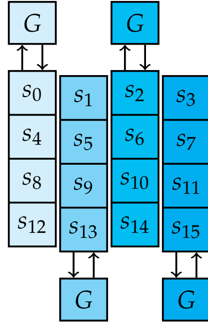

Figure 25: Function G applies on state columns.

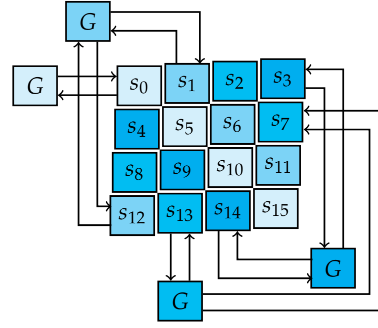

Figure 26: Function G applies on state diagonals.

Algorithm 1 – Compute $\mathsf{F}(S)$

Require: Internal state $(s_0, \ldots, s_{15})$
Ensure: Updated $(s_0, \ldots, s_{15})$

1: $(s_0, s_4, s_8, s_{12}) \leftarrow \mathsf{G}(s_0, s_4, s_8, s_{12})$
2: $(s_1, s_5, s_9, s_{13}) \leftarrow \mathsf{G}(s_1, s_5, s_9, s_{13})$
3: $(s_2, s_6, s_{10}, s_{14}) \leftarrow \mathsf{G}(s_2, s_6, s_{10}, s_{14})$
4: $(s_3, s_7, s_{11}, s_{15}) \leftarrow \mathsf{G}(s_3, s_7, s_{11}, s_{15})$
5: $(s_0, s_5, s_{10}, s_{15}) \leftarrow \mathsf{G}(s_0, s_5, s_{10}, s_{15})$
6: $(s_1, s_6, s_{11}, s_{12}) \leftarrow \mathsf{G}(s_1, s_6, s_{11}, s_{12})$
7: $(s_2, s_7, s_8, s_{13}) \leftarrow \mathsf{G}(s_2, s_7, s_8, s_{13})$
8: $(s_3, s_4, s_9, s_{14}) \leftarrow \mathsf{G}(s_3, s_4, s_9, s_{14})$
9: **return** S

Algorithm 2 – Compute $\mathsf{G}(a, b, c, d)$

Require: $4w$-bit tuple (a, b, c, d)
Ensure: Updated (a, b, c, d)

1: $a \leftarrow \mathsf{H}(a, b)$
2: $d \leftarrow (a \oplus d) \ggg r_0$
3: $c \leftarrow \mathsf{H}(c, d)$
4: $b \leftarrow (c \oplus b) \ggg r_1$
5: $a \leftarrow \mathsf{H}(a, b)$
6: $d \leftarrow (a \oplus d) \ggg r_2$
7: $c \leftarrow \mathsf{H}(c, d)$
8: $b \leftarrow (c \oplus b) \ggg r_3$
9: **return** (a, b, c, d)

Internally, the G function uses linear rotations of words and a non-linear operation, denoted by H, that mimics the modular addition modulo 2^w of bit-strings x and y:

$$\mathsf{H}(x, y) = (x \oplus y) \oplus ((x \wedge y) \ll 1).$$

The rotation constants r_0, r_1, r_2 and r_3 used in G depend on the word size (see Table 8).

Table 8: Rotation constants in the permutation G.

Instance	r_0	r_1	r_2	r_3
NORX-32	8	11	16	31
NORX-64	8	19	40	63

4.1.2 Security Claims

First of all, the designers of NORX claim no security in the event where nonces are reused: a key/nonce pair should be used only once for encryption. Similarly, there is no guarantee of security under the release of unverified plaintext [ABL+14]. Namely, if during the decryption of a ciphertext, any information on the plaintext leaks before the tag has been successfully verified, the security can no longer be ensured.

In other cases, the designers of NORX claim security levels for both confidentiality and authenticity that are equivalent to an exhaustive search of the key, which corresponds to a level of security of 128 bits for NORX-32 and 256 bits for NORX-64.

The designers also impose limitations on the amount of data that can be processed with one key. In particular, the security claims are valid as long as the usage of a key K induces fewer than 2^{2w} calls to the underlying permutation[2] P. Additionally, any forgery attack in which the adversary has x forgery attempts should succeed with probability close to $x \cdot 2^{-t}$.

4.1.3 NORX Variants

We outline here the differences between NORX v2.0 and the other members of the NORX family, either the successive entries to the CAESAR competition, or the lightweight variants. We also mention a parallel alternative to the serial mode of operation presented in Section 4.1.1.

NORX v1.0. NORX v1.0 (also named NORX v1 in some submission documents) is the initial version of NORX submitted to the CAESAR competition in March 2014. The main difference between NORX v1.0 and NORX v2.0 relates to the capacity size, which has been reduced from $6w$ bits to $4w$ bits. This change yields an increased rate with a direct impact on the efficiency of the cipher, and has been justified by security proofs, e.g. [JLM14]. The security claims are left unchanged between the two versions.

NORX v3.0. NORX v3.0 is the latest version of NORX submitted to the CAESAR competition in September 2016. Several changes have been brought to NORX between versions 2.0 and 3.0. In previous versions, a potential state-recovery attack would enable the adversary to forge valid tags by computing the encryption forwards, or even to recover the key by deducing the internal state

2 Note that the NORX specifications (v2.0 and v3.0) are unclear whether the data limitation refers either to a number of initializations or to a number of calls to the core permutation. We chose the latter as it captures both cases.

after initialization by computing backwards. In v3.0, this is no longer possible as the key K is XORed to the capacity part of the state after the initialization step, and after each of both applications of P during the generation of the authentication tag. Consequently, the tag is extracted as the capacity part S^r of the state after the last key addition.

Another modification is that NORX v3.0 uses $4w$-bit nonces instead of $2w$-bit nonces for previous versions. Again, the security claims are the same as in NORX v2.0.

NORX-8 and NORX-16. These two primitives target lightweight applications and are variants of the NORX v2.0 design, with smaller word sizes, namely $w = 8$ and $w = 16$, respectively. To achieve decent security levels, their capacities cannot be limited to $4w$ words (which would be 32 and 64 bits, respectively). Instead, their respective capacities are increased to 88 bits and 128 bits, respectively, and their capacity parts are defined as $(s_5, \dots, s_{15})$ and $(s_8, \dots, s_{15})$, respectively.

The respective key lengths for NORX-8 and NORX-16 are 80 and 96 bits, and the tag length is again the same as the key length, which define the security levels claimed for these two primitives.

In the case of NORX-8, the tag length exceeds the rate of the sponge construction. Consequently, the tag cannot be extracted at once. Instead, the 40 bits of the rate part are extracted as the first half of the tag, then an extra constant 0x08 is XORed to s_{15}, P updates the internal state, and the second half of the tag is extracted as the rate part of the state.

The amount of data processed with a given key is limited to respectively 2^{24} and 2^{32} messages.

Parallel Mode of Operation. The NORX variants submitted to the CAESAR competition offer a parallel mode of operation, which enables to process in parallel $p > 1$ blocks of plaintext simultaneously. Basically, the state of the mode of operation is diversified into p branches, the plaintext blocks are dispatched over the branches for processing, the branches are combined, and the trailer and tag are handled as in the serial mode.

4.2 Cryptanalysis of NORX v2.0

We give in this section the details of a ciphertext-only forgery attack on NORX v2.0 that exists due to a combination of aggressive choices made by the designers. The attack indeed relies on strong non-random properties

of the underlying permutation $\mathsf{P} = \mathsf{F}^l$ used in a keyed-sponge mode, as well as a relatively small sponge capacity. Additionally, we show that the forgery attack yields a plaintext-recovery attack and a key-recovery attack with the same complexities. We begin in Section 4.2.1 by giving non-random properties of F that extend to P, describe a simplified version of the forgery attack in Section 4.2.2 and then the full attack in Section 4.2.3. We discuss requirements for the adversarial model in Section 4.2.4 and give extensions of the attack in Section 4.2.5.

4.2.1 Non-Random Properties of F

In the specification document of NORX [AJN15b] and in another analysis paper [AJN15a], the designers acknowledge the use of a permutation that presents non-random properties. They argue that distinguishers on the permutation do not affect the overall monkeyDuplex construction since domain separation constants are used at the mode level. Security proofs have been written for the NORX mode, e.g. [JLM14], which assumes an ideal permutation and sets aside its structural weaknesses.

In the sequel, we recall a strong distinguisher on F and later show how to leverage it to attack the full primitive. We note that our attack does not invalidate the security proofs of the mode, which rely on the assumption that the permutation is ideal and does not present any distinguisher like the one we describe.

Previous Work. First, in [AJN15b], the designers use the constraint solver STP to confidently assume that the permutations used in all NORX variants present only a single fixed-point, namely the all-zero state: $\{x, \mathsf{F}(x) = x\} = \{0\}$. Later in [AJN15a], the same authors introduce a class of 2^{4w} *weak states* of the form

$$\begin{bmatrix} a & a & a & a \\ b & b & b & b \\ c & c & c & c \\ d & d & d & d \end{bmatrix}, \quad (a, b, c, d) \in \mathrm{GF}(2^w),$$

where all the four columns of the state are equal. Due to the column/diagonal applications of G in the permutation F (see Section 4.1), it is easy to see that the set of these weak states is stable by F: starting from a weak state, applying F any number of times leads to a weak state. In particular, the set of weak states is stable by $\mathsf{P} = \mathsf{F}^l$.

A Stronger Distinguisher. We note here that there exists a larger class of 2^{8w} states behaving in a similar way, where the two left columns equal the two right ones; namely, states of the form:

$$\begin{bmatrix} a & e & a & e \\ b & f & b & f \\ c & g & c & g \\ d & h & d & h \end{bmatrix}, \quad (a,b,c,d,e,f,g,h) \in \mathrm{GF}(2^w).$$

Again, this larger class is stable by F and P.

Additionally, we note that one can slightly generalize the notion by considering "rotated" variants of one state. More formally, we denote by $S^{\lll i}$ the state S where the columns are left-rotated by i positions. Given $x_i \in \mathrm{GF}(2^w)$, $0 \leq i < 16$, consider the state

$$S = \begin{bmatrix} x_0 & x_1 & x_2 & x_3 \\ x_4 & x_5 & x_6 & x_7 \\ x_8 & x_9 & x_{10} & x_{11} \\ x_{12} & x_{13} & x_{14} & x_{15} \end{bmatrix},$$

and the three states obtained by rotating the columns of S by one, two and three positions:

$$S^{\lll 1} = \begin{bmatrix} x_1 & x_2 & x_3 & x_0 \\ x_5 & x_6 & x_7 & x_4 \\ x_9 & x_{10} & x_{11} & x_8 \\ x_{13} & x_{14} & x_{15} & x_{12} \end{bmatrix}, S^{\lll 2} = \begin{bmatrix} x_2 & x_3 & x_0 & x_1 \\ x_6 & x_7 & x_4 & x_5 \\ x_{10} & x_{11} & x_8 & x_9 \\ x_{14} & x_{15} & x_{12} & x_{13} \end{bmatrix},$$

$$S^{\lll 3} = \begin{bmatrix} x_3 & x_0 & x_1 & x_2 \\ x_7 & x_4 & x_5 & x_6 \\ x_{11} & x_8 & x_9 & x_{10} \\ x_{15} & x_{12} & x_{13} & x_{14} \end{bmatrix}.$$

Our main observation is that F and the column rotations commute, that is:

$$\forall i \in \{1,2,3\}, \quad \mathsf{F}(S^{\lll i}) = \mathsf{F}(S)^{\lll i}.$$

We define by *symmetric* a state S that is invariant by rotation by two positions: $S = S^{\lll 2}$. Similarly, we say that the capacity part of an internal state is symmetric if this internal state restricted to that part is invariant by rotation by two positions.

In the following section, we show how the small proportion of the internal state allocated to the capacity in both NORX-32 v2.0 and NORX-64 v2.0 allows to use this structural distinguisher to mount a ciphertext-only forgery attack on these two primitives.

4.2.2 Ciphertext-Only Forgery of NORX v2.0 Without Padding

Recall that the security of NORX-w relies on a capacity of $4w$ bits, and its key and tag sizes are of the same size $4w$ bits.

We now consider a modified version of NORX, in which the plaintext (and therefore ciphertext) lengths are always a multiple of the block size $12w$. Therefore, no padding needs to be added to the plaintext before encryption. This modification enables us to describe a simplified version of our attack, which can be adapted to the full NORX v2.0 as shown in Section Section 4.2.3.

The following describes a ciphertext-only forgery attack against NORX v2.0 without padding, that requires q valid ciphertext/tag pairs (C, T), performs q forgery attempts, and has success probability

$$1 - \left(1 - \frac{1}{2^{2w}}\right)^q.$$

In particular, the forgery attacks succeeds with probability $1 - 1/e \approx 63\%$ for $q = 2^{2w}$, and with probability about $q \cdot 2^{-2w}$ for smaller values of q. We require that there is no trailer, that the plaintexts and ciphertexts lengths are multiples of the block size, and that the cipher does not apply any padding. Without loss of generality, we assume there is no header and that the plaintext and ciphertext length is exactly one block. If it is not the case, the attack applies directly be applying ciphertext modifications only on the last block.

Assume that an attacker has q known tuples (N^i, C^i, T^i) in its possession, resulting from the NORX-w encryption of unknown messages M^i, under known pairwise distinct nonces N_i and unknown key K:

$$(N^i, C^i, T^i) = \mathsf{Enc}(K, N^i, M^i).$$

Given such a tuple, (N, C, T), the attacker attempts to produce a forgery by considering the message $(N, C^{\lll 2}, T^{\lll 2})$. The ciphertext and tag parts of the message are rotated variants of the initial ciphertext and tag. In the event that the capacity part of the state is symmetric before the last two calls to P for the generation of the tag (see Figure 27), the states S_* and S'_* at the same point of the computation are rotated versions of each other, and due to the

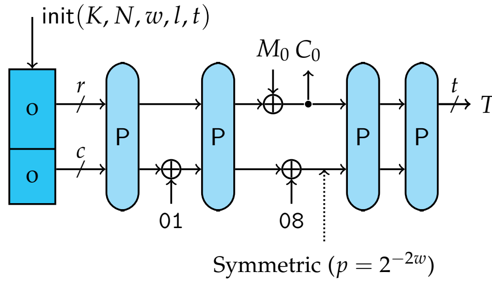

Figure 27: Forgery first step: assume the capacity is symmetric (probability 2^{-2w}).

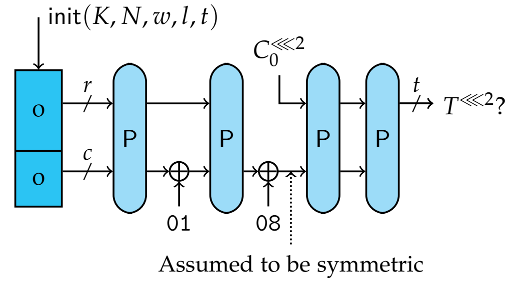

Figure 28: Forgery second step: attempt forgery with rotated ciphertext and tag.

fact that P and the rotation commute, this is also satisfied by the tags. More formally, we have the internal state S'_* as

$$\begin{aligned} S'_* &= C_0^{\lll 2} \mid\mid S_*^c, \\ &= C_0^{\lll 2} \mid\mid (S_*^c)^{\lll 2}, \\ &= (C_0 \mid\mid S_*^c)^{\lll 2}. \end{aligned}$$

and evaluate the two last applications of P, which gives

$$\begin{aligned} \mathsf{P}^2(S'_*) &= \mathsf{P}^2\left((C_0 \mid\mid S_*^c)^{\lll 2}\right), \\ &= \left(\mathsf{P}^2\left(C_0 \mid\mid S_*^c\right)\right)^{\lll 2}, \end{aligned}$$

and then yield the equality on the authentication tags

$$T'_* = T_*^{\lll 2}.$$

The probability for a tuple to yield an internal state such that its capacity is symmetric before the last two calls to P for the generation of the tag (see Figure 27) is 2^{-2w}.

All in all, as an attacker has a probability of 2^{-2w} to forge a valid message due to the symmetries in P, he only needs about 2^{2w} known ciphertext/tag pairs to launch the attack and break the authenticity of NORX-w.

4.2.3 Forgery Attack Against NORX v2.0

We now adapt the attack to take into account the padding systematically applied by NORX to any non-empty plaintext.

The difficulty introduced by the padding is that the attacker has no longer access to the whole rate part of the state S_*: the part corresponding to the padding is not included in the ciphertext. In order to minimize this unknown

component, we consider only messages of size $12w - 2$ bits, which lead to the minimal padding length of two bits.

In order to forge a message using the commuting rotation property of P, the attacker has to produce a ciphertext C' such that the state S'_* is the rotated version of state S_*. In addition to the constraint that the capacity part of the state remains unchanged, new constraints are introduced by the padding, stemming from the matching between

$$\left(S'_*\right)^r = \begin{bmatrix} c'_0 & c'_1 & c'_2 & c'_3 \\ c'_4 & c'_5 & c'_6 & c'_7 \\ c'_8 & c'_9 & c'_{10} & c'_{11} \,||\, v \end{bmatrix} \quad \text{and} \quad (S^r_*)^{\lll 2} = \begin{bmatrix} c_2 & c_3 & c_0 & c_1 \\ c_6 & c_7 & c_4 & c_5 \\ c_{10} & c_{11} \,||\, v & c_8 & c_9 \end{bmatrix},$$

with v the unknown part of S^r_*. Note that the 2-bit padding v only depends on C and C' through their length, and is thus repeated in both S_* and S'_*. Denoting by $\underline{x}$ the last two bits of x, the padding constraints are satisfied if we set the bits of C' to the corresponding known bits of C, and additionally

$$\underline{c'_9} = v \quad \text{and} \quad \underline{c_9} = v.$$

Setting $\underline{c_9} = \underline{c'_9}$, the constraints boil down to $\underline{c_9} = v$ which holds with probability 2^{-2}.

Overall, taking the padding into account results in a decrease of the advantage of the attacker, that can be limited to a factor 2^{-2} for the most favorable message length. This attack can trivially be extended to any padding length $p \leq 2w$ with complexity 2^{2w+p} instead of 2^{2w+2}.

4.2.4 Adversarial Model Discussion

Our attack is efficient on the padded version of NORX only if the length of the padding appended to the plaintext leading to the ciphertext the adversary tries to modify is minimal. Formally, if we keep the minimal padding length of two bits, this can lead to the following two scenarios:

- In a chosen-plaintext setting, the adversary can select plaintexts of length equal to $12w - 2 \pmod{12w}$. The success probability of each forgery attempt is then 2^{-2w-2}.
- In a ciphertext-only setting, the attack still works as the adversary does not need to know the value of the corresponding plaintext. However, it requires that ciphertexts whose last block has a specific length are available. Under the hypothesis that the length of the message follows a uniform distribution modulo $12w$, the adversary can try to modify only those ciphertexts, which introduces a factor $12w$ in the data complexity.

We note that this constraint relies on the general description of NORX at the bit level, whereas the functional requirements of the CAESAR competition acts on byte strings. Consequently, to launch the attack in that case, ciphertexts of L bytes are required, with $L = -1 \pmod{12w/8}$ and the advantage of the attacker becomes $q \cdot 2^{-2w-8}$. If this requirement on L does not hold, the data complexity would increase by a factor $12w/8$, assuming again that the ciphertext byte-lengths modulo $12w/8$ are uniformly distributed.

4.2.5 Key-Recovery Attack Against NORX v2.0

Recovering the Key. We now discuss whether it is possible to recover the encryption key from a successful forgery attempt. Once the adversary achieves such a forgery, he knows that with overwhelming probability, the capacity part of the state at the end of the encryption step is symmetric. Therefore, only 2^{2w} values are possible for the capacity part of the state at that point. As the adversary knows the value of the rate part, he can recover the full state by an exhaustive search over these 2^{2w} values. Trying all 2^{2w} possible symmetric values at the input of F^8 allows to filter (on average) one internal state.

Let us suppose that the adversary additionally knows at that point the value of the plaintext returned by the decryption algorithm on his successful forgery. He can then compute backwards up to the initialization of the state and filter the correct guess on the $4w$-bit constants, which subsequently reveals the $4w$-bit secret key.

We have successfully verified the forgery and key-recovery attacks on a toy version of NORX v2.0, by taking the word size $w = 8$ and adopting the rotation constants of NORX-8. The pseudo-code of the attack can be found in Section 4.3.

Adversarial Models. In a ciphertext-only setting, the adversary does not get the value of the plaintext after the decryption and cannot perform the last step of the key-recovery attack. It is however possible in chosen-plaintext or chosen-ciphertext settings. If the adversary can query a decryption oracle, he gets the value of the plaintext he needs to compute backwards and recovers the key.

If the adversary can query an encryption oracle, he can encrypt arbitrary one-block plaintexts and try to forge valid ciphertexts by modifying the answers of the oracle. He can then perform the key-recovery attack on the initial plaintext-ciphertext pair.

4.3 Pseudo-code for the Ciphertext-only Forgery and Key-Recovery Attack

The pseudo-code for the forgery and key-recovery attacks are given in the following Algorithm 3. We have implemented the attack on a toy example of NORX v2.0 derived from the source code provided by the designers as part of the CAESAR competition. We in particular emphasize that due to the CAESAR requirements, all the inputs are byte strings, hence the padding cannot be restricted to less than one byte.

Algorithm 3 – Forgery and Key-Recovery Attack on NORX v2.0

Require: 2^{2w} ciphertext/tag pairs (C_i, T_i), $2w$-bit nonce $N = n_0||n_1$
Ensure: Secret key K

1: **for** each ciphertext $C_i = (c_0, \ldots, c_{10})$ and tag $T_i = (t_0, \ldots, t_3)$ **do**
2:   $C_i^{'} \leftarrow (c_2, c_3, c_0, c_1, c_6, c_7, c_4, c_5, c_{10}, c_9, c_8)$
3:   $T_i^{'} \leftarrow (t_2, t_3, t_0, t_1)$
4:   $M' \leftarrow \mathsf{Dec}(N, C^{'}, T')$
5:   **if** $M' \neq \perp$ **then**
6:     **for all** words a, b **do**
7:       $S \leftarrow (c_0, c_1, c_2, c_3, c_4, c_5, c_6, c_7, c_8, c_9, c_{10}, c_9, a, b, a, b)$
8:       $S \leftarrow S \oplus M'||0^c$
9:       $s_{15} \leftarrow s_{15} \oplus 08$
10:       $S \leftarrow \mathsf{P}^{-1}(S)$
11:       $s_{15} \leftarrow s_{15} \oplus 02$
12:       $S \leftarrow \mathsf{P}^{-1}(S)$
13:       **if** $(s_0, s_1, s_2, s_3) = (n_0, n_1, u_2, u_3)$ **then**
14:         **return** $K = (s_4, s_5, s_6, s_7)$
15:       **end if**
16:     **end for**
17:   **end if**
18: **end for**

4.4 Application to Other Variants of NORX

In this section, we study the application of our attack to other versions or variants of NORX. Namely, we show the following properties that we explain below.

1. NORX-8 is not harmed at all by our attack.

2. The parameters chosen in NORX v1.0 and NORX-16 makes our attack just as efficient as generic attacks. A consequence is that increasing the key and tag sizes for these versions would not increase their security. In

particular, a surprising behavior is that if one increases the key and tag lengths of NORX-16 to 128 bits, then the security drops to 2^{66}.

3. NORX v3.0 has a small class of keys on which our attack is as efficient as a generic key-recovery attack.

NORX **v1.0.** We recall that the main difference between NORX v1.0 and NORX v2.0 is that in NORX v1.0, the rate part of the state consists of words $(s_0, \ldots, s_9)$ and the capacity part of the state consists of words $(s_{10}, \ldots, s_{15})$.

Let us consider an adversary who tries to launch our attack against NORX v1.0. Let us suppose that the bit-length of the last block is exactly $8w$. He can can only apply the rotation on the first two rows of the state after the output of the last ciphertext block, which are filled with the last eight ciphertext words. On the last row, the same symmetry condition as in NORX v2.0 has to hold, which occurs with probability 2^{-2w}.

The adversary then has to ensure that the third row of the state during its forgery attempts can be derived by a column-wise rotation of the third row of the state during the generation of the ciphertext he tries to modify. The third row of the state during the encryption equals $(s_8, s_9, s_{10}, s_{11})$, where s_8 and s_9 have just been updated by XORing the padding.

Then, during the verification of the forgery attempt, the third row contains the same value $(s_8, s_9, s_{10}, s_{11})$, The symmetry relations he tries to obtain are as follows:

$$s_8 = s_{10}, \quad s_9 = s_{11},$$

which hold with probability 2^{-2w}.

The overall success probability of the adversary is thus 2^{-4w}, which is exactly the success probability of a generic forgery attempt as the tag length is $t = 4w$.

NORX **v3.0.** During the tag generation phase, the only difference between NORX v2.0 and NORX v3.0 consists in XORing the key K after each application of P, as depicted on Figure 29.

As a consequence, the rotation property between the states during the real encryption and the forgery attempt can only be preserved before the last application of P if the key $K = k_0||k_1||k_2||k_3$ is itself symmetric; that is, if $k_0 = k_2$ and $k_1 = k_3$. In that case, our attack still works.

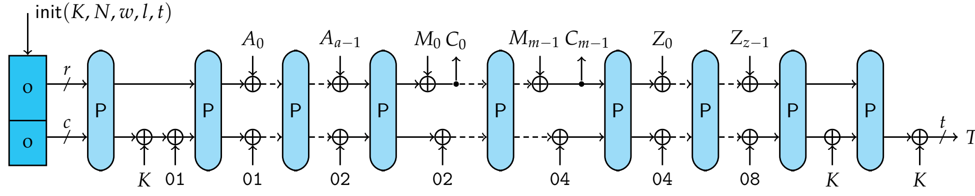

Figure 29: NORX v3.0 serial mode.

These relations can be seen as defining a class of 2^{2w} weak keys on NORX v3.0. However, the resulting attack enables an adversary to generate forgeries with data complexity 2^{2w}, which is equivalent to the size of the weak key set. Furthermore, the forgery attack cannot be trivially turned into a key-recovery attack against NORX v3.0. Our attack therefore has a very limited impact on the security of NORX v3.0.

NORX-8. Recall that NORX-8 is very similar to NORX v2.0, but that the authentication tag cannot be fully extracted at once from the rate part of the state. Instead, after the extraction of the first 40 tag bits, a diversification constant is injected in the state, P is computed and the last 40 tag bits are extracted from the rate part of the state.

Even if the adversary achieves the rotation property after the last ciphertext block, this property is broken after the addition of the diversification constant, and no predictable property holds for the second half of the tag. In that case, only the first 32 bits of the tag (which are extracted from the first row of the state) can be predicted, leading to a forgery with probability $2^{32-80} = 2^{-48}$.

Furthermore, the rotation property itself only holds with probability 2^{-48}, due to symmetry conditions on the last three rows of the state, which contain the capacity part. The overall success probability of our attack is therefore 2^{-96}, making it less efficient than a generic attack.

NORX-16. In NORX-16, the capacity part of the state covers the last two rows, i.e. $(s_8, \ldots s_{15})$. Therefore, the rotation property holds with probability $2^{-4w} = 2^{-64}$. NORX-16 uses 96-bit keys and produces 96-bit tags, which are extracted as $(s_0, \ldots, s_6)$ after the last application of P. If the rotation property holds, the adversary knows the target values of $(s_0, \ldots, s_3)$ (by rotation of the valid tag), but he still needs to guess s_4 and s_5. Taking account of the 2-bit loss due to the padding, the overall success probability of each forgery attempt is $2^{-64-2\times 16-2} = 2^{-98}$, which is just below the generic bound for a forgery attempt.

This shows that increasing the key and tag sizes of NORX-16 would not increase its security, as our attack would still be valid. More surprisingly, using 128-bit tags would enable the adversary to always forge successfully once the rotation property is verified, leading to an attack with success probability 2^{-66} for each forgery attempt.

4.5 Discussion About NORX Security Claims

NORX v2.0 Security Claims In [AJN15b, Section 3], the NORX designers claim that no forgery attack with q attempts should succeed with probability significantly greater than $q \cdot 2^{-4w}$. Our attack succeeds with probability about $q \cdot 2^{-2w-2}$, which violates this claim.

The designers also claim that no key-recovery attack should cost fewer than 2^{4w} operations. Our attack costs 2^{2w+2} operations on average. One could argue that the limitation of the amount of data treated with a given key limits the success probability of our attack. Nevertheless, contrary to attacks based on the birthday paradox, the marginal success probability of a single forgery attempt using our attack does not drop once the key is changed. Consequently, our attack enables the adversary to find the value of one of the keys used with time and data complexity of about 2^{66} operations (for $w = 32$), regardless of the change frequency.

Forgeries Against NORX v3.0 for Long Messages. For both NORX v2.0 and NORX v3.0, the security claim saying that any forgery attack with q attempts should have a success probability of about $q \cdot 2^{-4w}$ does not totally hold.

For any long ciphertext C that contains, say, $2^m + 1$ blocks of $12w$ bits, one can modify only the first block of the ciphertext, keep the same tag value and obtain a forgery with probability about 2^{m-4w}. Indeed, before each application of P during the decryption phase, the internal state during the forgery attempt collides with the internal state during the decryption of the initial message with probability $2^{-c} = 2^{-4w}$. Once a collision occurs, it holds for all the subsequent steps of the decryption process, as the two decrypted ciphertexts have common suffixes. The overall collision probability is therefore approximately $2^m \times 2^{-4w}$, and such a collision leads to equal tag values, making the forgery attempt successful. We note that this technique shares some ideas with the long-message internal collision attack on iterated MACs discussed in [PvO95, Section 3].

For NORX v2.0 and NORX v3.0, this property still holds when the nonce is modified in the forgery attempt. For NORX v2.0, as no key is involved after the initialization phase, one consequence of this property is that a given

ciphertext of 2^m blocks has the same tag value under two different keys and nonces with probability 2^{m-4w}.

The impact (at least on NORX v3.0) of this remark has to be mitigated by the fact that similar properties can apply to other AEAD schemes such as AES-GCM [MV04a]. It is also covered by the security proof, which leads to bounds involving the total length of encryption and decryption queries, and not only the number of forgery attempts.

NORX **Security Proof.** In [AJN15b], the designers partly derive their security analysis from security proofs of the keyed-sponge mode of operation which can be found in [JLM14]. Namely, the distinguishing advantage of any chosen-plaintext adversary against NORX is upper bounded by:

$$\Pr[\text{Privacy}] \leq \frac{3(q_p + \sigma_\mathcal{E})^2}{2^{b+1}} + \left(\frac{8eq_p\sigma_\mathcal{E}}{2^b}\right)^{1/2} + \frac{rq_p}{2^c} + \frac{q_p + \sigma_\mathcal{E}}{2^k}.$$

Similarly, the upper bound for the success probability of any forgery attempt is given by:

$$\Pr[\text{Forgery}] \leq \frac{(q_p + \sigma_\mathcal{E} + \sigma_\mathcal{D})^2}{2^b} + \left(\frac{8eq_p\sigma_\mathcal{E}}{2^b}\right)^{1/2} + \frac{rq_p}{2^c} + \frac{q_p + \sigma_\mathcal{E} + \sigma_\mathcal{D}}{2^k} + \frac{(q_p + \sigma_\mathcal{E} + \sigma_\mathcal{D})\sigma_\mathcal{D}}{2^c} + \frac{q_\mathcal{D}}{2^t}.$$

In these formulae, b is the state size, c is the capacity, r is the rate, q_p is the number of calls to the internal permutation, $q_\mathcal{D}$ is the number of forgery attempts, and $\sigma_\mathcal{E}$ and $\sigma_\mathcal{D}$ are the number of total computations of the internal permutations during encryption and decryption queries, respectively.

Our attack succeeds with probability $q_\mathcal{D}/2^{2w+2}$, which is significantly larger than this bound for a small number of queries (as we would have $\sigma_\mathcal{E} = \sigma_\mathcal{D} = 4q_\mathcal{D}$ as we only need to make one-block encryption and decryption queries).

We emphasize that our attack does not contradict the proof of the NORX mode of operation, as it relies on the use of an ideal internal permutation instead of P. However, it reveals that the proof does not apply to the instantiation of the mode chosen by the designers, as the selected NORX permutation presents (at least) one strong structural distinguisher.

Security Level of NORX-8**.** In [AJN15c], the authors do not provide an explicit link between the above security bounds and the claimed security level of NORX-8 and NORX-16. In particular, they only state that no more than 2^{24} (resp. 2^{32}) initialization phases should be performed with the same key, but they do not give any limit to the total length of messages encrypted with a

key. We can notice that if one encrypts constant 0 blocks, NORX can be viewed as a stream cipher, and therefore the Babbage-Golić [Bab95, Gol97] Time-Data tradeoff applies. In particular, NORX-8 has an internal state of only 128 bits. Therefore, if one can encrypt $2^m \gg 2^{48}$ message blocks under the same key with NORX-8, the security level drops below 80 bits since a state-recovery attack of time and memory complexity at most 2^{128-m} can be mounted, that can in turn easily be converted into a key-recovery attack using backward computations.

Interpretation of the NORX Proof. Finally, we would like to raise the following problem. In the bound derived from the proof of the NORX mode of operation, the term $q_p \sigma_{\mathcal{D}}/2^c$ appears. In the case of NORX-32 for both v2.0 and v3.0, the capacity equals $c = 128$. Note that σ_D can roughly be considered as the total length of decryption queries, and is only limited to 2^{64} in the specifications. In real-life applications, σ_D could possibly reach between 2^{40} and 2^{48}.

In that case, q_p has to be smaller than 2^{80} to 2^{88} if one wants to conclude any meaningful information from the bound. Note however that q_p represents the number of calls to the internal permutation made by the adversary. In our view, as P is an unkeyed permutation, these calls do not involve any secret and can therefore be interpreted as offline computations. The security of NORX as derived from the security proof then drops between 80 and 88 bits.

However, this remark is very unlikely to lead to an attack on NORX v3.0 that would match this bound, for two reasons. First, when looking at the details of the proof, this term captures the event that a direct call to P by the adversary collides with an application of P during the verification of a decryption query. As the adversary does not get much information from decryption queries, it is unlikely that he can detect such an event. Second, the mode of operation of NORX v3.0 (with key additions after initialization and during the tag computation) is close to the sandwich sponge construction by Naito [Nai16]. In the same paper, this construction is proved to be indistinguishable from a PRF up to a bound without such a term proportional to online-times-offline complexity; whereas a similar term still appears in the best known bounds for the usual sponge construction.

5 CRYPTANALYSIS OF KRAVATTE

In this last chapter, we present several attacks on the Kravatte algorithm, an instantiation of the permutation-based PRF construction `Farfalle`. Theses attacks were developed in a joint work with Thomas Fuhr, Henri Gilbert, Jian Guo, Jérémy Jean, Jean-René Reinhard and Ling Song. Our results were published in the paper *Key-Recovery Attacks on Full Kravatte* [CFG+18a] and presented at FSE 2018.

`Farfalle` is an efficiently parallelizable permutation-based construction of a variable input and output length *pseudorandom function* (PRF) recently proposed by Bertoni et al. [BDH+16]. It represents an extremely versatile building block for the design of symmetric mechanisms. It can indeed be used either directly as a message authentication code (MAC), as the keystream generation part of a stream cipher, as a key derivation function (KDF), or otherwise in a mode of operation allowing to convert it into a more complex mechanism, for instance an authenticated encryption scheme or a block cipher of variable block length.

`Farfalle` takes as input a key and a (sequence of) data string(s) of arbitrary length(s) and produces an output of arbitrary length. Its construction involves two basic ingredients: a set of *permutations* of a b-bit state, and a family of so-called *rolling functions* used to derive distinct b-bit *mask values* from a b-bit *mask key* or more generally b-bit variants of a b-bit state.

The `Farfalle` construction consists of a *compression layer* followed by an *expansion layer*. The compression layer produces a single b-bit *accumulator value* from a tuple of b-bit blocks representing the input data. The expansion layer first non-linearly transforms the accumulator value into a b-bit *rolling state* and then non-linearly transforms a tuple of variants of this rolling state, produced by iterating the rolling function into a tuple of (truncated) b-bit output blocks. Both the compression and the expansion layer involve b-bit mask values derived from the key by the *key derivation* part of the construction.

An efficient instantiation of the `Farfalle` construction named Kravatte is also specified in [BDH+16]. The underlying components are a set of `Keccak`-p

permutations of a $b = 1600$-bit state, and a family of simple $\mathbb{F}_2$-linear rolling functions. The variants of Kravatte are addressed according to the number of rounds in the internal permutations: n_b rounds for the key derivation, n_c rounds for the compression layer, n_d rounds for the non-linear transformation applied of the accumulator, and n_e rounds for the expansion layer. The specifications of Kravatte published on the IACR ePrint in July 2017 use $(n_b, n_c, n_d, n_e) = (6, 6, 4, 4)$ and an announcement at ECC 2017 of a strengthened variant [BDH+17b] uses $(n_b, n_c, n_d, n_e) = (6, 6, 6, 6)$.

Our Contributions

In this chapter, we present three families of attacks against full Kravatte (IACR ePrint and ECC versions), whose time and data complexities are far below the security claimed by the designers. Furthermore, one of them can even be successfully applied to strengthened variants of Kravatte.

The first two attack strategies focus on the expansion layer *after* the application of its initial non-linear transformation. They exploit that all output blocks are generated from the same initial rolling state, and the small number of Keccak-p rounds between the rolling state diversification and the block outputs. They require a long Kravatte output generated from a single and possibly unknown message. The compression layer and the derivation of the rolling state from the accumulator value do not contribute any security against these attacks. The third strategy focuses on a property of the compression layer.

Meet-in-the-Middle Algebraic Attack. The first attack can be seen as a meet-in-the-middle (MITM) algebraic attack, and bears some resemblance to the meet-in-the-middle approach applied to interpolation attacks [JK97]. The rolling state and the output masking key are the unknowns of an algebraic system built by forming expressions of the same intermediate state, either by forward computation from the rolling state, or by backward computation from the output. The expansion mechanism makes it possible to collect enough equations to solve the system by linearization.

Linear Recurrence Distinguisher. The second attack strategy leverages the linear branch diversification mechanism of the expansion layer: the rolling state can be assimilated to a short LFSR state, due to the restriction of the rolling function to only 320 bits of the 1600-bit state. As a consequence, the linear complexity of the sequence of blocks obtained by application to consecutive rolling state values of a small number of the low-degree Keccak round function is also limited, i.e., this sequence satisfies a linear recurrence of order far smaller than what is expected from the size of the state. Furthermore, the recurrence polynomial of this sequence of blocks can be derived

at a moderate cost. This observation provides the linear recurrence distinguisher used in our attack.

Higher Order Differential Distinguisher. The last attack strategy is essentially a higher order differential distinguisher. First, the compression layer of `Farfalle` produces an accumulator state equal to the *exclusive or* of nonlinear permutations of the b-bit blocks representing the input data. This property allows the compression layer to satisfy the design requirements of being efficiently parallelizable and *incremental*.[1] However, it also allows an adversary to construct simple structures of 2^n n-block input values whose images by the compression layer form an affine subspace of dimension n of $\{0,1\}^b$.

Moreover, Kravatte relies on the `Keccak`-p permutation, whose round function has an algebraic degree only two and the rolling function is $\mathbb{F}_2$-linear. Therefore, if we denote by r the number of `Keccak`-p rounds of the partial computation —on input the accumulator state and up to ϵ final `Keccak`-p rounds— of any of the output blocks of the expansion layer, the algebraic degree of this partial expansion is upper bounded by 2^r. This implies that if $n > 2^r$, the sum of the outputs of this partial expansion over the accumulator values associated with one of the structures mentioned above is equal to zero. This observation provides the higher order differential distinguisher used in our attack.

Last Round Attacks. The attacks all rely on the capacity to “invert” up to two of the last rounds of the expansion layer despite a final masking of the output values by a key block. This can be done algebraically, by expressing the intermediate values as a function of the Kravatte output block and of the unknown key block, setting up a system of multivariate polynomial equations, and solving this system by linearization. Surprisingly, this is more efficient than expected from the algebraic degree of the inverse of the last rounds due to the limited diffusion in a small number of iterations of the inverse round function of `Keccak`.

This notably offers the possibility to leverage distinguishers on partial versions of Kravatte, and then mount key-recovery attacks on the full primitive. We give in Table 9 a list of the key-recovery attacks that are described in the rest of the chapter.

Optimizations. Various technical improvements can be applied to the attack strategies in order to optimize the time, memory, or data complexity. We already note that some of these techniques improve some of the complexities

---

1 *Incremental* means that if two input data share the same prefix, their compression layer computations can be partly shared.

Table 9: Key-recovery attacks against KRAVATTE instantiations for several (n_d, n_e) values. All attacks are independent of n_b and n_c, and $\star$ means that n_d can take any value. The reference points to the section describing the attack type. The complexity figures are obtained after the selection of optimizations described in Section 5.4.

$(\mathbf{n_d}, \mathbf{n_e})$	**Type**	**Data**	**Memory**	**Time**	**Reference**
		blocks	bits	basic op.	
$(4, 4)$	Higher Order	$2^{74.7}$	$2^{62.3}$	$2^{112.2}$	Section 5.3
$(\star, 4)$	MITM	$2^{27.8}$	$2^{76.9}$	$2^{115.3}$	Section 5.2.1
$(\star, 4)$	Linear Recurrence	$2^{51.2}$	$2^{51.2}$	$2^{65.1}$	Section 5.2.2
$(\star, 4)$	Linear Recurrence	$2^{29.9}$	$2^{62.3}$	$2^{87.0}$	Section 5.2.2
$(\star, 6)$	Linear Recurrence	$2^{88.4}$	$2^{88.4}$	$2^{134.6}$	Section 5.2.2

but downgrade some others, which makes the selection of improvements a trade-off process. We discuss these optimizations in a dedicated section (Section 5.4) after the presentation of the attack strategies.

Organization. We give a description of KRAVATTE in Section 5.1, an instantiation of the `Farfalle` construction. In Section 5.2, we describe a MITM algebraic attack and an attack based on the linear recurrence distinguisher of KRAVATTE partial expansion layer. Both attack strategies focus on the expansion layer of KRAVATTE. In Section 5.3, we describe a higher order differential attack on KRAVATTE. In Section 5.4, we describe technical optimizations that can be applied to the attack strategies in order to improve their complexities, and provide a selection of attacks optimized either for time, memory or data complexity. Finally, we discuss in Section 5.5 the insights gained from these attacks.

5.1 Specifications of `Farfalle` and KRAVATTE

In this section, we give a description of permutation-based mode `Farfalle` and its original instantiation KRAVATTE, which is based on the permutation used in `Keccak` [BDPA11a, NIS14]. The two primitives `Farfalle` and KRAVATTE have both been designed by Bertoni et al., originally published on the IACR ePrint in [BDH+16], and strengthened versions have been accepted at ToSC and presneted at the FSE 2018 conference [BDH+17a].

5.1.1 The `Farfalle` Construction for Permutation-Based PRFs

KRAVATTE is a permutation-based variable input and output length pseudorandom function. It takes as input a key and a sequence of bit strings, and returns an arbitrary-length output. It relies on the `Farfalle` construction,

which allows to build a PRF from parallel applications of fixed permutations. In this chapter and without loss of generality, we focus on input sequences that contain only one bit string, while the general construction allows for vectors of bit strings.

Farfalle makes use of four permutations (possibly identical or related), denoted p_b, p_c, p_d and p_e of a b-bit block. Its instantiation requires the definition of three so-called *rolling functions*, denoted $roll_c$, $roll_e$ and $roll_f$. These functions should ensure that for an unknown value x, an adversary cannot predict the value of any number of iterations of the rolling functions $roll^i(x)$, nor the value of $roll^i(x) \oplus roll^j(x)$ for $i \neq j$.

The Farfalle construction takes as input a key K and a message M. For an ℓ_i-block input message and a ℓ_o-block output, Farfalle consists of the three following steps:

MASK DERIVATION: The key K is padded into a b-bit string $K\|10^*$, on which the permutation p_b is applied and yields $\mathbf{k}^{in} = p_b(K\|10^*)$. Denoting $\mathbf{k}^{out} = roll_c^{\ell_i+1}(\mathbf{k}^{in})$, $\ell_i + \ell_j$ masks are then computed as $\mathbf{k}_i^{in} = roll_c^i(\mathbf{k}^{in})$ for $i = 0, \ldots, \ell_i - 1$, and $\mathbf{k}_j^{out} = roll_f^j(\mathbf{k}^{out})$ for $j = 0, \ldots, \ell_o - 1$.

COMPRESSION LAYER: The message M is padded into a ℓ_i sequence of b-bit blocks m_i, by appending a 1-bit and a sequence of 0-bits. Then, one compresses these data into a single b-bit block accumulator x, by XOR-ing a key mask to each block, applying the permutation p_c to the results, and XOR-ing all the results together: $\texttt{Acc}(M) = \sum_i p_c\left(m_i \oplus \mathbf{k}_i^{in}\right)$.

EXPANSION LAYER: In a first step, the permutation p_d is applied on the accumulator to get $\mathbf{y} = p_d(\texttt{Acc}(M))$. Then, in a second step, ℓ_o output blocks $\mathbf{z}^j$ are computed from this value by applying consecutively a rolling function, the permutation p_e, and an XOR with the key mask $\mathbf{k}_j^{out}$: namely, $\mathbf{z}^j = p_e\left(roll_e^j(\mathbf{y})\right) \oplus \mathbf{k}_j^{out}$ for $j = 0, \ldots, \ell_o - 1$.

The output of Farfalle is the concatenation of bit strings $\mathbf{z}^0\| \cdots \|\mathbf{z}^{\ell_o-1}$.

5.1.2 The KRAVATTE Pseudo-Random Function

KRAVATTE is an instantiation of the Farfalle construction, which specifies the internal components. The Figure 30 outlines the overall primitive that relies on four different Keccak-p permutations [NIS14] on a block size of $b = 1600$ bits. The only distinction between those four permutations named p_b, p_c, p_d and p_e lies in their number of rounds, that we denote respectively by n_b, n_c, n_d and n_e.

Since the first publication, the designers substantially changed the construction of `Farfalle` and KRAVATTE, and as of the IACR ePrint KRAVATTE specification [BDH+16], the permutations p_b and p_c consist of $n_b = n_c = 6$ rounds of the `Keccak`-p permutation, while the remaining two permutations contain $n_d = n_e = 4$ rounds. In a private communication,[2] upon discovery of the higher order differential attack described in Section 5.3, the designers considered increasing the numbers of rounds to $(n_b, n_c, n_d, n_e) = (6, 6, 6, 6)$. The resulting updated version has been presented by the designers at the ECC 2017 conference [BDH+17b].

To conveniently address the various versions throughout the chapter, we use the notation KRAVATTE-(n_d, n_e) to refer to a version with specific n_d (resp. n_e) number of rounds in the permutation p_d (resp. p_e). Our results are independent of p_b and p_c, which is why we do not mention n_b and n_c.

Like in `Keccak`, the 1600-bit state is represented as a $5 \times 5 \times 64$ three-dimensional bit array B, where each bit is denoted $B_{x,y,z}$, with $x, y = 0, \ldots, 4$ and $z = 0, \ldots, 63$. Arithmetic operations performed on indices x, y and z are reduced modulo 5, 5 and 64, respectively, and we omit the modulo for the sake of simplicity.

Additionally, while `Farfalle` uses several rolling functions, the ECC version of KRAVATTE only relies on one, that we simply denote by *roll*, and whose n-th iteration is depicted as $(n\circlearrowleft)$ on Figure 30. More precisely, $roll_b = roll_c = roll_d = roll$ and $roll_e$ is the identity. The *roll* function transforms a state A into $B = roll(A)$ as follows:

$$
\begin{aligned}
B_{x,y,z} &\leftarrow A_{x,y,z} && \text{if } y < 4,\\
B_{x,4,z} &\leftarrow A_{x+1,4,z} && \text{if } x < 4,\\
B_{4,4,z} &\leftarrow A_{0,4,z-7} \oplus A_{1,4,z} && \text{if } z > 60,\\
B_{4,4,z} &\leftarrow A_{0,4,z-7} \oplus A_{1,4,z} \oplus A_{1,4,z+3} && \text{if } z \leq 60.
\end{aligned}
$$

Following the communication of the cryptanalysis to the designers, a tweaked version of Kravatte was released in December 2017 and published at FSE 2018 [BDH+17a], in which the $roll_e$ function is replaced by a non-linear rolling function.

Security Claims. In the original document, the designers of KRAVATTE claim a security of 256 bits when the amount of data does not exceed 2^{137} input and output blocks, that is $\ell_i + \ell_o \leq 2^{137}$.

2 November 5, 2017.

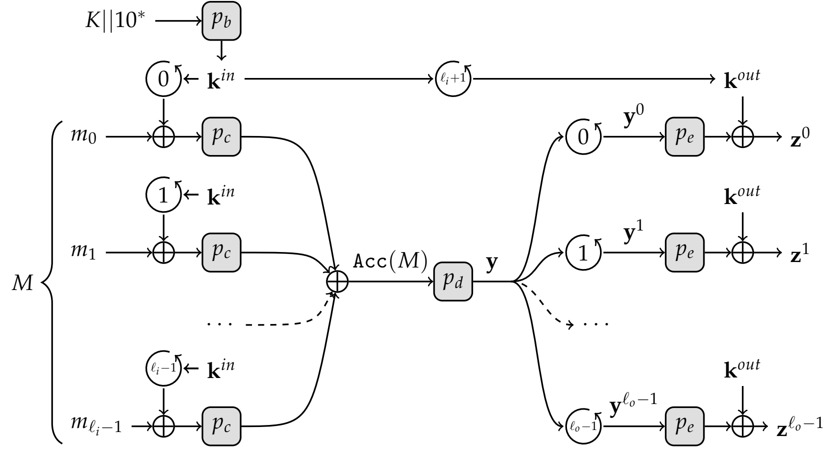

Figure 30: The KRAVATTE primitive. The input message M is padded and split into the b-bit blocks m_i. The function (n) refers to the linear function $x \to roll^n(x)$.

5.1.3 Round Function of the Keccak-p Permutation

We now give a brief description of the Keccak-p permutation, which can also be found in [NIS14]. It is based on the iteration of a round function, defined as the composition of the following operations (in this order), that each produce a state A' from A:

LINEAR DIFFUSION θ: The sum of the five bits of columns with indices $(x-1, z)$ and $(x+1, z-1)$ are added to bit each bit (x, y, z) of the state:

$$A'_{x,y,z} \leftarrow A_{x,y,z} \oplus \sum_{j=0}^{4} A_{x-1,j,z} \oplus \sum_{j=0}^{4} A_{x+1,j,z-1}.$$

LANE-WISE ROTATION ρ: Each lane of the state is rotated by a different number of positions, whose exact values are not relevant for the description of the attacks.

LANE-PRESERVING PERMUTATION π: Lanes positions are switched according to a constant pattern: $A'_{x,y,z} \leftarrow A_{x+3y,x,z}$

SUBSTITUTION LAYER χ: A 5-bit Sbox of degree two is computed on each row (y, z) of the state. More specifically, each output bit depends on three input bits by the following equation (omitting y and z indices):

$$A'_x \leftarrow A_x \oplus \overline{A_{x+1}} \cdot A_{x+2}.$$

CONSTANT ADDITION ι: A round constant produced by an LFSR is XORed to the lane indexed by $(0, 0)$. We omit the exact values of the constants, as they are not relevant to understand the cryptanalysis. We refer the interested reader to [NIS14] for more details.

In the remaining of the chapter, we also use the inverse of the Keccak-p round function, obtained by inverting the sequence of operations. The transformations ι, ρ and π all have straightforward inverses. The inverse Sbox χ^{-1} has algebraic degree three, and omitting y and z indices, its polynomial expression is given by

$$A'_x \leftarrow \overline{A_{x+1}} \cdot \overline{A_{x+3}} \cdot A_{x+4} \oplus \overline{A_{x+1}} \cdot A_{x+2} \oplus A_x.$$

The transformation θ^{-1} is a high-density linear layer whose exact expression is not relevant for the analysis conducted in the chapter. It consists in XOR-ing to each bit of the state the sum of all five bits of about half of the columns of the state, and we note that, for a given column, the value XOR-ed to all the five positions is the same.

5.2 Algebraic Cryptanalysis of Full Kravatte

In this section, we describe key-recovery algebraic attacks against Kravatte-(n_d, n_e) for any n_d and $n_e \in \{4, 6\}$ and for a single message. These attacks rely on a remark on the linearization of the algebraic systems describing iterated Keccak-p^{-1} rounds, on the structure of the expansion layer in the Kravatte construction, and on the small number n_e of Keccak-p rounds in the p_e permutation. We note that these attacks are entirely independent of the compression layer of Kravatte as well as the application of p_d on the accumulator that initiates the expansion layer.

We first describe an attack based on a meet-in-the-middle strategy (Section 5.2.1), which can be enhanced by an observation borrowed from stream-cipher cryptanalysis (Section 5.2.2). This last technique can be further improved by refining the study of the linearization of iterated Keccak-p^{-1} rounds, which is covered in Section 5.4.

5.2.1 Meet-in-the-Middle Algebraic Attack

We present a key-recovery meet-in-the-middle (MITM) algebraic attack on full Kravatte. The key observation underlying the attack is that the *same* unknown value at the output of p_d is used at the input of *all* the branches in the expansion phase. If we denote this value by $\mathbf{y}$, then the j-th output block becomes $\mathbf{z}^j = p_e(roll^j(\mathbf{y})) + \mathbf{k}^{out}$. By considering a system of equations where the unknowns are bits of both $\mathbf{k}^{out}$ and of $\mathbf{y}$, we can mount a meet-in-the-middle attack by equating two states for Branch j: $A^j = B^j$, where A^j corresponds to n_1 forward rounds of Keccak-p applied on $\mathbf{y}^j = roll^j(\mathbf{y})$, and B^j to n_2 backward rounds of Keccak-p applied on $\mathbf{z}^j + \mathbf{k}^{out}$, with $n_1 + n_2 = n_e$. The A-states contain expressions in $\mathbf{y}$ and can be precomputed, while the B-states contain output-dependent expressions in $\mathbf{k}^{out}$. By considering a single

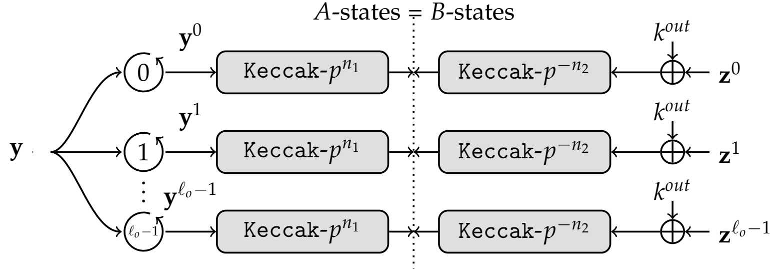

Figure 31: Meet-in-the-middle algebraic attack on Kravatte, with n_1 forward and n_2 backward rounds, $n_1 + n_2 = n_e$.

input message (possibly unknown) together with its ℓ_o-block output, for ℓ_o sufficiently large, we can collect enough equations to form a system that can be solved through linearization, which recovers $\mathbf{k}^{out}$.

Linearization Principle. Linearization is a well-known technique to solve multivariate polynomial systems of equations. It relies on a fairly simple idea: the system of polynomial equations is turned into a system of linear equations by adding new variables that replace all the monomials of the system whose degree is strictly greater than 1. This linear system of equations can be solved using linear algebra if there are enough equations to make the linearized system overdetermined, typically at least on the same order as the number of variables after linearization. All the attacks in this chapter heavily rely on this technique.

In the case of the MITM algebraic attack on Kravatte, the middle state can be described as a polynomial expression in $\mathbf{y}$ bits (resp. $\mathbf{k}^{out}$ bits) in the forward (resp. backward) direction. By linearization and summation of both expressions, one gets a linear equation in $\mathbf{y}$ and $\mathbf{k}^{out}$ monomials, with no composite monomial involving both types of unknowns.

Basic Linearization. The most straightforward way to linearize algebraic expressions in n unknowns of degree limited by d is to introduce a new variable for every monomial in the unknowns of degree at most d. The set of monomials considered has cardinality

$$S(n,d) \overset{\text{def}}{=} \sum_{i=1}^{d} \binom{n}{i}.$$

This approach can be used directly in the context of the MITM algebraic attack on Kravatte. The algebraic degree of Keccak-p (resp. Keccak-p^{-1}) is two (resp. three) and *roll* is linear, thus the number of monomials involved by a basic linearization is approximately $S(b, 2^{n_1}) + S(b, 3^{n_2})$. We give in Table 10 the number of monomials required to describe the forward and backward parts of the meet-in-the-middle algebraic system.

Table 10: Number of monomials in input (resp. output) variables after n rounds of Keccak-p or Keccak-p^{-1} for $b = 1600$ ($\log_2$ scale).

n	Keccak-p	Keccak-p^{-1}	
		Basic	Improved
1	20.3	29.3	13.0
2	38.0	77.3	36.5
3	69.8	194.0	106.4

Improved Linearization in the Backward Direction The number of monomials to consider in the backward direction can be drastically reduced if we take into account the row structure of χ^{-1} non-linear layers and the absence of diffusion before the first χ^{-1} layer encountered. Indeed, through the backward computations, new monomials are only created in χ^{-1} layers through multiplicative combination of input sum of monomials. There are two limiting factors to the combination power of the χ^{-1} layers. First, the χ^{-1} has only degree three, restricting the newly created monomials to the product of at most three input monomials. Secondly, it operates on only five inputs, which has a significant effect since input monomials for the external χ^{-1} layer can be limited by position: as no diffusion takes places after the unmasking by $\mathbf{k}^{out}$, only five $\mathbf{k}^{out}$ bit variables can occur in the monomials occurring at the output of a given Sbox. Consequently, the number of monomials required to express the outputs of an Sbox is upper bounded by $S(5,3)$, so that expressing the outputs of all Sboxes only requires $N = \frac{b}{5}S(5,3)$ monomials. The input bits of internal χ^{-1} layers have undergone linear diffusion, so they cannot be restricted in the same manner. However, the degree limitation still applies, and since N monomials can be used to describe the polynomial expressions of all bits before the χ^{-1} layer, the number of monomials that appear in the output bits of this layers is upper-bounded by $S(N,3)$. This can be iterated to cover more rounds. Note that the improved linearization does not apply in the forward direction due to the application of *roll* and θ prior to the first χ layer. We give in Table 10 estimates for the number of monomials to consider for a small number of Keccak-p rounds in the backward direction.

For $n_e = 4$, choosing $n_1 = n_2 = 2$, Kravatte can be attacked by a meet-in-the-middle algebraic attack. The attack requires $\frac{1}{1600}(2^{38.0} + 2^{36.5}) \approx 2^{27.8}$ output blocks to get enough equations, has memory complexity $(2^{38.0} + 2^{36.5})^2 \approx 2^{76.9}$ bits to represent the system and the time for the resolution of the linearized system is at most cubic in the number of monomials, which yields about $(2^{38.0} + 2^{36.5})^3 \approx 2^{115.3}$ elementary operations.

The time complexity to build the system boils down to the construction of the $2^{27.8}$ equations, where each requires to compute the expressions coming from both sides of the meet-in-the-middle. For one equation, the backward contribution is dominated by the product of three polynomials in $320S(5,3) = 8000$ monomials, while the forward contribution is dominated by the multiplication of two polynomials of at most $S(1600,2) \approx 2^{20.29}$ monomials in $\mathbf{y}$. All in all, the time complexity is dominated by the time for solving the system.

The above observations about how to linearize the backward computation of up to two last rounds of Keccak-p and the results of Table 10 will be re-used in the key-recovery part of the attacks introduced in Section 5.2.2 and Section 5.3.

5.2.2 Cancellation of Monomials Using a Linear Recurrence

We now describe a second attack that exploits the linearity of the rolling function to cancel the monomials in $\mathbf{y}$ from the system. Indeed, after the application of p_d, the first half of the remaining expansion layer can be seen as a filtered linear recurrent sequence of states, with update function *roll* and output function Keccak-p^{n_1}. Filtered linear recurrences, e.g. filtered LFSR, are classic stream cipher constructions, which have been deeply studied. A line of work [Key76, RH07, RGH07] observes that not only do the LFSR state bits follow linear recurrences, but the same holds for the monomials that are formed from these bits.

In this section, we start by exposing a recurrence polynomial of sequences of values taken by bits of the rolling state. We then show how this can be generalized to obtain a recurrence polynomial for sequence of values taken by products of bits of the rolling state. As the state A^j can be expressed as a sum of such products of $\mathbf{y}^j$ bits, this constitutes a linear complexity distinguisher on partial Kravatte, with the last n_2 Keccak-p rounds and final masking removed. Finally, we show this can be used to combine equations of the system describing the expansion phase of Kravatte to eliminate the monomials in $\mathbf{y}$.

Linear Recurrence of Rolling State Bits. As stated above, the beginning of the expansion layers acts like a LFSR filtered by the fixed non-linear function Keccak-p^{n_1}. After the initial value $\mathbf{y} = p_d(\texttt{Acc}(M))$ of the rolling state is formed, it is updated linearly through the rolling function *roll*: the value of the rolling state that appears at the start of Branch j is given by $\mathbf{y}^j = roll^j(\mathbf{y})$. The rolling function *roll* is a linear transformation of the rolling state that leaves Planes 0 to 3 unchanged. The matrix $\mathbf{M}_{roll}$ of size 320 describing how *roll* affects Plane 4 has a primitive characteristic polynomial P_{roll} of degree

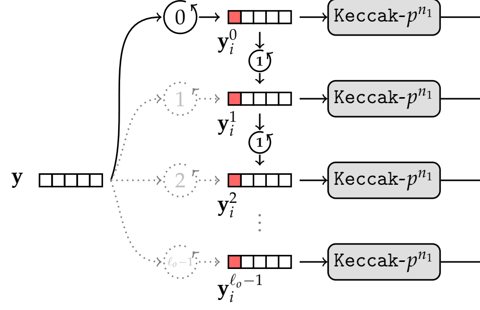

Figure 32: Linear recurrence in the KRAVATTE branches: the sequence $(\mathbf{y}_i^j)_j$ of highlighted bits at a prescribed Position i across the branches $j = 0, \ldots, \ell_o - 1$ follows a linear recurrence described by the polynomial $(X+1) \cdot P_{roll}$.

320. By the Cayley-Hamilton theorem, we know that $P_{roll}(\mathbf{M}_{roll}) = 0$. We can associate to Bit i of Plane 4 a vector $\mathbf{e}_i$ of the standard basis of $\mathbb{F}_2^{320}$, and the values taken by the Bit i of Plane 4 of the state $\mathbf{y}^j$ in Branch j is then given by $\mathbf{e}_i^T \cdot \mathbf{M}_{roll}^j \cdot \mathbf{y}_{roll}$, where $\mathbf{y}_{roll}$ is the restriction of $\mathbf{y}$ to Plane 4. Then, we observe that the sequence of values taken by a given state bit of the part affected by *roll* over the branches of the expansion layer (see Figure 32) is a linear recurrence sequence with recurrence polynomial P_{roll}. Indeed, noting $P_{roll} = \sum_n c_n X^n$, we have for all j:

$$\begin{aligned} P_{roll}(\mathbf{y}_i^j) &= \mathbf{e}_i^T \cdot \left(\sum_n c_n \mathbf{M}_{roll}^{j+n} \right) \cdot \mathbf{y}_{roll} \\ &= \mathbf{e}_i^T \cdot \left(\mathbf{M}_{roll}^j \cdot P_{roll}(\mathbf{M}_{roll}) \right) \cdot \mathbf{y}_{roll} \\ &= 0. \end{aligned}$$

Furthermore, the bits in Planes 0 to 3 follow a linear recurrence with recurrence polynomial $X+1$, so all the state bits follow the linear recurrence given by $(X+1) \cdot P_{roll}$.

Linear Recurrence of Monomials Formed on the Rolling State. This can be generalized to the monomials formed from the bits of the rolling state. In the same way that $\mathbf{M}_{roll}$ describes the evolution of state bits, one can consider the matrix $\mathbf{M}_{roll}^{\leq d}$ describing the evolution through *roll* of monomials of degree at most d in the $\mathbf{y}^j$ state bits, since the transformation is also linear on this set. Indeed, the updated value of every state bit after *roll* is a linear combination of state bits before the update, so the product of d updated values can be written, by developing the product of the linear combinations, as a linear combination of monomials of state bits before update with degree at most d. The characteristic polynomial $P_{roll}^{\leq d}$ of this matrix provides a recurrence polynomial for all $S(320, d)$ monomials of degree at most d in the 320 variables of Plane 4. It is also a recurrence polynomial for monomials of

degree at most d in all state variables with at least one variable coming from Plane 4, since the product of variables of Planes 0 to 3 is constant and can be factored out, leaving a monomial of degree strictly less than d of variables from Plane 4. Monomials with variables only from Planes 0 to 3, together with the constant monomial, are constant and have $X+1$ as recurrence polynomial. Thus, $(X+1) \cdot P_{roll}^{\leq d}$ is a recurrence polynomial for all monomials of degree at most d in all 1600 variables of the rolling state. Since Keccak-p has degree two, the recurrence polynomial $(X+1) \cdot P_{roll}^{\leq 2^{n_1}}$ cancels the sequences of all monomials involved in the algebraic expression of the outputs of the first part of the expansion layer. Its degree is $S(320, 2^{n_1}) + 1$. We give estimates of this value for $n_1 = 2, 3, 4$ in Table 11. For larger values of n_1, the technique is not applicable since the degree of the polynomial, e.g., $2^{146.5}$ for $n_1 = 5$, and $2^{227.3}$ for $n_1 = 6$ goes beyond the limit set on the data complexity in the security claims of KRAVATTE.

Computing the Recurrence Polynomial for Monomials of Degree at Most d. Computing the characteristic polynomial of a matrix usually requires to compute a determinant, but can be done in the case of $P_{roll}^{\leq d}$ in time quasilinear in the size of the matrix $\mathbf{M}_{roll}^{\leq d}$, without even forming the matrix, due to algebraic properties of linear recurring sequences. Indeed, it has been shown in [Key76] that the roots of this polynomial are all simple and elements of the algebraic extension $\mathbb{F}_2[X]/P_{roll}$. Denoting by α the class of X, they are given by α^t, were $t \in [1, 2^{320} - 1]$ takes all values with Hamming weight at most d. Thus, $P_{roll}^{\leq d}$ can be formed as the product of $N = S(320, 2^{n_1})$ polynomials of the form $X + \alpha^t$.

In a first stage, the polynomials whose roots are conjugates are multiplied together, resulting in a set of irreducible polynomials in $\mathbb{F}_2[X]$. In a second stage, these polynomials are multiplied together. Multiplication of two polynomials in $\mathbb{F}_2[X]$ of degree at most n can be performed efficiently for large n using the Schönhage algorithm [Sch77], with asymptotic complexity $M(n) = O(n \log n \log\log n)$. This algorithm has been implemented in the gf2x library [BGTZ08] and experimental data indicates the hidden constant is small. To take full advantage of fast polynomial multiplications, the computation of $P_{roll}^{\leq d}$ can be performed tree-wise: At each step, polynomials are multiplied by pairs, resulting in a set of half as many polynomials with double degree. Taking into account the asymptotic complexity of fast polynomial multiplication, the complexity T_P of the computation can be estimated by

$$\sum_{i=0}^{\log N} 2^i M\left(\frac{N}{2^i}\right) \leq N \log^2 N \log\log N.$$

We give estimates of this time complexity T_P for small values of n_1 in Table 11.

Impact on the Cryptanalysis of KRAVATTE**.** Using the linear recurrence given by the polynomial $(X+1) \cdot P_{roll}^{\leq 2^{n_1}} = \sum_j d_j X^j$, we eliminate all monomials in $\mathbf{y}$ from the system. More precisely, the i-th equation is obtained by summing $A^{i+j} \oplus B^{i+j} = 0$ equations:

$$\sum_j d_j \cdot \texttt{Keccak-}p^{n_1}\left(roll^{i+j}(\mathbf{y})\right) \oplus \sum_j d_j \cdot \texttt{Keccak-}p^{-n_2}\left(\mathbf{k}^{out} \oplus \mathbf{z}^{i+j}\right) = 0,$$

and since the first sum is null due to the relation from the linear recurrence, this yields

$$\sum_j d_j \cdot \texttt{Keccak-}p^{-n_2}(\mathbf{k}^{out} \oplus \mathbf{z}^{i+j}) = 0. \tag{1}$$

As a consequence, in the case $n_2 = 2$, the number of monomials to write this system goes down to $2^{36.5}$, since only the monomials in $\mathbf{k}^{out}$ remain (see Table 10). Each equation of the system requires $S(320, 2^{n_1})$ consecutive output blocks to be formed, but since a sliding-window mechanism can be used to form the equations, only $S(320, 2^{n_1}) + \frac{1}{1600}2^{36.5}$ blocks are necessary to form the system. We can process the available blocks on the fly: For each block, we add its contribution to the system of $\mathbf{k}^{out}$ monomials in the equations prescribed by the recurrence polynomial, i.e., Block $\mathbf{z}_j$ contributes to Equation i if $d_{j-i} = 1$. This does not increase the memory complexity, but increases the time complexity of building the system by a factor of $S(320, 2^{n_1})$.

Applying this attack with $n_1 = 2$, we can attack KRAVATTE-$(n_d, 4)$ for any n_d. The recurrence polynomial to store has degree $2^{28.7}$, so to collect enough data to solve the linearized system, we need $2^{28.7} + \frac{1}{1600}2^{36.5} \approx 2^{28.8}$ output blocks. Computing the recurrence polynomial requires about $2^{40.7}$ basic operations(see Table 11). Solving the linearized system requires $T_{solve} = (2^{36.5})^3 \approx 2^{109.5}$ operations, and a memory of $\left(2^{36.5}\right)^2 \approx 2^{73}$ bits. However, the most time-consuming part of the attack resides in the construction of the system. For each output block and every bit of the intermediate state, the bit is written as a linearized algebraic expression of bits in $\mathbf{k}^{out}$, depending on $\mathbf{z}^j$, as explained at the end of Section 5.2.1. Then, this expression is added to the equations it contributes to build, depending on the recurrence polynomial. Every equation is built by the addition of at most $S(320, 2^{n_1})$ contributions, and the cost of adding one contribution is given by the size of an expression, which is about $2^{36.5}$. Computing the algebraic expression of one bit essentially boils down to the multiplication of three polynomials in $320S(5,3) = 8000$ monomials each that appear in the inversion of $n_2 = 2$ rounds of KRAVATTE. Overall, constructing the system amounts to approximately

$$T_{build} = \left(S(320, 2^{n_1}) + \frac{1}{1600}2^{36.5}\right) \cdot 1600 \cdot 8000^3 + S(320, 2^{n_1}) \cdot \left(2^{36.5}\right)^2$$

operations. We give estimations of the overall time complexity $T_P + T_{build} + T_{solve}$ in Table 11.

With $n_1 = 4$, the attack breaks the security claim of Kravatte-$(n_d, 6)$ for any n_d. Indeed, the recurrence polynomial has degree $2^{88.4}$ and can be computed in about 2^{104} simple operations, which allows to collect the equations using about $2^{88.4}$ output blocks. Then, the system can be constructed as before (with non-optimized computations, it requires about $2^{161.4}$ basic operations) and solved similarly as in the case $n_1 = 2$.

We summarize the attack complexities in Table 11. In this table, the two first lines provide attacks for the ePrint version of full Kravatte-$(4, 4)$, and the last line gives an attack for the strengthened variant announced at ECC 2017.

The Particular Case of One Backward Round. In the case of one backward round, i.e., $n_2 = 1$, (1) can be solved by exhaustive search. Note that the linear layer of the round considered can be removed from the analysis, and thus no diffusion takes place. As a consequence, it is possible to recover $\mathbf{k}^{out}$ Sbox by Sbox, by guessing the five $\mathbf{k}^{out}$ bits corresponding to a given Sbox, and checking that sums of $\chi^{-1}(\mathbf{k}^{out} \oplus \mathbf{z}^j)$ over positions j determined by the recurrence polynomial yields zero, which is the case for the correct guess. For each sum, this gives a $t = 5$-bit test on the $g = 5$ guessed bits of $\mathbf{k}^{out}$. With only one sum, the probability that no false alarm occurs is $(1 - 2^{-t})^{2^g - 1} \approx 0.37$, so the rate of Sboxes with false alarms on corresponding $\mathbf{k}^{out}$ bits is too high to recover the complete $\mathbf{k}^{out}$ by key enumeration. However, with two sums, one gets a $t = 10$-bit test, and the probability of absence of false alarms raises to 0.97, which amounts to about 10 Sboxes with false alarms, making a final offline key candidate enumeration possible.

With $(n_1, n_2) = (3, 1)$, it is thus possible to attack Kravatte-$(n_d, 4)$ for any n_d with a data complexity of $2^{51.2}$ blocks and a time complexity dominated by the recurrence polynomial computation time $T_p = 2^{65.1}$. The attack requires to precompute and store $P_{roll}^{\leq 3}$ and thus has memory complexity $2^{51.2}$.

5.3 Higher Order Differential Cryptanalysis of Full Kravatte

In this section, we highlight the existence of higher order differential attacks against Kravatte. We describe in Section 5.3.1 a property of the compression layer of `Farfalle`, which weakens the overall construction against higher order differential attacks. The process of our attack is depicted in Section 5.3.2.

Table 11: Degree and computation time of recurrence polynomial for all monomials in $\mathbf{y}$ after n_1 rounds of Keccak-p, and attack complexity against Kravatte-(n_d, n_e), for any n_d and $n_e = n_1 + n_2$. For optimized attacks, see Section 5.4.

$\mathbf{n_e}$	$\mathbf{n_1 + n_2}$	$\mathbf{deg}\left(\mathbf{P}_{\mathbf{roll}}^{\leq \mathbf{d}}\right)$	$\mathbf{T_P}$	**Data***	**Memory***	**Time***
4	$2+2$	$2^{28.7}$	$2^{40.7}$	$2^{29.3}$	$2^{73.0}$	$2^{109.5}$
4	$3+1$	$2^{51.2}$	$2^{65.1}$	$2^{51.2}$	$2^{51.2}$	$2^{65.1}$
6	$4+2$	$2^{88.4}$	$2^{104.0}$	$2^{88.4}$	$2^{88.4}$	$2^{161.4}$

*: These complexities are given without the optimizations addressed in Section 5.4.

To experimentally validate the correctness of the approach, we use a round-reduced variant of Kravatte.

In Section 5.3.3, we show how an adversary can use the higher order distinguisher to mount a chosen-message key-recovery attack against Kravatte-(n_d, n_e) such that $n_d + n_e \leq 8$.

In the last section of this chapter dedicated to various optimizations, we present a variant of this attack allowing to improve the overall data complexity (Section 5.4.2) and various techniques to substantially decrease the complexities.

5.3.1 Construction of Affine Spaces in the Accumulator

We describe here a property of the compression layer of Farfalle, already identified in [BDH+16, Section 5.4], that enables an adversary to construct an affine space of dimension n in the accumulator block. Given an n-block padded message $M = (m_0, \ldots, m_{n-1})$, we recall that we denote $\texttt{Acc}(M)$ the associated accumulator value $\sum_{i=0}^{n-1} p_c\left(m_i \oplus \mathbf{k}_i^{in}\right)$.

Let $M^0 = (m_0^0, \ldots, m_{n-1}^0)$ and $M^1 = (m_0^1, \ldots, m_{n-1}^1)$ denote an arbitrary pair of padded messages such that $m_i^0 \neq m_i^1$ for all i. These messages are used to build the following structure of 2^n n-block input messages: $\mathcal{S} = \{(m_0^{\epsilon_0}, \ldots, m_{n-1}^{\epsilon_{n-1}}), (\epsilon_0, \ldots, \epsilon_{n-1}) \in \{0,1\}^n\}$.

We denote by δ_i the one-block difference $\delta_i = p_c\left(m_i^0 \oplus \mathbf{k}_i^{in}\right) \oplus p_c\left(m_i^1 \oplus \mathbf{k}_i^{in}\right)$. If $n \ll b = 1600$, the δ_i are linearly independent with overwhelming probability. It is easy to see that $\texttt{Acc}(\mathcal{S})$ is then the n-dimensional affine subspace $\texttt{Acc}(M_0) \oplus \langle \delta_0, \ldots, \delta_{n-1} \rangle$.

In other words, we can easily build structures of 2^n n-block messages that are transformed by the compression layer into an affine space of one-block accumulator values of dimension n. Note that this does not depend on the number of rounds in p_c.

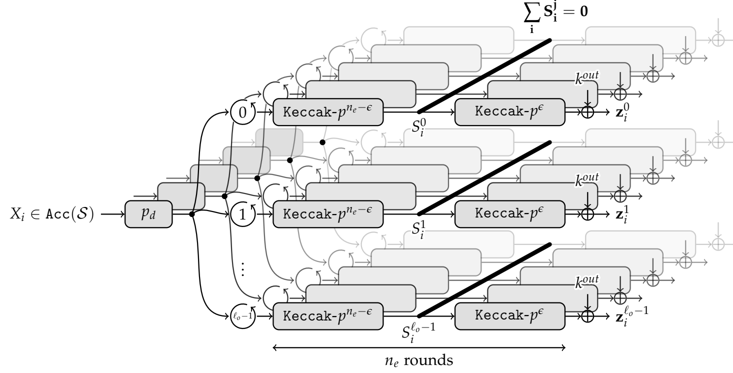

Figure 33: Higher order differential distinguisher on Kravatte. Summing over the whole affine space Acc($\mathcal{S}$) the states obtained after application of $\ell = n_d + n_e - \epsilon$ rounds to the blocks X_i of the affine space, i.e., summing along every bold line, yields zero.

5.3.2 Higher Order Differential Attacks Against Kravatte

We can use the property of Farfalle described above to mount higher order differential attacks on Kravatte-(n_d, n_e), as long as $n_d + n_e \leq 8$.

Summing the images of a function f over an affine subspace of dimension n is equivalent to applying the n-th differential of f to an element of the subspace [Lai94]. The round function of the Keccak-p permutation used in the Kravatte instance is of algebraic degree two. Hence, the partial expansion layer, starting from the accumulator value and applying ℓ permutation layers, is of degree 2^ℓ. By building an affine space of dimension $n = 2^\ell + 1$ with an input structure $\mathcal{S}$ of 2^n messages, each one containing n blocks, the sum over this affine space of the intermediate values after the partial expansion is zero (see Figure 33).

This distinguishing property can then be used to mount last-round attacks: Starting from the Kravatte output values of the plaintext in the structure, by inverting the last $\epsilon = n_d - n_e - \ell$ permutation layers of the expansion layer, and summing all the contributions, one gets equations on the output key $\mathbf{k}^{out}$: namely, $\sum_{m \in \mathcal{S}}$ Keccak-$p^{-\epsilon}(\mathbf{k}^{out} \oplus \mathbf{z}^m) = 0$.

To demonstrate the validity of the higher order differential distinguisher described above, we applied it on ShortKravatte. In ShortKravatte, $n_d = 0$ so that the expansion layer consists of four rounds of the Keccak-p permutation, instead of $n_d + n_e = 8$ rounds for the full Kravatte instance. Hence, with a structure of 2^{16+1} input messages of 17 blocks, the higher order differential distinguisher spans the whole expansion layer and can be observed by summing directly the output values of the messages in the structure.

5.3.3 Last-Round Attacks

One Last-Round Attack. One can apply the higher order differential strategy to mount a basic key-recovery attack against Kravatte-(n_d, n_e), $n_d + n_e \leq 8$, by considering a 7-round partial expansion layer and a final last-round (i.e., $\epsilon = 1$). This implies to use structures of $2^{2^7+1} = 2^{129}$ messages of 129 blocks. The higher order differential distinguisher continues to apply through the linear layer of the last round. Thus, no diffusion takes place in the inverted part of the last round, and the key-recovery method given at the end of Section 5.2.2 for the case of one backward round applies, with the small variation that one gets $t = 10$-bit tests by requesting two output blocks per message. This attack has a time and data complexity of $2^{129} \times (129 + 2) \approx 2^{136.0}$, and negligible memory complexity.

We have implemented this attack on a reduced version of Kravatte where $n_d + n_e = 5$ using structures of 2^{17} messages of 17 blocks, and find that the number of candidates for $\mathbf{k}^{out}$ is reduced from 2^{256} to about 2^{18}. We note that $\mathbf{k}^{out}$ can be uniquely determined using three to six output blocks.

Two Last-Round Attack. We now describe how to improve the time and data complexity, leveraging the analysis of the algebraic expression of Keccak-$p^{-2}(\mathbf{k}^{out} \oplus \mathbf{z})$, in the same way as for the previous attack (Section 5.2.1). This enables to consider a higher order distinguisher over $\ell = 6$ Keccak-p rounds (i.e., $\epsilon = 2$). The adversary builds a structure $\mathcal{S}$ of 2^{65} plaintexts of 65 blocks as described above. As a consequence, the 2^{65} intermediate values at any bit position before the penultimate non-linear layer sum to zero.

As described in the previous section (see Table 10), the number of monomials involved in the system corresponding to the inversion of the two last rounds of Kravatte is about $2^{36.5}$, and the system can be solved if one collects about the same number of equations. These can be obtained considering, for every message of the structure, outputs of $\frac{1}{1600}2^{36.5} \approx 2^{25.9}$ blocks. The total data complexity (expressed as the sum of all input and output blocks) of this attack is therefore $2^{65} \times (65 + 2^{25.9}) \approx 2^{90.9}$ blocks. The system of equations can be computed on the fly, therefore there is no need to store all the inputs and outputs of Kravatte. However, storing the system requires $(2^{36.5})^2 = 2^{73.0}$ bits of memory. The evaluation of the time complexity is more involved, as one needs to consider all the steps of the attack. For each equation and each input, the most expensive step consists in computing the penultimate χ^{-1} layer, which requires to get the product of three polynomial expressions, each of which can contain up to 8000 monomials. Therefore, we can estimate the complexity of this step of the attack to $2^{36.5} \times 2^{65} \times 8000^3 \approx 2^{140.4}$ bit operations. Solving the system of equations is

far less expensive, as its time complexity is at most cubic in the number of equations, which leads to $(2^{36.5})^3 = 2^{109.5}$ operations.

As we show in Section 5.4, these "naive" data and time complexities can be substantially improved using various optimizations.

5.4 Optimization Techniques for the Cryptanalysis

5.4.1 Minimizing the Number of Variables for Two Inverse Rounds

In this section, we improve further the linearization of `Keccak`-p^{-2}. Notations used in the following are summarized on Figure 34.

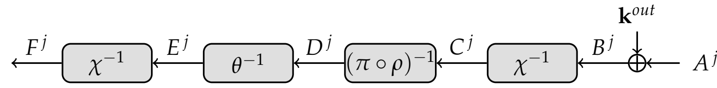

Figure 34: Notations used in Section 5.4.1.

The attacks described in Section 5.2 and Section 5.3 are all based on the construction and the linearization of polynomial expressions whose variables are the bits of $\mathbf{k}^{out}$. Each bit of F^j can be expressed as a low-degree polynomial in key bits $\mathbf{k}^{out}$, whose coefficients are functions of the output states A^j. We have seen in Section 5.2.1 that using the row structure of χ^{-1} layers and the absence of diffusion before the first χ^{-1} layer enables to decrease the number of variables to be considered in the linearized expressions. We now show that this can be improved further by additionally considering the number of monomials in the polynomial expression of χ^{-1}, by limiting the number of considered bit positions of F^j and by summing two positions of F^j. We note that this improvement comes at the cost of a slight increase of the data complexity, since only one equation is extracted from each output block. We also note that this last optimization is compatible with the attacks presented in Section 5.2 and Section 5.3 because the sum over sets of output block messages they consider are performed consistently on all the positions of the blocks. We study each of the successive layers of the backward computation of KRAVATTE.

External χ^{-1} Layer. The inverse Sbox has algebraic degree three. More precisely, we get the following expressions (we omit indexes y and z and the block number j):

$$\begin{aligned} C_x &= \overline{B_{x+1}B_{x+3}}B_{x+4} \oplus \overline{B_{x+1}}B_{x+2} \oplus B_x \\ &= \left(\mathbf{k}^{out}_{x+1} \oplus A_{x+1} \oplus 1\right)\left(\mathbf{k}^{out}_{x+3} \oplus A_{x+3} \oplus 1\right)\left(\mathbf{k}^{out}_{x+4} \oplus A_{x+4}\right) \\ &\oplus \left(\mathbf{k}^{out}_{x+1} \oplus A_{x+1} \oplus 1\right)\left(\mathbf{k}^{out}_{x+2} \oplus A_{x+2}\right) \oplus \left(\mathbf{k}^{out}_{x} \oplus A_{x}\right). \end{aligned}$$

Introducing the new variables $w_x = \mathbf{k}^{out}_{x+1}\mathbf{k}^{out}_{x+3}\mathbf{k}^{out}_{x+4} \oplus \mathbf{k}^{out}_{x+1}\mathbf{k}^{out}_{x+2} \oplus \mathbf{k}^{out}_{x}$, $u_x = \mathbf{k}^{out}_{x+3}\mathbf{k}^{out}_{x+4} \oplus \mathbf{k}^{out}_{x+2}$, and $v_x = \mathbf{k}^{out}_{x}\mathbf{k}^{out}_{x+2}$, this can be rewritten as $w_x \oplus P_x(A)$, where $P_x(A)$ is an affine combination of u_x, v_{x+1}, v_{x+4}, $\mathbf{k}^{out}_{x+1}$, $\mathbf{k}^{out}_{x+3}$, $\mathbf{k}^{out}_{x+4}$, with coefficients determined by A. The definition of new variables for the sum of monomials sharing the same coefficient instead of for each monomial enables us to limit the number of variables per bit of the C state to 8, including the variable w with constant coefficient and the coefficient of the degree-0 monomial. The total number of variables over the state is however the same with both approaches. For each of the 320 Sboxes, one generates 10 variables u, v of algebraic degree two in key bits and 5 variables w of algebraic degree three. Taking into account the key bits, the total number of variables is therefore $320 \times (5 + 10 + 5) = 6400$ at that point. This already improves on Section 5.2.1 since all degree-3 variables are not considered anymore.

Intermediate Inverse Affine Layer. The linear layers consist in the bit-moving layers ρ and π and the linear diffusion layer θ. All these layers do not create new monomials: monomials are simply moved or added to other polynomial expressions. The linear layers only contribute indirectly to the complexity of the algebraic expressions by breaking any independent subset, leading to consider that any bit of the state can be affected by monomials in variables coming from any position of the final state A. This is especially true for the high-diffusion transformation θ^{-1}, whose output bits depend on approximately half of its input state. This has the effect to allow the creation during the next non-linear layer of nearly all the combinations of monomials output by the previous non-linear layer.

More precisely, ρ^{-1} and π^{-1} move every bit of the state. We denote σ the permutation such that bit $\sigma(x, y, z)$ of the state is moved to position (x, y, z). Then, θ^{-1} is a linear diffusion layer with the following property. For each column $D_{x,z}$ of the state, there is a set of bit positions $S_{x,z}$ such that each bit after θ^{-1} is given by

$$E_{x,y,z} = D_{x,y,z} \oplus \sum_{(x',y',z') \in S_{x,z}} D_{x',y',z'}.$$

Therefore, we have

$$\begin{aligned} E_{x,y,z} &= C_{\sigma(x,y,z)} \oplus \sum_{(x',y',z') \in S_{x,z}} C_{\sigma(x',y',z')} \\ &= w_{\sigma(x,y,z)} \oplus P_{\sigma(x,y,z)}(A) \oplus \sum_{(x',y',z') \in S_{x,z}} \left(w_{\sigma(x',y',z')} \oplus P_{\sigma(x',y',z')}(A) \right) \\ &= w'_{x,y,z} \oplus P_{\sigma(x,y,z)}(A) \oplus Q_{x,z}(A). \end{aligned}$$

In this expression, $w'_{x,y,z}$ is a new variable defined as the linear combination of all the w variables involved in the expression of $E_{x,y,z}$, and $Q_{x,z}(A)$ is the

sum of the P's over position set $S_{x,z}$. Please note that each $Q_{x,z}$ is considered to potentially involve all the u, v and $\mathbf{k}^{out}$ variables, whereas $P_{x,y,z}$ only has 7 potentially nonzero variables. Also, w variables influence of E and F is completely given by w' variables, which are independent of A. Therefore, w' variables can replace w variables in the description of the algebraic expressions of all F^j.

Partial Internal χ^{-1} Layer. We now consider only two bits of information from the state F^j in a same column, e.g., at Positions $(0,0,0)$ and $(0,1,0)$, and sum their algebraic expressions. With this approach, we cancel out the multiplication of the term contributing the most monomials to the expressions of these bits, decrease the total number of variables and therefore limit the time and memory complexity of our attack by reducing the complexity of the final linearized system. To this end, we denote $P'_{x,y,z} = w'_{x,y,z} + P_{\sigma(x,y,z)}$. Omitting index z, we have:

$$\begin{aligned} F_{0,y} = &\left(\overline{P'_{1,y}} \oplus Q_1\right)\left(\overline{P'_{3,y}} \oplus Q_3\right)\left(P'_{4,y} \oplus Q_4\right) \\ &\oplus \left(\overline{P'_{1,y}} \oplus Q_1\right)\left(P'_{2,y} \oplus Q_2\right) \oplus \left(P'_{0,y} \oplus Q_0\right). \end{aligned}$$

When considering $F_{0,0} \oplus F_{0,1}$, all products of Q-components cancel out, as Q polynomials are identical over a column. In particular, all arbitrary products of three u, v and $\mathbf{k}^{out}$ variables do not occur anymore. We get:

$$\begin{aligned} F_{0,0} \oplus F_{0,1} = &\left(P'_{1,0} \oplus P'_{1,1}\right) Q_3 Q_4 \oplus \left(P'_{3,0} \oplus P'_{3,1}\right) Q_1 Q_4 \oplus \left(P'_{4,0} \oplus P'_{4,1}\right) Q_1 Q_3 \\ &\oplus \left(\overline{P'_{1,0}P'_{3,0}} \oplus \overline{P'_{1,1}P'_{3,1}}\right) Q_4 \oplus \left(\overline{P'_{1,0}}P'_{4,0} \oplus \overline{P'_{1,1}}P'_{4,1}\right) Q_3 \\ &\oplus \left(P'_{2,0} \oplus \overline{P'_{3,0}}P'_{4,0} \oplus P'_{2,1} \oplus \overline{P'_{3,1}}P'_{4,1}\right) Q_1 \oplus \left(P'_{1,0} \oplus P'_{1,1}\right) Q_2 \\ &\oplus \left(P'_{0,0} \oplus \overline{P'_{1,0}}P'_{2,0} \oplus \overline{P'_{1,0}P'_{3,0}}P'_{4,0} \oplus P'_{0,1} \oplus \overline{P'_{1,1}}P'_{2,1} \oplus \overline{P'_{1,1}P'_{3,1}}P'_{4,1}\right). \end{aligned}$$

All the Q polynomials are affine combinations of the same set of all the $320 \times (5+10) = 4800$ $\mathbf{k}^{out}, u$ and v variables, and each P' polynomial is an affine combination of 7 variables. Taking into account the constant coefficients of these polynomials, the number of variables required to linearize the expression of $F_{0,0} \oplus F_{0,1}$ is therefore:

$$3 \times 2 \times 8 \times \binom{4801}{2} + \left(3 \times 2 \times 8^2 + 2 \times 2 \times 8\right) \times 4801 + 2 \times \left(8 + 8^2 + 8^3\right),$$

which gives approximately $2^{29.0}$ variables, instead of the approximately $2^{36.5}$ monomials obtained with the simpler bound from Section 5.2.

Note that there is a trade-off between the number of variables of the linearized system and the number of equations that are obtained from on block. Indeed, by considering more than one pair of positions, additional w' variables have to be considered, together with the monomials resulting from the

products of these variables with $\mathbf{k}^{out}$, u and v variables. We do not investigate further this trade-off, since we are mainly concerned with the reduction of the size of the system in order to improve the attacks time complexity, and because this reduction of the system size limits the degradation of the data complexity.

5.4.2 Super Structure of Input Messages

As already presented in Section 5.3, the higher order differential distinguisher used for the attack is based on the sum of output messages over a structure of input messages of dimension $n = 65$. Due to the property of `Farfalle` and the algebraic degree of the `Keccak`-p round function, this is guaranteed to lead to a sum of corresponding intermediate states equal to zero. In order to improve the data complexity, we use a super structure of messages, from which structures of dimension 2^n can be extracted. This technique has been used previously, e.g, in [DLMW15].

Principle. Let us consider the set of messages obtained by the concatenation of $n+t$ blocks, Block j being chosen among two possibilities $(m_j^i)_{i\in\{0,1\}}$:

$$\mathcal{S} = \left\{(m_0^{\epsilon_0}, \cdots, m_{n+t-1}^{\epsilon_{n+t-1}}), (\epsilon_0, \cdots, \epsilon_{n+t-1}) \in \{0,1\}^{n+t}\right\}.$$

We can then extract from this super structure several n-dimensional structures by fixing the values of the blocks at t given positions, and subsequently build from each of these structures equations in the output key bits $\mathbf{k}^{out}$.

When extracting n-dimensional structures from an $n+t$ super structure, care has to be taken to avoid linear dependencies that decrease the expected amount of equations that can be formed. Indeed, let us consider $\mathcal{S}_*$, an $(n+1)$-dimensional super structure, and the n-dimensional structures $\mathcal{S}_{0*} = m^0 * \ldots *$, $\mathcal{S}_{1*} = m^1 * \ldots *$, $\mathcal{S}_{*0} = * \ldots * m^0$ and $\mathcal{S}_{*1} = * \ldots * m^1$, where $*$ denotes consecutive positions with two possible values at each position, and 0 (resp. 1) denotes a position with fixed m^0 (resp. m^1), value. Then, we have $\mathcal{S}_{0*} \cup \mathcal{S}_{1*} = \mathcal{S}_* = \mathcal{S}_{*0} \cup \mathcal{S}_{*1}$. As a consequence, given the sum of the intermediate states over three of these five structures, we can derive linearly the sum over the remaining structures, and thus they does not provide additional equations to include into the system.

We can obtain $\binom{n+t}{t}$ structures by selecting m^0 blocks at exactly t positions of the super structure, and keeping the choice among two values on the other n positions. These structures lead to linearly independent equations since the message containing n blocks equal to m^1 at given positions only appear in one specific structure.

Increasing the Number of Structures Extracted from a Super Structure. More generally, we can obtain $S(n+t, t)$ structures by selecting m^0 blocks at any given $0 \leq i \leq t$ positions of the super structure, and keeping the choice among two values on the other n positions. Indeed, let us associate to every structure an integer, whose binary representation represents the indeterminate positions of the structure. Let us also associate to every message an integer whose binary representation is the selection pattern of m^0 and m^1 blocks. Ordering the structures by decreasing value of their representative and the messages by decreasing values of their associated integers, the binary structure/message membership matrix $(\delta_{M_j \in \mathcal{S}_i})_{i,j}$ is in row echelon form (see Figure 35), which proves the linear independence of equations generated from these structures.

Finally, we remark that the $(n+t-i)$-dimensional structures generated above can be generated linearly from the n-dimensional structures where the fixed message blocks are selected from $\{m^0, m^1\}$. It it thus possible to select $S(n+t, t)$ such n-dimensional structures leading to independent equations.

Computing Equations from Super Structures. There are at least two possible strategies to build the equation system using super structures. First, we can store all the data blocks that we get, and compute the system equation by equation, recomputing the contribution from each block of a given structure. Otherwise, we can handle the data block per block, building all the system of equations at once.

The first strategy requires to store all the data blocks during the computation of the system, whereas the second approach has no specific memory requirements and potentially allows to spare computation time. Nevertheless, extra memory can be required for the construction of the system of equations itself. This will be studied in Section 5.4.3. In the following, we assume that we select the second approach, and compute all equations simultaneously, handling the available data block by block.

Complexity Analysis. The major benefit of using super structures of input messages is to reduce the data complexity of the attack. By considering super structures of size $n+t$ and ℓ_o output blocks per message, the data complexity is $(n+t+\ell_o)2^{n+t}$ blocks. The number of equations we get is $b\ell_o S(n+t, t)$, assuming one does not use the optimization from Section 5.4.1 and computes b equations per structure. Otherwise, only one equation is recovered per structure, and the total number of equations is $\ell_o S(n+t, t)$.

We select the parameters of our attack as follows. One aims at getting N_{eq} equations, therefore we need that $b\ell_o S(n+t, t) \geq N_{eq}$ (resp. $\ell_o S(n+t, t) \geq N_{eq}$) if we do not reduce (resp. we reduce) the number of equations. Thus,

we choose $\ell_o = \lceil N_{eq}/(S(N+t,t)b) \rceil$ (resp. $\ell_o = \lceil N_{eq}/S(N+t,t) \rceil$. As our aim is to reduce the amount of data necessary for the attack, we then need to choose t so as to minimize $(n+t+\ell_o)2^{n+t}$.

If we combine these super structures with the reduction of the number of equations, we need $N_{eq} \approx 2^{29.0}$. This leads to $t = 5$, $\ell_o = 4$ and a data complexity of $2^{74.7}$ blocks. If we use only super structures, we need $N_{eq} \approx 2^{36.5}$ equations. We then compute $t = 4$, $\ell_o = 5$ and reach a data complexity of $2^{73.9}$ blocks.

	1111	1110	1101	1100	1011	1010	1001	1000	0111	0110	0101	0100	0011	...
****	1	1	1	1	1	1	1	1	1	1	1	1	1	...
***0		1		1		1		1		1		1		...
**0*			1	1			1	1			1	1		...
**00				1				1				1		...
*0**					1	1	1	1						...
*0*0						1		1						...
00							1	1						...
0***									1	1	1	1	1	...
0**0										1		1		...
0*0*											1	1		...
00**													1	...

Figure 35: Example of structure/message membership matrix, with $(n,t) = (2,2)$.

5.4.3 Counters

We specify here two algorithms that the adversary might use to build a system of equations from the outputs, and study their time and memory complexities, in order to determine the optimal choice in each version of our attack.

Description of the Systems of Equations. The attacks we want to optimize are either based on the higher order differential property or on the use of a polynomial derived from a stream-oriented description of the expansion layer of KRAVATTE. Each equation of the system is derived from a linear relation on bit values two rounds before the output. It is obtained by summing the contributions of several output blocks. The expression of such a contribution requires the computation of the polynomial expression

of one bit (or one linear combination of bits) two rounds before the end of KRAVATTE, considering key bits as variables.

In the following, we consider that the bottleneck of such an operation consists in the multiplications of three polynomials that stems from the degree-3 term of the internal χ^{-1} layer. The complexity of such an operation is estimated as the product of the number of terms of each of the three polynomials.

Definitions and Notations. We call *equation* a relation involving key bits and newly introduced key-dependent variables. The equations we use are computed by summing polynomials that depend on one output block. We use the generic term *expression* to refer to such a polynomial, which is computed by inverting two rounds of KRAVATTE. The addition of a given expression when building an equation is called a *contribution*.

We use N_{eq} to denote the total number of equations that is needed to solve the system, which is supposed to be equal to the number of variables. Therefore, we have $N_{eq} = 2^{29.0}$ if optimization of Section 5.4.1 is implemented, and $2^{36.5}$ otherwise. We denote by S the number of contributions that one needs to add to get each equation. This number depends on the kind of distinguisher that is used. For the 6-round higher order differential distinguisher, we have $S = 2^{65}$. If we use the linear recurrence distinguisher, we need to sum expressions over all the block positions given by the nonzero coefficients of the recurrence polynomial. We estimate it as half the degree of the polynomial (see Table 11). Conversely, we call R the average number of equations each computed expression contributes to. When handling a specific output block, one computes the polynomial expression of bits two rounds before the output, then add its contribution to the R equations (on average) it is involved in. We also denote by N_{par} the maximum number of equations that are being computed at the same time during the process. When using the linear recurrence distinguisher or the higher order differential distinguisher with super structures, we compute all equations in parallel, and $N_{par} = N_{eq}$, whereas for the higher order differential distinguisher without super structures, we build 1 or 1600 equations in parallel, depending on the number of bits of information we use per block. We also denote by N_{prod} the number of products of three monomials that one needs to compute when inverting the internal χ^{-1} layer. In the general case, N_{prod} is dominated by the product of three polynomials in 8000 variables, which can be considered to cost $8001^3 \approx 2^{38.9}$ elementary operations. If we apply optimizations of Section 5.4.1, the most expensive part consists in 6 multiplications of 3 polynomials: 1 with 8 nonzero coefficients and 2 with 4801 nonzero coefficients. We then have $N_{prod} = 6 \times 8 \times 4801^2 \approx 2^{30.0}$ elementary operations.

A Direct Approach. The most straightforward way to proceed is to run through all the output blocks that are needed to get enough equations: For each of them, we compute each expression that is needed to build the system, and add it to all equations it contributes to.

Using this technique does not require a specific amount of memory, other than the one used to store the linearized system. We assume that all expressions originating from a single block of output are computed independently. The time complexity to compute each expression in key bits that contributes to the system of equations is N_{prod} operations. The total number of contributions is $N_{eq} \times S$, and therefore the total number of expressions one needs to compute is $N_{eq} \times S/R$. After computing these expressions, one needs to add them to the equations they contribute to. The total number such additions is $N_{eq} \times S$. As the number of equations equals the number of variables, such an addition costs N_{eq} elementary operations. Therefore, the time complexity T_{direct} of this technique is given by

$$T_{direct} = N_{eq} \times S \times \left(N_{prod}/R + N_{eq}\right).$$

An Optimization Based on the Use of Parity Counters. Our second technique relies on the following observation. Each of the polynomials that are multiplied during the internal χ^{-1} layer are linear combinations of output bits of the external χ^{-1} layers. Such a bit only depends on the five bits of the output block corresponding to one Sbox output. Moreover, when expanding the products of the internal χ^{-1} layer, one gets the sum of products of three such bits, which only depends on 15 bits of the output blocks, corresponding to the outputs of three Sboxes.

For each equation, the computation of all the S contributions leads to computing several times these products, as soon as S exceeds 2^{15}. Our idea is then to compute these products only once for each value V of the 15 output bits. This can be done in a precomputation step. Then, the contribution of each of these results is added to the final equation if and only if the number of occurrences of V is odd (as the sum of an even number of identical values cancels out in characteristic two). This can be achieved as follows: For each output block, one runs through all useful sets of three Sboxes, and update a parity counter for the 15 output bits of these Sboxes, for each equation the current output block contributes to. Each parity counter consists of 2^{15} parity bits, which are used to store the parity of the number of occurrences of all values of the 15 output bits of the set of three Sboxes defining the counter. Then, one adds to each equation the contribution corresponding of each possible value of each counter.

We denote by N_{ctr} the number of counters that are used for each equation. In the general case, one needs a parity counter set for each possible combination of three output Sboxes, which makes $N_{ctr} = \binom{320}{3} \approx 2^{22.4}$. When the number of equations is optimized, we consider the multiplications of six bits after the external χ^{-1} layer with two dense polynomials. Therefore, one only needs at most $N_{ctr} = 6 \times \binom{320}{2} \approx 2^{18.2}$.

The memory complexity of this step is the amount of memory that is needed to store all the parity bits, which is $M_{ctr} = N_{par} \times N_{ctr} \times 2^{15}$ bits. The time required for the attack encompasses the updates of parity counters and the addition of contributions of individual counter values to all the equations. The value of each counter only contributes to the coefficients of monomials in key bits at the 15 same positions. Therefore, adding the contribution of such a counter requires at most 2^{15} key additions. For each set of three Sboxes, there are 2^{15} counter values to consider. Therefore, we have: $T_{ctr} = N_{eq} \times N_{ctr} \times (S + 2^{30})$.

Comparison Between the Two Techniques. From the formulae above, we always have $T_{direct} > N_{eq}^2 \times S$. If T_{ctr} is smaller than this value, the time complexity of the second algorithm is better. This is equivalent to $N_{ctr} \times (S + 2^{30}) < N_{eq} \times S$. Moreover, the number of possible counters is bounded by $\binom{320}{3} \approx 2^{22.4}$, whereas the number of equations is at least $2^{29.0}$ when optimization of Section 5.4.1 is implemented. Therefore, a sufficient condition for the second algorithm to be more efficient becomes $S + 2^{30} \leq 2^{29.0-22.4}$, which is equivalent to $S \geq 2^{30-6.6} = 2^{23.4}$. In our attacks, S is the number of elements of an affine space of a higher order differential distinguisher or the number of nonzero coefficients of a recurrence polynomial. In all cases we focus on, it is larger than this bound.

The parity counter based attack should therefore be used in any case, unless one aims at optimizing the memory required by the attack and the storage of counter values is its bottleneck in terms of memory complexity.

5.4.4 Optimizing the Attacks

The high number of different attacks and potential combinations of optimizations makes it difficult to give an exhaustive list of all the possible combinations and their complexities. However, we give numerical applications for a few of them. These complexities are summarized in Table 9. We explain here how we obtain the optimized complexities.

Linear-Recurrence Attack on KRAVATTE-$(\star, 4)$ with $n_e = 2 + 2$. We use both optimizations of Section 5.4.1 and Section 5.4.3. The precomputation step, as shown in Section 5.2.2, has a time complexity of $T_P = 2^{40.7}$ elemen-

tary operations and a memory complexity of $2^{28.7}$ bits to store the recurrence polynomial. As we reduce the number of variables, we only get one expression per block. The data complexity is then changed to $2^{28.7} + 2^{29.0} \approx 2^{29.9}$ blocks (from $2^{28.7} + \frac{1}{1600}2^{36.5}$). The time complexity to build the system is about $2^{77.5}$ operations, and the memory complexity to store the counters is $2^{62.2}$ bits. Finally, solving the system requires $\left(2^{29}\right)^3 = 2^{87}$ operations, and storing it about 2^{58} bits. Overall, the time complexity of the attack is 2^{87} basic operations, the memory complexity is $2^{62.3}$ bits and the data complexity is $2^{29.9}$ blocks.

Linear-Recurrence Attack on Kravatte-$(\star, 6)$ **with** $n_e = 4 + 2$**.** We use the same optimizations for the attack on $n_e = 6$ rounds. The memory required for the attack is now mainly due to the storage of the recurrence polynomial, which requires $2^{88.4}$ bits. As the data complexity mainly comes from the high degree of this polynomial, it is still $2^{88.4}$ blocks as in Section 5.2.2. Finally, the time complexity of this attack is dominated by the construction of the system, which amounts to $2^{134.6}$ elementary operations.

Higher Order Differential Attack on Kravatte-$(4, 4)$ **Using All Optimizations.** We now focus on attacks based on the higher order differential distinguisher, and first try to apply all three optimizations. As shown in Section 5.4.2, the data complexity is $2^{74.7}$ blocks. During the construction phase, the memory required for the parity counters is again $2^{62.2}$ bits, and its time complexity is $2^{112.2}$ operations, which is the most time-consuming part of the attack. The memory and time complexities of the system resolution are the same as for the linear-recurrence attack, increasing the memory complexity to $2^{62.3}$ bits.

Memory-optimized Higher Order Differential Attack on Kravatte-$(4, 4)$**.** As the memory required mainly comes from the storage of parity counters, we can drop the optimization based on super structures. The data complexity goes up to $2^{65} \times (65 + 2^{29}) \approx 2^{94}$ blocks, and the memory complexity drops to the 2^{58} bits required to store the system. The time complexity is left unchanged.

Data-optimized Higher Order Differential Attack on Kravatte-$(4, 4)$**.** Similarly, to minimize the amount of data needed for the attack, we can drop the optimization of Section 5.4.1. As shown in Section 5.4.2, the data complexity decreases to $2^{73.9}$ blocks, but the number of equations required increases to $2^{36.5}$. The time and memory complexities of the construction step are respectively increased to $2^{123.9}$ operations and $2^{73.9}$ bits. Adding the storage of the system (2^{73} bits), the memory complexity of the attack becomes $2^{74.5}$ bits.

5.5 Concluding Remarks and Discussion

We depicted in this chapter several key-recovery attack strategies breaking the security claims of the recent PRF proposal Kravatte. The attacks are primarily focused on either the convergence point or the divergence point of the high-level structure that allows to compress virtually any number of blocks to a single one in an incremental way, and conversely, to expand a single block to almost any number of output blocks. The properties of these two sensitive points of the computation, where all the input information is packed into a single block (right after the compressing phase and right before the second step of the expansion phase), together with the low algebraic degree of the Keccak-p permutation, are leveraged in our attacks. From this ambitious and aggressive design structure and the proposed attacks, we would like to draw some high-level conclusions.

First, the non-linear permutations p_c in the compression layer do not prevent the construction of an affine space at its output. This is inherent to the design and cannot be thwarted by simply increasing the number of rounds in p_c. Secondly, the middle non-linear permutation p_d applied to the accumulator does not increase the security of the expansion layer, as one can target the second step of the expansion layer, i.e., applications of p_e to the evolving rolling state, independently. The design incentive was probably to factor out a part of the non-linear transformations of the expansion layer to increase the performances of the output generations, but it appears that such an optimization strongly decreases the security. Finally, the last non-linear permutations p_e used to produce each output block in Kravatte have low algebraic degree and are applied after a small linear diversification mechanism. This results in a bit mixing much simpler than expected, which can be distinguished before the end of the expansion layer and used to recover the key by inverting the remaining part.

We note that one can interpret all ours attacks in terms of stream cipher analysis, with either attacks on the IV-processing part (the compression layer) or on the keystream generation part (the expansion layer). Recasting Kravatte in this light may help to improve its design.

BIBLIOGRAPHY

[AA16] Ralph Ankele and Robin Ankele. Software benchmarking of the 2nd round caesar candidates. Cryptology ePrint Archive, Report 2016/740, 2016. https://eprint.iacr.org/2016/740.

[ABD+] Elena Andreeva, Andrey Bogdanov, Nilanjan Datta, Atul Luykx, Bart Mennink, Mridul Nandi, Elmar Tischhauser, and Kan Yasuda. Security of COLM. https://competitions.cr.yp.to/round3/colm-addendum.pdf.

[ABD+16a] Elena Andreeva, Andrey Bogdanov, Nilanjan Datta, Atul Luykx, Bart Mennink, Mridul Nandi, Elmar Tischhauser, and Kan Yasuda. AES-COPA v.2. Submission to the CAESAR Competition, 2016.

[ABD+16b] Elena Andreeva, Andrey Bogdanov, Nilanjan Datta, Atul Luykx, Bart Mennink, Mridul Nandi, Elmar Tischhauser, and Kan Yasuda. COLM v1. Submission to the CAESAR Competition, 2016.

[ABL+14] Elena Andreeva, Andrey Bogdanov, Atul Luykx, Bart Mennink, Nicky Mouha, and Kan Yasuda. How to securely release unverified plaintext in authenticated encryption. In Sarkar and Iwata [SI14], pages 105–125.

[ADMA15] Elena Andreeva, Joan Daemen, Bart Mennink, and Gilles Van Assche. Security of keyed sponge constructions using a modular proof approach. In *Fast Software Encryption - 22nd International Workshop, FSE 2015, Istanbul, Turkey, March 8-11, 2015, Revised Selected Papers*, pages 364–384, 2015.

[AEL+18] Tomer Ashur, Maria Eichlseder, Martin M. Lauridsen, Gaëtan Leurent, Brice Minaud, Yann Rotella, Yu Sasaki, and Benoît Viguier. Cryptanalysis of MORUS. Cryptology ePrint Archive, Report 2018/464, 2018. https://eprint.iacr.org/2018/464.

[AES01] Advanced Encryption Standard, National Institute of Standards and Technology (NIST), FIPS PUB 197, U.S. Department Of Commerce. https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.197.pdf, 2001.

[AHMP10] Jean-Philippe Aumasson, Luca Henzen, Willi Meier, and Raphael C.-W. Phan. SHA-3 proposal BLAKE. Submission to NIST (Round 3), 2010.

[AJN15a] Jean-Philippe Aumasson, Philipp Jovanovic, and Samuel Neves. Analysis of NORX: Investigating differential and rotational properties. In Diego F. Aranha and Alfred Menezes, editors, *LATINCRYPT 2014*, volume 8895 of *LNCS*, pages 306–324. Springer, Heidelberg, September 2015.

[AJN15b] Jean-Philippe Aumasson, Philipp Jovanovic, and Samuel Neves. NORX v2.0. Submission to the CAESAR Competition, 2015.

[AJN15c] Jean-Philippe Aumasson, Philipp Jovanovic, and Samuel Neves. NORX8 and NORX16: Authenticated Encryption for Low-End Systems. Cryptology ePrint Archive, Report 2015/1154, 2015.

[ANS14] ANSSI. Référentiel Général de Sécurité version 2.0, 2014.

[ANWW13] Jean-Philippe Aumasson, Samuel Neves, Zooko Wilcox-O'Hearn, and Christian Winnerlein. BLAKE2: Simpler, smaller, fast as MD5. In Michael J. Jacobson Jr., Michael E. Locasto, Payman Mohassel, and Reihaneh Safavi-Naini, editors, *ACNS 13*, volume 7954 of *LNCS*, pages 119–135. Springer, Heidelberg, June 2013.

[Bab95] Steve Babbage. Improved "Exhaustive Search" Attacks on Stream Ciphers. In *European Convention on Security and Detection, no. 408 in IEE Conference Publication*, pages 161–166. IET, 1995.

[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 Guido Bertoni and Jean-Sébastien Coron, editors, *CHES 2013*, volume 8086 of *LNCS*, pages 142–158. Springer, Heidelberg, August 2013.

[BCK96] Mihir Bellare, Ran Canetti, and Hugo Krawczyk. Keying hash functions for message authentication, 1996.

[BDD+17] Xavier Bonnetain, Patrick Derbez, Sébastien Duval, Jérémy Jean, Gaëtan Leurent, Brice Minaud, and Valentin Suder. AEZ forgeries. Post by G. Leurent to the "Cryptographic competitions" Google group on March 12, 2017. Available at http://goo.gl/3P4K51., 2017.

[BDH+16] Guido Bertoni, Joan Daemen, Seth Hoffert, Michaël Peeters, Gilles Van Assche, and Ronny Van Keer. Farfalle: parallel permutation-based cryptography. Cryptology ePrint Archive, Report 2016/1188, 2016.

[BDH+17a] Guido Bertoni, Joan Daemen, Seth Hoffert, Michaël Peeters, Gilles Van Assche, and Ronny Van Keer. Farfalle: Parallel Permutation-Based Cryptography. *IACR Transactions on Symmetric Cryptology*, 2017(4), 2017.

[BDH+17b] Guido Bertoni, Joan Daemen, Seth Hoffert, Michaël Peeters, Gilles Van Assche, and Ronny Van Keer. Innovations in Permutation-Based Crypto. Slides from ECC 2017, 2017.

[BDJR97] Mihir Bellare, Anand Desai, Eron Jokipii, and Phillip Rogaway. A concrete security treatment of symmetric encryption, 1997.

[BDPA11a] G. Bertoni, J. Daemen, M. Peeters, and G. Van Assche. The Keccak reference. Round 3 submission to NIST SHA-3, 2011.

[BDPA11b] Guido Bertoni, Joan Daemen, Michaël Peeters, and Gilles Van Assche. The keccak reference, 2011.

[BDPV12] Guido Bertoni, Joan Daemen, Michael Peeters, and Gilles Van Assche. Duplexing the sponge: Single-pass authenticated encryption and other applications. In Ali Miri and Serge Vaudenay, editors, *SAC 2011*, volume 7118 of *LNCS*, pages 320–337. Springer, Heidelberg, August 2012.

[BDPVA11] Guido Bertoni, Joan Daemen, Michaël Peeters, and Gilles Van Assche. Duplexing the sponge: single-pass authenticated encryption and other applications. In *International Workshop on Selected Areas in Cryptography*, pages 320–337. Springer, 2011.

[Ber08] Daniel J. Bernstein. ChaCha, a variant of Salsa20, 2008.

[BGTZ08] Richard P. Brent, Pierrick Gaudry, Emmanuel Thomé, and Paul Zimmermann. Faster Multiplication in GF(2)[x]. In *Algorithmic Number Theory, 8th International Symposium, ANTS-VIII, Banff, Canada, May 17-22, 2008, Proceedings*, pages 153–166, 2008.

[BHJ+16] Nasour Bagheri, Tao Huang, Keting Jia, Florian Mendel, and Yu Sasaki. Cryptanalysis of reduced NORX. In Thomas Peyrin, editor, *FSE 2016*, volume 9783 of *LNCS*, pages 554–574. Springer, Heidelberg, March 2016.

[BN00] Mihir Bellare and Chanathip Namprempre. Authenticated Encryption: Relations among notions and analysis of the generic composition paradigm. Cryptology ePrint Archive, Report 2000/025, 2000. https://eprint.iacr.org/2000/025.

[BR00] John Black and Phillip Rogaway. CBC MACs for arbitrary-length messages: The three-key constructions. In *Annual International Cryptology Conference*, pages 197–215. Springer, 2000.

[BS91] Eli Biham and Adi Shamir. Differential cryptanalysis of DES-like cryptosystems. *Journal of Cryptology*, 1991.

[CFG15] Colin Chaigneau, Thomas Fuhr, and Henri Gilbert. Full key-recovery on ACORN in nonce-reuse and decryption-misuse settings, 2015. https://groups.google.com/d/msg/crypto-competitions/RTtZvFZay7k/-_nVcA7EadUJ.

[CFG+17] Colin Chaigneau, Thomas Fuhr, Henri Gilbert, Jérémy Jean, and Jean-René Reinhard. Cryptanalysis of NORX v2.0. *IACR Transactions on Symmetric Cryptology*, 2017(1):156–174, Mar. 2017.

[CFG+18a] Colin Chaigneau, Thomas Fuhr, Henri Gilbert, Jian Guo, Jérémy Jean, Jean-René Reinhard, and Ling Song. Key-Recovery Attacks on Full Kravatte. *IACR Transactions on Symmetric Cryptology*, 2018(1):5–28, Mar. 2018.

[CFG+18b] Colin Chaigneau, Thomas Fuhr, Henri Gilbert, Jérémy Jean, and Jean-René Reinhard. Cryptanalysis of NORX v2.0, 2018. To appear, https://www.springerprofessional.de/en/cryptanalysis-of-norx-v2-0/15826164.

[CG16] Colin Chaigneau and Henri Gilbert. Is AEZ v4.1 Sufficiently Resilient Against Key-Recovery Attacks? *IACR Transactions on Symmetric Cryptology*, 2016(1):114–133, Dec. 2016.

[CHP+17] Carlos Cid, Tao Huang, Thomas Peyrin, Yu Sasaki, and Ling Song. Cryptanalysis of Deoxys and its Internal Tweakable Block Ciphers. Cryptology ePrint Archive, Report 2017/693, 2017. https://eprint.iacr.org/2017/693.

[Cla45] Claude Shannon. A Mathematical Theory of Cryptography, 1945.

[Cou03] Nicolas T. Courtois. Fast Algebraic Attacks on Stream Ciphers with Linear Feedback, 2003.

[CSA16] ECRYPT – CSA. D5.2 Algorithms, Key Size and Protocols Report, 2016.

[DD07] Morris J Dworkin and MJ Dworkin. Recommendation for block cipher modes of operation: The CCM mode for authentication and confidentiality, 2007.

[DEMS] Christoph Dobraunig, Maria Eichlseder, Florian Mendel, and Martin Schläffer. Ascon v1.2.

[DEMS15] Christoph Dobraunig, Maria Eichlseder, Florian Mendel, and Martin Schläffer. Cryptanalysis of Ascon. Cryptology ePrint Archive, Report 2015/030, 2015. https://eprint.iacr.org/2015/030.

[DES77] Data Encryption Standard, National Bureau of Standards, NBS FIPS PUB 46, U.S. Department Of Commerce, january 1977.

[DJ15] Itai Dinur and Jérémy Jean. Cryptanalysis of FIDES. In Carlos Cid and Christian Rechberger, editors, *FSE 2014*, volume 8540 of *LNCS*, pages 224–240. Springer, Heidelberg, March 2015.

[DKM+16] Ashutosh Dhar Dwivedi, Miloš Klouček, Pawel Morawiecki, Ivica Nikolić, Josef Pieprzyk, and Sebastian Wójtowicz. SAT-based Cryptanalysis of Authenticated Ciphers from the CAESAR Competition. Cryptology ePrint Archive, Report 2016/1053, 2016.

[DKR97] Joan Daemen, Lars Knudsen, and Vincent Rijmen. The block cipher Square. In *Fast Software Encryption*, pages 149–165, Berlin, Heidelberg, 1997. Springer Berlin Heidelberg.

[DLMW15] Itai Dinur, Yunwen Liu, Willi Meier, and Qingju Wang. Optimized interpolation attacks on LowMC. In Tetsu Iwata and Jung Hee Cheon, editors, *ASIACRYPT 2015, Part II*, volume 9453 of *LNCS*, pages 535–560. Springer, Heidelberg, November / December 2015.

[DMA17] Joan Daemen, Bart Mennink, and Gilles Van Assche. Full-state keyed duplex with built-in multi-user support. In *Advances in Cryptology - ASIACRYPT 2017 - 23rd International Conference on the Theory and Applications of Cryptology and Information Security, Hong Kong, China, December 3-7, 2017, Proceedings, Part II*, pages 606–637, 2017.

[DMM15] Sourav Das, Subhamoy Maitra, and Willi Meier. Higher Order Differential Analysis of NORX. Cryptology ePrint Archive, Report 2015/186, 2015.

[DN16] Nilanjan Datta and Mridul Nandi. Proposal of ELmD v2.1. Submission to the CAESAR Competition, 2016.

[Dwo01] Morris Dworkin. Recommendation for block cipher modes of operation. methods and techniques. Technical report, NATIONAL INST OF STANDARDS AND TECHNOLOGY GAITHERSBURG MD COMPUTER SECURITY DIV, 2001.

[Fer02] Niels Ferguson. Collision Attacks on OCB, 2002.

[FLS15] Thomas Fuhr, Gaëtan Leurent, and Valentin Suder. Collision Attacks Against CAESAR Candidates - Forgery and Key-Recovery Against AEZ and Marble. In *ASIACRYPT (2)*, volume 9453 of *Lecture Notes in Computer Science*, pages 510–532. Springer, 2015.

[FMS01] Scott Fluhrer, Itsik Mantin, and Adi Shamir. Weaknesses in the key scheduling algorithm of RC4. In *International Workshop on Selected Areas in Cryptography*, pages 1–24. Springer, 2001.

[Gol97] Jovan Dj. Golić. Cryptanalysis of Alleged A5 Stream Cipher. In Walter Fumy, editor, *Advances in Cryptology - EUROCRYPT '97, International Conference on the Theory and Application of Cryptographic Techniques, Konstanz, Germany, May 11-15, 1997, Proceeding*, volume 1233 of *Lecture Notes in Computer Science*, pages 239–255. Springer, 1997.

[GRW16] Faruk Göloğlu, Vincent Rijmen, and Qingju Wang. On the division property of S-boxes. Cryptology ePrint Archive, Report 2016/188, 2016. https://eprint.iacr.org/2016/188.

[GWDE15] Hannes Groß, Erich Wenger, Christoph Dobraunig, and Christoph Ehrenhöfer. Suit up! Made-to-Measure Hardware Implementations of Ascon. Cryptology ePrint Archive, Report 2015/034, 2015. https://eprint.iacr.org/2015/034.

[HKR15a] Viet Tung Hoang, Ted Krovetz, and Phillip Rogaway. AEZ v4.1: Authenticated Encryption by Enciphering. http://web.cs.ucdavis.edu/~rogaway/aez/aez.pdf, 2015.

[HKR15b] Viet Tung Hoang, Ted Krovetz, and Phillip Rogaway. Robust Authenticated-Encryption AEZ and the Problem That It Solves. In *EUROCRYPT (1)*, volume 9056 of *Lecture Notes in Computer Science*, pages 15–44. Springer, 2015.

[HRRV15] Viet Tung Hoang, Reza Reyhanitabar, Phillip Rogaway, and Damian Vizár. Online Authenticated-Encryption and its Nonce-Reuse Misuse-Resistance. In *Advances in Cryptology - CRYPTO 2015 - 35th Annual Cryptology Conference, Santa Barbara, CA, USA, August 16-20, 2015, Proceedings, Part I*, pages 493–517, 2015.

[HW16] Tao Huang Hongjun Wu. The Authenticated Cipher MORUS (v2). Submission to the CAESAR Competition, 2016.

[JJ́] Jérémy Jean. TikZ for Cryptographers. http://www.iacr.org/authors/tikz/.

[JK97] Thomas Jakobsen and Lars R. Knudsen. The interpolation attack on block ciphers. In Eli Biham, editor, *FSE'97*, volume 1267 of *LNCS*, pages 28–40. Springer, Heidelberg, January 1997.

[JLM14] Philipp Jovanovic, Atul Luykx, and Bart Mennink. Beyond $2^{c/2}$ security in sponge-based authenticated encryption modes. In Sarkar and Iwata [SI14], pages 85–104.

[JNPS16] Jérémy Jean, Ivica Nikolić, Thomas Peyrin, and Yannick Seurin. Deoxys v1.41. Submission to the CAESAR Competition, 2016.

[Jou06] Antoine Joux. Authentication failures in NIST version of GCM, 2006. https://csrc.nist.gov/csrc/media/projects/block-cipher-techniques/documents/bcm/comments/800-38-series-drafts/gcm/joux_comments.pdf.

[JS15] Rebhu Johymalyo Josh and Santanu Sarkar. Some observations on ACORN v1 and Trivia-SC. In *Lightweight Cryptography Workshop, NIST, USA*, pages 20–21, 2015.

[Key76] E.L. Key. An Analysis of the Structure and Complexity of Nonlinear Binary Sequence Generators. *IEEE Transactions on Information Theory*, 22(6):732–736, 1976.

[KN10] Dmitry Khovratovich and Ivica Nikolić. Rotational cryptanalysis of ARX. In Seokhie Hong and Tetsu Iwata, editors, *FSE 2010*, volume 6147 of *LNCS*, pages 333–346. Springer, Heidelberg, February 2010.

[Lai94] Xuejia Lai. Higher Order Derivatives And Differential Cryptanalysis. *Kluwer International Series In Engineering And Computer Science*, pages 227–227, 1994.

[LDW17] Zheng Li, Xiaoyang Dong, and Xiaoyun Wang. Conditional Cube Attack on Round-Reduced ASCON. Cryptology ePrint Archive, Report 2017/160, 2017. https://eprint.iacr.org/2017/160.

[LL14] Meicheng Liu and Dongdai Lin. Cryptanalysis of Lightweight Authenticated Cipher ACORN, 2014. https://groups.google.com/d/msg/crypto-competitions/2mrDnyb9hfM/tjlpmfSZ0TcJ.

[LLMH16] Frédéric Lafitte, Liran Lerman, Olivier Markowitch, and Dirk Van Heule. SAT-based cryptanalysis of ACORN. Cryptology ePrint Archive, Report 2016/521, 2016. https://eprint.iacr.org/2016/521.

[LMR15] Gregor Leander, Brice Minaud, and Sondre Rønjom. A generic approach to invariant subspace attacks: Cryptanalysis of robin, iSCREAM and Zorro. In Elisabeth Oswald and Marc Fischlin, editors, *EUROCRYPT 2015, Part I*, volume 9056 of *LNCS*, pages 254–283. Springer, Heidelberg, April 2015.

[LRW11] Moses Liskov, Ronald L. Rivest, and David Wagner. Tweakable Block Ciphers. *J. Cryptology*, 24(3):588–613, 2011.

[Min14] Brice Minaud. Linear biases in AEGIS keystream. In *International Workshop on Selected Areas in Cryptography*, pages 290–305. Springer, 2014.

[mMS18] Alireza mehrdad, Farokhlagha Moazami, and Hadi Soleimany. Impossible Differential Cryptanalysis on Deoxys-BC-256. Cryptology ePrint Archive, Report 2018/048, 2018. https://eprint.iacr.org/2018/048.

[MV04a] David McGrew and John Viega. The Galois/Counter Mode of Operation (GCM). *Submission to NIST.*, 2004.

[MV04b] David A. McGrew and John Viega. The Security and Performance of the Galois/Counter Mode (GCM) of Operation. In *INDOCRYPT*, volume 3348 of *Lecture Notes in Computer Science*, pages 343–355. Springer, 2004.

[Nai16] Yusuke Naito. Sandwich Construction for Keyed Sponges: Independence Between Capacity and Online Queries. In Sara Foresti and Giuseppe Persiano, editors, *Cryptology and Network Security - 15th International Conference, CANS 2016, Milan, Italy, November 14-16, 2016, Proceedings*, volume 10052 of *Lecture Notes in Computer Science*, pages 245–261, 2016.

[NIS14] NIST Computer Security Division. SHA-3 Standard: Permutation-Based Hash and Extendable-Output Functions. FIPS Publication 202, National Institute of Standards and Technology, U.S. Department of Commerce, May 2014.

[PS15] Thomas Peyrin and Yannick Seurin. Counter-in-Tweak: Authenticated Encryption Modes for Tweakable Block Ciphers. Cryptology ePrint Archive, Report 2015/1049, 2015. https://eprint.iacr.org/2015/1049.

[PvO95] Bart Preneel and Paul C. van Oorschot. MDx-MAC and building fast MACs from hash functions. In Don Coppersmith, editor, *CRYPTO'95*, volume 963 of *LNCS*, pages 1–14. Springer, Heidelberg, August 1995.

[RBBK01] Phillip Rogaway, Mihir Bellare, John Black, and Ted Krovetz. OCB: a block-cipher mode of operation for efficient authenticated encryption. In *ACM Conference on Computer and Communications Security*, pages 196–205. ACM, 2001.

[RGH07] S. Rønjom, G. Gong, and T. Helleseth. On Attacks on Filtering Generators Using Linear Subspace Structures. In *SSC*, pages 204–217, 2007.

[RH07] S. Rønjom and T. Helleseth. A New Attack on the Filter Generator. *IEEE Transactions on Information Theory*, 53(5):1752–1758, 2007.

[Rog04] Phillip Rogaway. Efficient instantiations of tweakable blockciphers and refinements to modes ocb and pmac, 2004.

[RS06] Phillip Rogaway and Thomas Shrimpton. Deterministic Authenticated-Encryption: A Provable-Security Treatment of the Key-Wrap Problem. Cryptology ePrint Archive, Report 2006/221, 2006. https://eprint.iacr.org/2006/221.

[Sch77] Arnold Schönhage. Schnelle Multiplikation von Polynomen über Körpern der Charakteristik 2. *Acta Inf.*, 7:395–398, 1977.

[SI14] Palash Sarkar and Tetsu Iwata, editors. *ASIACRYPT 2014, Part I*, volume 8873 of *LNCS*. Springer, Heidelberg, December 2014.

[SWB+15] Md Iftekhar Salam, Kenneth Koon-Ho Wong, Harry Bartlett, Leonie Simpson, Ed Dawson, and Josef Pieprzyk. Finding State Collisions in the Authenticated Encryption Stream Cipher ACORN. Cryptology ePrint Archive, Report 2015/918, 2015. https://eprint.iacr.org/2015/918.

[Tez16] Cihangir Tezcan. Truncated, Impossible, and Improbable Differential Analysis of Ascon. Cryptology ePrint Archive, Report 2016/490, 2016. https://eprint.iacr.org/2016/490.

[TG91] Anne Tardy-Corfdir and Henri Gilbert. A known plaintext attack of FEAL-4 and FEAL-6. In *Advances in Cryptology - CRYPTO '91, 11th Annual International Cryptology Conference, Santa Barbara, California, USA, August 11-15, 1991, Proceedings*, pages 172–181, 1991.

[WP16] Hongjun Wu and Bart Preneel. AEGIS: A Fast Authenticated Encryption Algorithm (v1.1). Submission to the CAESAR Competition, 2016.

[Wu16] Hongjun Wu. ACORN: A Lightweight Authenticated Cipher (v3), howpublished = Submission to the CAESAR Competition, 2016.

[ZDW18] Rui Zong, Xiaoyang Dong, and Xiaoyun Wang. Related-Tweakey Impossible Differential Attack on Reduced-Round

Deoxys-BC-256. Cryptology ePrint Archive, Report 2018/680, 2018. https://eprint.iacr.org/2018/680.