Design and Analysis of Symmetric Primitives

Lauridsen, Martin Mehl; Rechberger, Christian; Knudsen, Lars Ramkilde

Publication date:
2016

Document Version
Publisher's PDF, also known as Version of record

Link back to DTU Orbit

Citation (APA):
Lauridsen, M. M., Rechberger, C., & Knudsen, L. R. (2016). Design and Analysis of Symmetric Primitives. Kgs. Lyngby: Technical University of Denmark (DTU). (DTU Compute PHD-2015; No. 382).

Design and Analysis of Symmetric Primitives

Martin M. Lauridsen

August 2015

Advisor:	Christian Rechberger
Co-advisor:	Lars R. Knudsen

Technical University of Denmark
Department of Applied Mathematics and Computer Science

ISSN:	0909-3192
Serial no.:	PHD-2015-382

Til mine forældre

Abstract

The subject of this thesis is the study of symmetric cryptographic primitives. We investigate these objects from three different perspectives: cryptanalysis, design and implementation aspects.

The first part deals with cryptanalysis of symmetric primitives, where one tries to leverage a property of the design to achieve some adversarial goal. Two of the most successful types of cryptanalysis are differential- and linear attacks. We apply variants of differential cryptanalysis to the lightweight block cipher SIMON which was proposed by researchers from the National Security Agency (NSA) in 2013. In particular, we present a search heuristic to find differentials of high probability, and we investigate the clustering of characteristics known as the differential effect. Finally, we apply impossible differential attacks using truncated differentials to a number of SIMON variants. Next, we define a theoretical model for *key-less linear distinguishers*, which captures the meaning of distinguishing a block cipher from an ideal permutation using linear cryptanalysis, when the key is either known or chosen by the adversary. Such models exist using differential properties but were never before defined using linear cryptanalysis. We apply this model to the standardized block cipher PRESENT. Finally, we present very generic attacks on two authenticated encryption schemes, AVALANCHE and RBS, by pointing out severe design flaws that can be leveraged to fully recover the secret key with very low complexity.

In the second part, we delve into the matter of the various aspects of designing a symmetric cryptographic primitive. We start by considering generalizations of the widely acclaimed Advanced Encryption Standard (AES) block cipher. In particular, our focus is on a component operation in the cipher which permutes parts of the input to obtain dependency between the state bits. With this operation in focus, we give a range of theoretical results, reducing the possible choices for the operation in generalized ciphers to a particular set of classes. We then employ a computer-aided optimization technique to determine the best choices for the operation in terms of resistance towards differential- and linear cryptanalysis. Also in the vein of symmetric primitive design we present PRØST, a new and highly secure permutation. Employing existing third-party modes of operation, we present six proposals based on PRØST for the ongoing CAESAR competition for authenticated encryption with associated data. We describe the design criteria, the usage modes and give proofs of security.

Finally, in the third part, we consider implementation aspects of symmetric cryptography, with focus on high-performance software. In more detail, we analyze and implement modes recommended by the National Institute of Standards and Technology (NIST), as well as authenticated encryption modes from the CAESAR competition, when instantiated with the AES. The data processed in our benchmarking has sizes representative to that of typical Internet traffic. Motivated by a significant improvement to special AES instructions in the most recent microarchitecture from Intel, codenamed Haswell, our implementations are tailored for this platform. Finally, we introduce the *comb scheduler* which is a low-overhead look-ahead strategy for processing multiple messages in parallel. We show that it significantly increases the throughput for sequential modes of operation especially, but also for parallel modes to a lesser extent.

Resumé

Emnet for denne afhandling er analyse af symmetriske kryptografiske primitiver. Vi studerer disse objekter fra tre forskellige perspektiver: kryptoanalyse, design, samt implementeringsaspekter.

Den første del handler om kryptoanalyse af symmetriske primitiver, hvor man forsøger at udnytte en egenskab i designet til at opnå et mål som strider med sikkerheden. De to mest succesfulde typer af kryptoanalyse er differentielle- og lineære angreb. Vi benytter varianter af differentiel kryptoanalyse på block cipheret SIMON, som blev fremsat af forskere fra National Security Agency (NSA) i år 2013. Vi giver en søgeheuristik som finder differentialer af høj sandsynlighed, og vi undersøger grupperingen af karakteristikker kendt som differentiel-effekten. Endelig fremstiller vi på baggrund af trunkerede differentialer angreb på adskillige varianter af SIMON med umulige differentialer. Derefter definerer vi en teoretisk model for nøgle-løse lineære distinguishers, som indfanger begrebet at skelne mellem et block cipher og en ideel permutation ved brug af lineær kryptoanalyse, når nøglen enten er kendt eller valgt af angriberen. Sådanne modeller som gør brug af differentielle egenskaber eksisterer allerede, men er aldrig før blevet defineret ved brug af lineær kryptoanalyse. Vi anvender denne model på det standardiserede block cipher PRESENT. Endelig fremsætter vi generiske angreb på to autentificerede krypteringssystemer, AVALANCHE og RBS, ved at påpege alvorlige designfejl som kan udnyttes til fuldstændigt at bestemme den hemmelige nøgle med meget lav kompleksitet.

I den anden del dykker vi ned i forskellige design perspektiver af symmetriske kryptografiske primitiver. Vi starter med at undersøge generaliseringer af det bredt anerkendte Advanced Encryption Standard (AES) block cipher. Navnligt er vores fokus på en operation i AES som permuterer dele af inputtet for at opnå en afhængighed mellem bits i krypteringstilstanden. Med denne operation i fokus giver vi en række teoretiske resultater, som reducerer de mulige valg af operationer i generaliserede ciphers til en række bestemte klasser. Vi bruger dernæst en computerstyret optimiseringsteknik til at bestemme det bedste valg af operation hvad angår modstandsdygtigheden overfor differentiel- og lineær kryptoanalyse. Også i designretningen præsenterer vi PRØST, en ny permutation af høj sikkerhed. Ved brug af eksisterende tredjeparts operationsmodi giver vi seks forslag baseret på PRØST til den igangværende CAESAR konkurrence for autentificeret kryptering med tilhørende data. Vi beskriver designkriterier, anvendelsesmodi og beviser for sikkerheden.

Endelig, i tredje del studerer vi implementeringsaspekter af symmetrisk kryptering med fokus på software af høj ydeevne. Navnligt analyserer og implementerer vi modi anbefalet af National Institute of Standards and Technology (NIST), såvel som modi til autentificeret kryptering fra CAESAR konkurrencen, når disse er instantieret med AES som det underliggende block cipher. Den behandlede data i vores benchmarkings har størrelser som er repræsentative for typisk internet traffik. Motiveret af en signifikant forbedring af specielle AES instruktioner i den seneste mikroarkitektur fra Intel med kodenavnet Haswell, er vores implementeringer skræddersyet til denne platform. Afslutningsvis introducerer vi *comb scheduleren* som anvender en look-ahead strategi af lav overhead til parallelt at processere flere datastrømme. Vi viser at dette giver en betydelig forøgning af throughput for sekventielle modi især, men også for paralleliserbare modi til en mindre grad.

Acknowledgments

First and foremost, I wish to thank my supervisor Christian Rechberger, and my co-supervisor Lars R. Knudsen. Your good spirits and relaxed approach to supervision made my past three years a truly enjoyable experience. My thanks go also to Søren S. Thomsen, my first tutor during my Master's studies, and to Gregor Leander, who was my supervisor before leaving me in Christian's care. Your inspiration played a crucial role in sparking my interest for cryptology. I also thank Anne Canteaut and Thomas Johansson for joining my committee, and Peter Beelen for chairing it.

For the first year of my studies, I was fortunate to be part of the Danish-Chinese Center for Applications of Algebraic Geometry in Coding Theory and Cryptography. During this time I had the opportunity to discuss many topics outside my own area of research. Many thanks go to the whole team for our endeavors around Shanghai, and for our late-night session with the Danish *Højskolesangbog* around the piano in Skagen.

During the past three years, I have been fortunate to work on interesting topics with a number of great people. I wish to thank my co-authors Mohamed Ahmed Abdelraheem, Javad Alizadeh, Hoda A. Alkhzaimi, Elena Andreeva, Mohammad Reza Aref, Nasour Bagheri, Christof Beierle, Andrey Bogdanov, Christoph Dobraunig, Maria Eichlseder, Praveen Gauravaram, Philipp Jovanovic, Elif Bilge Kavun, Lars R. Knudsen, Stefan Kölbl, Abhishek Kumar, Gregor Leander, Atul Luykx, Florian Mendel, Bart Mennink, Christian Rechberger, Somitra Kumar Sanadhya, Martin Schläffer, Peter Schwabe, Tyge Tiessen, Elmar Tischhauser, Tolga Yalçın, and Kan Yasuda for the fruitful collaboration.

I want to thank Vincent Rijmen for having me as a visiting researcher in the COSIC group at Katholieke Universiteit Leuven for four months, during the last year of my studies. Thanks to all the people in COSIC for giving me a warm welcome to their group, and to Péla Noé for helping arrange my stay in Leuven. My special thanks go to Atul, Eva, Tomer and Michal for hosting me, and for having beers with me.

To my former and current employees at the DTU crypto group, Mohamed Ahmed Abdelraheem, Martin R. Albrecht, Hoda A. Alkhzaimi, Subhadeep Banik, Andrey Bogdanov, Julia Borghoff, Christina Boura, Lars R. Knudsen, Stefan Kölbl, Gregor Leander, Christiane Peters, Christian Rechberger, Arnab Roy, Tyge Tiessen, Elmar Tischhauser, and to visiting researchers Farzaneh Abed, Ralph Ankele, Tomer Ashur, Takanori Isobe, Philipp Jovanovic, Florian Mendel, Bart Mennink, Paweł Morawiecki, Sondre Rønjom, and Hongbo Yu: thank you all for the collaboration, discussions and invaluable feedback. You have helped further my understanding of cryptology more than anyone else.

To my childhood friends, the *Insanity* guys, who stand with me to this day: your grotesque humor makes me laugh in ways nothing else can. Thank you so much.

I am deeply grateful to my family, my parents Edith and Hans, and my brothers Søren and Ole. You have given me the opportunity and encouragement to pursue my ambitions. Thank you for that. My thanks go also to the Schadegg family, my in-laws who have always shown a great interest in my work.

Finally, to Sarah, my partner in this wonderful thing called life: you alone are my biggest support of all. Thank you so much, for being who you are, and no one else.

Martin M. Lauridsen

Kgs. Lyngby, August 2015

Notation

By a *bit* we mean a value from the set $\{0,1\}$, which we also identify with the finite field $\mathbb{F}_2$. We use interchangeably several different representations of a binary string (of length n), which we all identify with each other: as a string of n symbols from the alphabet $\{0,1\}$; as an integer in the range $0,\dots,2^n-1$ written in any radix; as a vector in $\mathbb{F}_2^n$; and as a element in the field $\mathbb{F}_{2^n} = \mathbb{F}_2[x]/f(x)$ where $f(x)$ is an irreducible polynomial of degree n. For a binary string X, we use X_i to denote the i^{th} bit of X, where X_0 denotes the least significant bit.

Letting K be a set (or another mathematical structure considered as a set) we denote vectors with elements from K by K^* when the length can be arbitrary (including zero); by K^+ when the length is positive; and by K^m when the length is some positive integer m. For example, we denote by $\mathbb{F}_2^n$ the set of n-bit binary strings and $\mathbb{F}_2^*$ the set of binary strings of any length. We let the notation extend to single symbols, such that e.g. 0^n denotes the binary string consisting of n zeroes. Strings in `monospace` font represent numbers written in hexadecimal (radix 16) notation.

List of Symbols

$\mathbb{N}$	The natural numbers $\{1,2,3,\ldots\}$
$\mathbb{N}_0$	The natural numbers with zero $\mathbb{N} \cup \{0\}$
$\mathbb{Z}$	The integers $\{\ldots,-2,-1,0,1,2,\ldots\}$
$\mathbb{Z}_n$	The ring of integers modulo n; or the set $\{0,\ldots,n-1\}$
$\mathbb{F}_{q^n}$	The finite field of size q^n with q prime and $n \in \mathbb{N}$
$\mathsf{Perm}(n)$	The set of all permutations on $\mathbb{F}_2^n$
$\mathsf{OPerm}(n)$	The set of *online permutations* on $\mathbb{F}_2^n$ (see Section 3.2.6)
$\sharp S$	The number of elements in the set S
$X \| Y$ or XY	Concatenation of binary strings X and Y
$X \oplus Y$	Bit-wise addition of X and Y
$X \odot Y$	Bit-wise multiplication of X and Y
$X \mid Y$	Bit-wise OR of X and Y
$X \boxplus Y$	Modular addition (modulus understood from context)
$X \boxminus Y$	Modular subtraction (modulus understood from context)
$X \ll k$	Binary left shift by k positions; or X is "much less than" k
$X \gg k$	Binary right shift by k positions; or X is "much greater than" k
$X \lll k$	Binary left rotation by k positions
$X \ggg k$	Binary right rotation by k positions
$\lvert X \rvert$	Bit-length of X; or absolute value of X
ε	The empty binary string
$\mathbf{e}_i$	Binary string with a 1-bit on position i and zeroes elsewhere
$\mathsf{lsb}_k(X)$	The k least significant bits of the binary string X
$\mathsf{msb}_k(X)$	The k most significant bits of the binary string X
$\mathsf{pad}_k(X)$	Padding binary string X
$X \overset{\$}{\leftarrow} S$	Sampling of X uniformly at random from a set S
$\Pr[E]$	Probability of an event E
$\mathbb{E}[X]$	Expected value of a random variable X
$\mathscr{B}(N,p)$	Binomial distribution with N samples of probability p
$\mathcal{N}(\mu,\sigma^2)$	Normal distribution with mean μ and variance σ^2

List of Abbreviations

AE(AD)	Authenticated Encryption (with Associated Data)
AES	Advanced Encryption Standard
AES-NI	AES New Instructions
CAESAR	Competition for Authenticated Encryption: Security, Applicability, and Robustness
CBC	Cipher Block Chaining
CCM	Counter with CBC-MAC
CFB	Ciphertext Feed Back
CTR	Counter (mode of operation)
DDT	Difference distribution table
DES	Data Encryption Standard
ECB	Electronic Code Book
GCM	Galois/Counter Mode
IPsec	Internet Protocol Security
IV	Initialization vector
MAC	Message Authentication Code
MILP	Mixed-integer linear programming
NIST	National Institute of Standards and Technology
NMR	Nonce misuse resistance
NSA	National Security Agency
SAC	Strict avalanche criterion
SEM	Single-Key Even-Mansour construction
SHA	Secure Hash Algorithm
SSH	Secure Shell
SSL	Secure Sockets Layer
TLS	Transport Layer Security
XE	Xor-encrypt
XEX	Xor-encrypt-xor

Contents

1 Introduction

In this chapter, we present an introduction to the concept of the *symmetric cryptography*. The idea is, that after covering this chapter, the reader should be set with the necessary tools and concepts to understand the results presented in subsequent Chapters 2 through 4.

First, in Section 1.1, we discuss the purpose of using symmetric cryptography and we provide the setting in which the rest of the introduction is framed. Section 1.2 introduces the five most important constructions in symmetric cryptography, four of which are major topics of this thesis. While Section 1.2 discusses how the goals of cryptography as introduced in Section 1.1 can be obtained, we turn to analyzing symmetric constructions using various attack techniques in Section 1.3, under the term *cryptanalysis*.

1.1 Cryptographic Goals

To explain the fundamentals of cryptography and why we need them, we consider two parties communicating over an insecure channel. We typically name the parties Alice and Bob[1]. The channel is considered insecure, because Eve, the malicious adversary, is trying to thwart the goals of Alice and Bob in any way possible. The very basic setup is depicted in Figure 1.1.

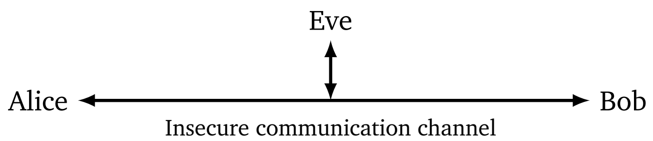


Figure 1.1: Two parties communicating over an insecure channel

[1]Alice and Bob are placeholder names that have found their way into any cryptography textbook, since their introduction with the RSA paper in 1977

While, arguably, most people think cryptography is about keeping communications secret, this is only part of the truth. In reality, cryptography provides three important properties that guarantee the security of the communication between Alice and Bob. The three properties are:

- **Confidentiality**: When Eve is a *passive adversary*, meaning she listens to the communication between Alice and Bob without modifying any transmitted data, it should not be possible for her to infer anything about the messages exchanged between the communicating parties.

- **Integrity**. When Eve is an *active adversary*, meaning she can modify the data being transferred over the insecure channel, the communicating parties should be able to detect when such modification has occurred.

- **Authenticity**: When Eve is an active adversary, the communicating parties should be able to verify that they are indeed communicating with whom they think they are communicating with. In particular, it should not be possible for Eve to send messages to Bob, appearing as though they originated from Alice, or vice versa.

A plethora of constructions in cryptography exist which try to obtain one or more of these three properties in a wide range of applications. In general, we can categorize the constructions into three groups: *symmetric*, *asymmetric* and *unkeyed* primitives. The symmetric primitives are named so, because a prerequisite to use them is, that the communicating parties posses some secret, shared information, which is used to secure the communication. This piece of information is called a *key*. The security of the primitives completely fail if the key becomes known to an unauthorized party. For this reason, these primitives are also referred to as *secret-key* primitives. Correspondingly, the asymmetric primitives are constructions in which each communicating party has a *pair* of keys: a *private key* which should be known only to the holder, and a *public key* which can be freely given to anyone who wishes to communicate with the holder. For this reason, asymmetric primitives are also referred to as *public-key* primitives. The benefit of the public-key primitives are obvious: they can be used even without the assumption that the communicating parties posses any shared data. Finally, the category of unkeyed primitives are constructions which, as the name indicates, do not require neither a shared, secret key, nor a public key to serve their purpose. Such purposes may include the generation of pseudo-random numbers or computing a message digest (hashing).

In this thesis, we focus almost explicitly on symmetric primitives. In particular, we will be discussing certain aspects of designing symmetric primitives, but also how one might attempt to render such constructions insecure. We also discuss implementation aspects of these primitives; a point which is becoming increasingly more important with the ubiquity of tiny computing devices in our everyday lives. Besides our treatment of symmetric primitives, will also cover the topic of the unkeyed primitive of cryptographic hash functions. In that regard, our discussions on cryptographic hash functions will mostly be in relation to their construction from symmetric primitives.

1.2 Symmetric Primitives

Symmetric primitives can be considered the LEGO blocks of symmetric cryptography: we use them to build *something bigger*. Typically, this means employment in protocols that enable secure communication, such as Secure Shell (SSH), Secure Sockets Layer (SSL), Transport Layer Security (TLS), Internet Protection Security (IPsec), and so on. In the following, we describe a range of cryptographic primitives, with particular focus on symmetric primitives. We also discuss cryptographic hash functions which are in fact unkeyed primitives. However, as the design and analysis of the most widespread hash functions is closely related to that of symmetric primitives, we consider them all under one umbrella.

As already mentioned, symmetric primitives is the name we use for cryptographic constructions that have the prerequisite that the communicating parties share some particular piece of secret information that no unauthorized party can know. We also hinted that this impracticality is not present for the public-key primitives mentioned. What we did not mention is, that the public-key primitives are several orders of magnitude slower than the symmetric primitives. Thus, when two parties, e.g. a web browser and a web server, must start to communicate in practice, one uses a mixture of both. In particular, a public-key primitive is used *once* to *securely* (but slowly) exchange a piece of secret information, which can then be used as the secret key for the much faster symmetric primitives.

In our following treatment, we pay special attention to *block ciphers* in Section 1.2.1 and *authenticated encryption schemes* in Section 1.2.4, as this thesis to a large extent is about their design, analysis and implementation. Finally, we cover cryptographic hash functions in the end, which are strictly not symmetric primitives, but rather unkeyed.

1.2.1 Block Ciphers

In the world of symmetric primitives, block ciphers are extremely versatile building blocks which can be used in the designs of many other cryptographic primitives, including stream ciphers, hash functions and MACs (all of which we describe in the following). Returning to the three goals of cryptography, the purpose of block ciphers is to provide confidentiality. A block cipher is a cipher mapping a fixed-length input, called a *block*, together with a key, to an output block. We give their formal definition next.

Definition 1 (Block cipher). *Let κ and n be positive integers. A* block cipher *is an* encryption function

$$\begin{aligned} \mathscr{E} : \mathbb{F}_2^{\kappa} \times \mathbb{F}_2^{n} &\to \mathbb{F}_2^{n} \\ (K, M) &\mapsto C, \end{aligned} \tag{1.1}$$

which, for any fixed $K \in \mathbb{F}_2^{\kappa}$, acts as a permutation on $\mathbb{F}_2^n$. The parameter κ is called the key size *or* key length*, and the parameter n is called the* block size*. We refer to K, M and C, respectively, as the* key*, the* message *(or* plaintext*) and the* ciphertext*. For a fixed key K, the inverse $\mathscr{D} = \mathscr{E}^{-1}$ is the associated block cipher* decryption function*.*

We introduce the notation $\mathscr{E}_K(\cdot) = \mathscr{E}(K, \cdot)$ and $\mathscr{D}_K(\cdot) = \mathscr{D}(K, \cdot)$, and use them interchangeably throughout this thesis. The generic depiction of a block cipher is illustrated in Figure 1.2. We

$$\begin{tikzcd} & K \arrow[d] & \\ M \arrow[r] & \boxed{\mathscr{E}} \arrow[r] & C \end{tikzcd}$$

Figure 1.2: A block cipher

remark that the correctness of a block cipher is given by the requirement that

$$\forall K \in \mathbb{F}_2^\kappa, \forall M \in \mathbb{F}_2^n : \mathscr{D}(K, \mathscr{E}(K, M)) = M, \tag{1.2}$$

which states that for a fixed K, $\mathscr{D}_K$ is the inverse of $\mathscr{E}_K$. As stated by the definition, for a fixed key K, we require that $\mathscr{E}_K$ is a permutation on $\mathbb{F}_2^n$. There are $2^n!$ such permutations and 2^κ possible values for the key K. Thus, if $2^\kappa \ll 2^n!$ (which is always the case in practice), then the encryption functions that are ever used are far from covering the family of all permutations on $\mathbb{F}_2^n$. For that reason, it is important that the number of *actual* encryption functions 2^κ is large enough that the adversary Eve is not able to determine the correct, secret value of K simply by trying all possibilities. In practical systems, it is most common to have $\kappa \geq n$. Typically, we see values of $\kappa \in \{64, 80, 96, 128, 192, 256\}$ and for the block size it is most common that $n \in \{64, 128, 256\}$. A recent study by the European Network and Information Security Agency (ENISA) from 2014 [286] recommends that for block ciphers, the key size is $\kappa \geq 128$ and that one should encrypt at most $2^{n/2}$ messages using the same key K. If Alice and Bob can agree on a block cipher to use, together with a fixed key $K \in \mathbb{F}_2^\kappa$ of sufficient length, which is not known by any unauthorized party, then their communication can remain confidential, provided the block cipher is secure.

It is important to understand, that no one block cipher suits all purposes. The design choices made for a particular block cipher directly influence not only the security, but also practical aspects including efficiency and other platform-dependent characteristics such as implementation size in hardware, production cost, etc. For example, an application might only realistically give up a tiny portion of hardware circuit for cryptographic primitives, while still requiring that the cryptographic implementation has a very high performance. In such a case, it may be that one has to obtain these properties at a loss of security, which may or may not be acceptable in the particular scenario. In the end, the choice of a block cipher for a particular application comes down to a trade-off with many parameters involved.

Two important concepts to cipher design, identified by Claude Shannon in his seminal paper [284] from 1949, are those of *confusion* and *diffusion*. Confusion refers to the relationship between the message and the ciphertext; the relationship should be as complex as possible. A modern interpretation of confusion, due to Massey [192], is that statistics on the ciphertext must not depend on that of the plaintext, in a way which is exploitable by the cryptanalyst. We will see examples in the following, of exactly how this is accomplished. The concept of diffusion says that any statistical properties of the message, which potentially could reveal information about the content of said message, should not be present in the ciphertext. Again, a more modern interpretation loosely says that each bit message and key material should influence many bits of the ciphertext. We remark that this implies that a block cipher should be highly non-linear. For example, a block cipher where some bit of ciphertext depends linearly on some message bits and key bits, is weak because the particular key bits can be easily recovered. The consideration of linear approximations involving message-, key- and ciphertext bits forms the basis of what is known as *linear cryptanalysis*, an attack method we describe in Section 1.3.

In the following, we proceed with describing how block ciphers can be constructed. We limit our treatment to what is known as *iterated designs*, as this is the prevailing design approach in modern block ciphers.

Iterated Designs

Most modern block ciphers, and indeed all block ciphers considered in this thesis, follow an *iterated* design strategy. In such a design, a core component is being iterated again and again, up to a small change each time. Iterated designs are also often referred to as *round-based* designs, as each iteration of the core component is called a *round function*, or *round* for short. As such, the encryption function can be written as the composition of T rounds,

$$\mathscr{E}_K = F_{T-1}(K_{T-1}, \cdot) \circ \cdots \circ F_0(K_0, \cdot), \tag{1.3}$$

where $F_t(K_t, \cdot)$ is the t^{th} round and K_t is called the t^{th} *round key*. To ensure the correctness of the block cipher, we remark that each round function F_t is required to be a bijection on $\mathbb{F}_2^n$. Let us denote the bit length of each K_t by m. In many cases, this round key size equals the block size, i.e. $m = n$, and $T \cdot m$ is much larger than κ. As such, we need to expand the key K (called the *master key*) into several round keys. This is accomplished by a *key schedule algorithm*, which we denote by a function

$$\begin{aligned} KS : \mathbb{F}_2^\kappa &\to \left(\mathbb{F}_2^m\right)^T \\ K &\mapsto (K_0, \ldots, K_{T-1}). \end{aligned} \tag{1.4}$$

A particular subclass of iterated block ciphers are referred to as *key-alternating* ciphers. Here, the round keys are not directly part of the round function itself as indicated in Eq. (1.3), but rather the round keys are added (using the XOR operation) to the block state in between each round function F_t. Such a construction is shown in Figure 1.3. Naturally, such a key-alternating design implies that the round key size m equals the block size n. As indicated in Figure 1.3, extra round keys are added before the first round function and after the last round function. Such keys are called *whitening keys*. Their intention is, that an adversary should not be able to directly control the input to- and output from the first- and last rounds, respectively. This helps to increase the hardness of a type of attacks known as *meet-in-the-middle* (MiTM) attacks (see Section 1.3.7 and e.g. the work of Fouque and Karpman [135]). We remark, that for a key-alternating cipher, one needs one more round key than the number of rounds, and thus the co-domain of the key schedule algorithm of Eq. (1.4) would be $\left(\mathbb{F}_2^n\right)^{T+1}$.

The reason that iterated designs are particularly appealing is twofold. First, as the round functions are nearly identical, this means that when implementing the block cipher, the code can be reused in software implementations and the electrical circuit can be reused for hardware implementations. This means that optimization efforts can be focused on the (comparatively) small core component, i.e. the round function, in order to achieve an optimized overall implementation of the block cipher. Second, it is much easier to understand one small core component,

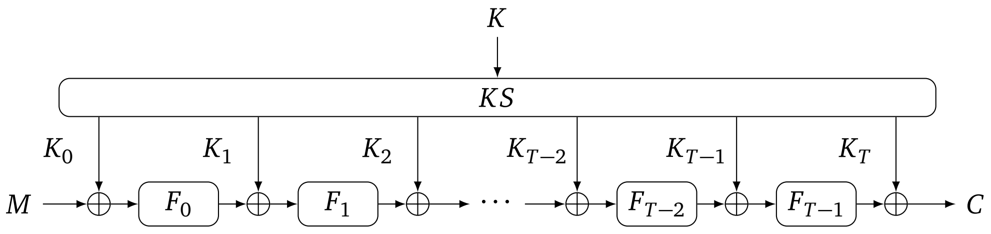


Figure 1.3: Key-alternating block cipher construction of T rounds

and how the overall block cipher behaves when it is repeated, than it is to understand a complex composition of many different components. This yields obvious benefits to the designer of the block cipher. If it is possible to predict, or even prove, how local properties of the core component behave when iterated a number of times, this is of great help to the designer who wants to determine how many round function applications are required to attain a certain security level. On the other hand, cryptanalysts whose job it is to try to break ciphers, can also gain insight from this very systematic design approach. Sometimes, this can lead to broken designs. In general, as there is no way to effectively *prove* the security of a block cipher, we believe them to be strong when they have stood the test of time without being broken. Especially, requirements for a good block cipher are good design characteristics, and much cryptanalytic effort put into analyzing the cipher, without it being broken. The iterated design approach follows the philosophy, known as *Kerckhoffs' principle*, that a cipher should be secure because it is well understood, and no feasible attack has been found, rather than obscurity of the design [192, Chapter 1].

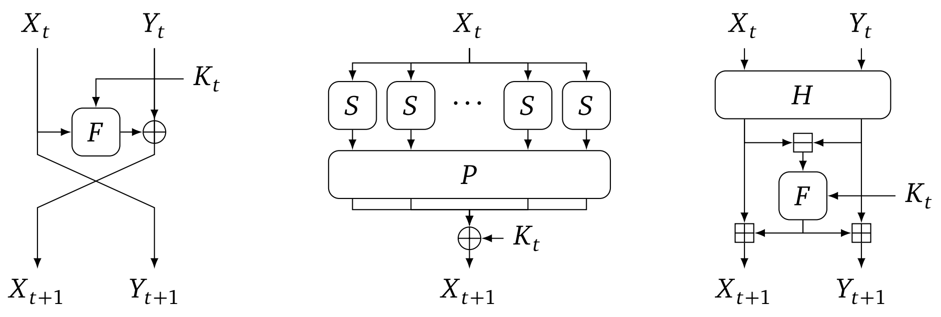
In iterated block cipher designs, we deal with three general types: *Feistel ciphers*, *substitution-permutation networks* (*SP-networks* or *SPNs* for short) and *Lai-Massey schemes*. We briefly describe each of the three in the following. Later in this thesis, we will see examples of the two former types.

Feistel Ciphers

Feistel ciphers owe their name to Horst Feistel whom, when working for IBM, designed the Lucifer cipher in 1973 [123]. The most well-known example of a Feistel cipher today is the Data Encryption Standard (DES) [98] which was developed by IBM, based on Lucifer with modifications by the National Security Agency (NSA) [175]. Other examples include FEAL [233], Twofish [281], Camellia [27] and SIMON [40], the latter being a recent lightweight design by the NSA. We discuss and present cryptanalytic results for the block cipher SIMON in Section 2.1.

During the encryption process, the state is typically maintained in two halves, called the *left* and *right* halves, each of $n/2$ bits, and we write the state as (X_t, Y_t). A single round consists of applying a cryptographically weak update function F to one of the halves (without loss of generality, the left half X_t) of the state, together with a round key. The output is added using XOR to the other half of the state, and the two halves are swapped. Typically, the last round is special, as the halves are not swapped, because this would only incur computational overhead. As such, the general round function can be described as

$$\left(X_{t+1}, Y_{t+1}\right) = \left(F_{K_t}(X_t) \oplus Y_t, X_t\right). \tag{1.5}$$



(a) Feistel construction **(b)** SP-network **(c)** Lai-Massey scheme

Figure 1.4: Round functions for three common block cipher constructions

A round of a general Feistel construction is illustrated in Figure 1.4a. This round function is iterated many times to obtain a cryptographically strong block cipher. A property of Feistel ciphers that is very appealing to hardware implementations is, that encryption and decryption look very similar, and indeed one needs just re-order the round keys to obtain one from the other. With Feistel ciphers, the round function F is only ever applied in one direction, and thus, it needs not be invertible.

Substitution-Permutation Networks

The widely acclaimed block cipher design approach of *substitution-permutation networks*, is another paradigm following the iterated approach. In SPNs, the round function consists generally of steps of three types:

1. **Substitution:** The current state (or parts of it) are substituted with new values in a non-linear fashion, i.e. the new values should not have a linear expression in terms of the input values. This step is typically implemented using *substitution boxes*, or *S-boxes* for short, using a lookup table.

2. **Permutation:** An efficient permutation on $\mathbb{F}_2^n$ is applied to the state. Typically, this is accomplished using the combination of two steps. Firstly, the state bits are permuted either by mapping bits to new positions in the state individually, or by mapping larger portions (words) of the state to new positions, at the same time. Secondly, a linear mixing is applied to the state, in which each transformed output bit of the state can be written as a *linear* combination of bits from the input state. The linear mixing is typically accomplished by multiplication by a binary matrix, or by a matrix over an appropriate finite field of size 2^w for some word size w.

3. **Key addition:** The round key K_t is mixed into the current state, most typically using the XOR operation.

The general round function of an SP-network is illustrated in Figure 1.4b. The different types of operations of the SPN round function serve different purposes. The purpose of the S-boxes are to create non-linearity in the round function. Without these, it would be very easy to determine precisely which bits of the ciphertext depend on each bit of the message. Coming back to Shannon's desirable properties of a cipher, the purpose of the key addition is to create confusion and the purpose of the permutation is to create diffusion in the cipher.

The most prominent example of an SPN is the Advanced Encryption Standard (AES) [252]. In 1997, the necessity of a new encryption standard, eventually to become the AES, became evident after cryptographic researchers found serious security flaws in the DES block cipher in the mid 90s. The AES is a 128-bit block cipher standardized by the National Institute of Standards and Technology (NIST), following a nearly five year long period from call for proposals in January 1997 to standardization in November 2001. While the story of how the AES came to be is a fascinating one, we will not dig into it here. Instead, we refer the interested reader to [106]. The AES is comprised of three block ciphers using a block size of $n = 128$ and key sizes $\kappa \in \{128, 192, 256\}$. These three block ciphers are a subset of the *Rijndael* block cipher family, which was the winner from all the submissions, selected by NIST [252]. The name Rijndael comes from the contraction of the names of its authors, Joan Daemen and Vincent Rijmen, two Belgian cryptographers.

Currently, the AES is perhaps the most used block cipher in the world. What is very interesting is that, contrary to the DES, the decision on the AES was a very public process, compared to the DES. The design of the AES is particularly interesting, as it defines a new design approach coined the *wide-trail design strategy* by Daemen and Rijmen [105]. This approach was a new way, at the time, of obtaining confusion and diffusion in a block cipher. We describe the AES in full detail in Section 3.1, where, in particular, we discuss symmetric primitives with a *generalized* AES structure, and how we might choose good parameters for such primitives.

Lai-Massey Schemes

The final type of block cipher construction we describe are Lai-Massey schemes. Constructions following this approach are not so widespread as the former Feistel and SPN constructions. A prominent example includes the IDEA block cipher by Lai and Massey (for a description of IDEA, see the thesis of Lai [200]).

Like Feistel ciphers, Lai-Massey schemes also split the block cipher state into two parts of $n/2$ bits each. Two functions, not necessarily invertible, are used,

$$\begin{aligned} H &: \mathbb{F}_2^n \to \mathbb{F}_2^n \\ F &: \mathbb{F}_2^{n/2} \to \mathbb{F}_2^{n/2}. \end{aligned} \tag{1.6}$$

The function H is called the *half-round function* and F is the round function. The state (X_t, Y_t) is updated to produce (X_{t+1}, Y_{t+1}) as

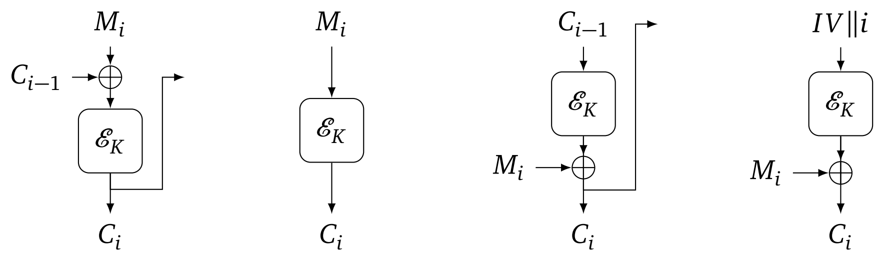
$$\begin{aligned} (X_t', Y_t') &= H(X_t, Y_t) \\ T_t &= F_{K_t}(X_t' \boxminus Y_t') \\ (X_{t+1}, Y_{t+1}) &= (X_t' \boxplus T_t, Y_t' \boxplus T_t). \end{aligned} \tag{1.7}$$

A single round of a general Lai-Massey scheme is depicted in Figure 1.4c.

Modes of Operation

So far, in our treatment of block ciphers, we have been discussing encryption functions mapping from $\mathbb{F}_2^n$ to $\mathbb{F}_2^n$, i.e. permutations on n-bit strings of data. Naturally, in real applications, communication to be encrypted has a size much larger than the typical block size of $n = 128$ bits. So what happens in such a scenario? We need a way to extend the use of block ciphers to such use cases. The solution is to use a block cipher *mode of operation*. We assume that we are given a message, being a binary string of several bits. We can now split this string into blocks of n bits. If the total length of the string is not a multiple of n, we employ a *padding scheme* to handle this issue, such that the input becomes compatible with an integral number of block cipher calls.

At the time of writing, the perhaps most popular block cipher mode of operation is *cipher block chaining* (CBC) together with the AES block cipher. For example, it is one of the most frequently used cipher suite choices in the TLS protocol. Other prominent modes are *electronic code book* (ECB), which is highly insecure and never recommended for use (as we shall see below), *ciphertext feed back* (CFB) mode and *counter* (CTR) mode. We describe each of them briefly in the following. The encryption of the i^{th} message block M_i, using each of the four block cipher modes CBC, ECB, CFB and CTR, is depicted in Figure 1.5.



(a) CBC mode **(b)** ECB mode **(c)** CFB mode **(d)** CTR mode

Figure 1.5: Block cipher modes of operation

- **CBC**: The CBC mode of operation takes as input a number ℓ of message blocks $M_1, \dots, M_\ell$, each of n bits. It also requires an n-bit *initialization vector* (IV). Encryption proceeds as $C_i = \mathcal{E}_K(M_i \oplus C_{i-1})$, where we define $C_0 = IV$. With CBC, the ciphertext block C_i depends only on the message blocks $M_1, \dots, M_i$. As such, if the IV is used twice to encrypt two different messages $M_1, \dots, M_\ell$ and $M'_1, \dots, M'_{\ell'}$, and $M'_1, \dots, M'_j$ are equal to $M_1, \dots, M_j$ for some $j > 0$, then the corresponding ciphertext blocks are equal, i.e. $C'_i = C_i$ for $1 \leq i \leq j$.

- **ECB**: As stated already, ECB is a very insecure mode of operation and should not be used. The reason is, that the encryption of message block M_i does not depend on the encryption of any other message block M_j for $j \neq i$. As such, it operates by encrypting each message block so $C_i = \mathcal{E}_K(M_i)$, and the resulting ciphertext blocks are concatenated to form the complete ciphertext. This means that an adversary can, for example, form arbitrary valid message/ciphertext pairs, once she has observed any single message/ciphertext pair.

- **CFB**: Ciphertext feed back mode, like CBC, uses an IV value of n bits. The i^{th} ciphertext block, $1 \leq i \leq \ell$, is obtained as $C_i = \mathcal{E}_K(C_{i-1}) \oplus M_i$, where we define $C_0 = IV$. CFB has the same chaining dependency property as CBC, which also means that any error in the reception of C_i results in incorrect decryption of all following ciphertext blocks. A very similar mode, *output feedback mode* (OFB), produces ciphertext in the same way, but sets $C_i = \mathcal{E}_K(C_{i-1} \oplus M_{i-1}) \oplus M_i$.

- **CTR**: The CTR mode of operation is rather unlike the other modes mentioned. It still uses an IV (called a *nonce*) of $|IV| < n$ bits, which should not be repeated for different messages. The nonce is concatenated with a counter value of $n - |IV|$ bits, starting from zero, to form an n-bit block. This block is now encrypted using the block cipher, and the output is XORed to the message block to provide the ciphertext. For the next message block, the same nonce is concatenated with a counter value increased by one, and so on. As such, in CTR mode, the block cipher input is deterministic and does not depend on the message block at all. Previously, there has been some discussion as to whether this deterministic input to the block cipher is a bad idea. While CTR is a quite popular mode of operation, some argue that properly managing a counter can be difficult in some applications. See [213] for a discussion on CTR mode by Lipmaa, Rogaway, and Wagner.

1.2.2 Stream Ciphers

While block ciphers encrypt blocks of a fixed number of bits at a time, *stream ciphers* work by encrypting single bits at a time. This is done by considering the message as a string of arbitrary length $|M|$ and combining the message with a so-called *key stream*, a binary string Z of length $|M|$. We can define a stream cipher in the following way.

Definition 2 (Stream cipher). *Let κ and η be positive integers. A* stream cipher *is given by a function*

$$\begin{aligned} \mathcal{S} : \mathbb{F}_2^{\kappa} \times \mathbb{F}_2^{\eta} \times \mathbb{F}_2^{*} &\to \mathbb{F}_2^{*} \\ (K, IV, M) &\mapsto C = M \oplus Z, \end{aligned} \tag{1.8}$$

where Z is the key stream generated by the stream cipher.

A favorable property of stream ciphers is, that individual message bits can be encrypted on the fly, which can be advantageous in real-time systems. Also, errors in transmission affect only bits locally, so it does not propagate to other parts of the decrypted message, as we saw was the case with e.g. CBC mode. This makes stream ciphers favorable in applications with a high error rate.

One way to think about stream ciphers is, that they try to mimic the *one-time pad*, also called the *Vernam cipher*. The one-time pad computes the ciphertext as $C = M \oplus K$, where $|K| = |M| = |C|$. Assuming the key is chosen randomly and never repeated, it can be shown that the cipher is theoretically unbreakable, i.e. an attacker can do no better than try each combination of the key [228, Section 1.5.4]. In terms of Eq. (1.8), the one-time pad uses a completely random string Z, while a stream cipher generates a pseudo-random string Z.

Stream ciphers are motivated by the obvious impracticality of the one-time pad: one needs to obtain just as many random bits as the number of bits in the message one wants to encrypt.

Stream ciphers are stateful and use the secret key K, possibly in combination with an IV, to maintain a state which is updated over time, and which at the same time produces bits for the key stream Z. We remark that the CTR block cipher mode, discussed in Section 1.2.1, can be considered as a stream cipher, in the sense that it generates a key stream Z, being the outputs from the block cipher calls, and adds these using XOR to the message blocks to provide the ciphertext blocks.

While there is much and more to be said about stream ciphers, we do not go into their detail here. We suggest the interested reader to look into eSTREAM [250], an EU-funded project running from 2004 to 2008, with the purpose of selecting a new range of stream ciphers suitable for widespread use.

1.2.3 Message Authentication Codes

A *Message Authentication Code*, or MAC for short, can be considered as a keyed fingerprint computed on a plaintext of any length. The purpose of the MAC is twofold: it should provide integrity and authenticity.

Consider the scenario where Alice uses e.g. a block cipher to encrypt a message M to obtain a ciphertext C, which she wants to send to Bob. If she computes a key-dependent MAC on the message and attaches it to the ciphertext, Bob can, on the receiving end, first decrypt to obtain the message and use his copy of the secret key to compute the MAC. If the MAC he computes matches the one Alice attached to the ciphertext, he trusts that the message indeed came from Alice. On the other hand, if the two MACs do *not* match, a modification happened to the MAC, the ciphertext or both, when being transferred from Alice to Bob, be it due to a faulty channel or the malicious adversary Eve.

We remark that a MAC, as we shall see, is a many-to-one function. Thus, it is mathematically possible for Eve to find a ciphertext $C' \neq C$ that decrypts to a message $M' \neq M$ which has the same MAC as M. In this case Bob will not be able to detect any suspicious behavior. The important thing is that it should be *hard* for Eve to do this. Generally, the idea is that the problem of providing a modified ciphertext together with a modified MAC which appears valid on Bob's end, without knowing the secret key, should be just as hard as finding the secret key. Functionally, we define a MAC in the following.

Definition 3 (Message Authentication Code). *Let κ and n be positive integers. A* message authentication code *(MAC), is a keyed fingerprint (or hash) of a message of arbitrary length,*

$$\mathsf{MAC} : \mathbb{F}_2^\kappa \times \mathbb{F}_2^* \to \mathbb{F}_2^n, \tag{1.9}$$

where we use the short notation that $\mathsf{MAC}_K(\cdot) = \mathsf{MAC}(K, \cdot)$. The parameter κ is the key size and n is the MAC size.

There are two main desirable properties of a MAC:

1. **Computability**: The MAC should be efficiently computable, and

2. **Computation-resistance**: Without knowledge of the secret key K, it should be computationally infeasible to determine a valid pair $(M, \mathsf{MAC}_K(M))$, even with the knowledge of *other* valid $(M_i, \mathsf{MAC}_K(M_i))$ pairs, where each $M_i \neq M$.

A MAC which does not offer computation-resistance is said to be subject to *MAC forgery*, i.e. an attacker can forge a valid pair $(M, \mathsf{MAC}_K(M))$ without knowledge of the secret key K. Note, however, that this does not imply that the adversary is able to recover K, but the other implication is, of course, true. That is, if the adversary is able to obtain the secret key K, she can encrypt any message M of her choosing to get $C = \mathscr{E}_K(M)$ *and* compute $\mathsf{MAC}_K(M)$.

When an adversary tries to thwart the purpose of a symmetric scheme, the attack usually works in two phases. In the first phase, data is collected under some adversarial assumption. In the second phase, the adversary attempts to use the collected data to obtain some goal. In Section 1.3, we describe the concepts of adversarial assumptions and goals in detail. However, for the purpose of understanding what a MAC should be able to provide, we discuss adversarial assumptions and goals for them at this point. We consider three types of adversarial assumptions when talking about the security of a MAC:

- **Known-text**, where the adversary is assumed to possess either one or several valid pairs $(M_i, \mathsf{MAC}_K(M_i))$,
- **Chosen-text**, where the adversary is assumed able to obtain valid pairs $(M_i, \mathsf{MAC}_K(M_i))$ for M_i of her choosing, and
- **Adaptively chosen-text**, which is like a chosen-text assumption, except that the adversary can choose each M_i depending on the values of previous pairs $(M_j, \mathsf{MAC}_K(M_j))$ for $j < i$.

We also make a distinction between the types of forgery, i.e. the adversarial goals, an attacker is capable of doing. First, a *selective forgery* is when the adversary is able to (partially) choose a *new* message M and provide the correct value of $\mathsf{MAC}_K(M)$. Second, an *existential forgery* is when the adversary can provide a *new* valid pair, but she is not able to control the value of M. Clearly, a selective forgery is the harder adversarial goal of the two, as it implies an existential forgery.

The canonical example of why MACs are required, and in particular why encryption is not enough in itself, is the example of a bank transfer request. Say, Alice sends Bob an encrypted message with her bank credentials because Bob owes her money. Without a MAC, Eve would be able to manipulate, without being detected, the destination account to which Bob transfers his owed amount. However, she can still do this even with a MAC, if she is able to forge. Unfortunately, it is quite common in real applications to see encryption implemented without any authentication or data integrity. This can, perhaps, be credited to introductory texts focusing solely on the importance of encryption and decryption, and not stressing the importance of integrity and authentication.

Popular MACs include CBC-MAC, *hash-based MAC* (HMAC) and *parallelizable MAC* (PMAC). We briefly describe these in the following.

- **CBC-MAC** is a block cipher-based MAC algorithm, which operates by processing the message blocks in CBC mode with $IV = 0$. Instead of producing any ciphertext output, the block cipher outputs are fed back and added to the next message block, just as in regular CBC. Finally, the output of the last block cipher call is taken to be the MAC value.
- **HMAC** uses not a block cipher but a *hash function* $\mathscr{H}$ (we describe these in Section 1.2.5). For now, think of $\mathscr{H}$ as a function $\mathscr{H} : \mathbb{F}_2^* \to \mathbb{F}_2^n$ with certain desirable properties. Now,

HMAC uses certain padding values p_1 and p_2, and the MAC is computed on a message M as

$$\mathsf{HMAC}(M) = \mathscr{H}(K \| p_1 \| \mathscr{H}(K \| p_2 \| M)). \tag{1.10}$$

- **PMAC** is a more recent design by Black and Rogaway [65]. It is a block cipher-based mode, motivated by the inherently sequential nature of both CBC-MAC and HMAC. The PMAC design, on the other hand, is fully parallelizable, which is highly desirable on modern, high-end CPUs. The MAC is computed by XORing a *mask* (a particular, cleverly chosen value) to each message block before applying the block cipher to each message block. The outputs are then all XORed together, and a final block cipher call is made to provide the MAC value. There are a few more details, but we do not describe those here. Most importantly, the n-bit masks that are added to the message blocks need to all be different.

1.2.4 Authenticated Encryption

When a block cipher is combined with a MAC, we obtain what is called *authenticated encryption*. This way of obtaining authenticated encryption, i.e. by combining two primitives, is called *generic composition*. Another currently trending design approach in symmetric primitives is *authenticated encryption with associated data* (AEAD). Such primitives essentially obtain the confidentiality of encryption and the authenticity and integrity provided by a MAC, but the two are included in the algorithm design from the beginning. It comes with the added functionality of allowing authentication of data which is not encrypted (the associated data) which may include e.g. information about the destination of an IP packet on a network, which must not be encrypted, because otherwise routers would not be able to correctly forward the packet to its recipient.

At the time of writing there are two main players in the field of AEAD, and both are block cipher modes of operation: *Galois/Counter Mode* (GCM) [121] and *Counter with CBC-MAC* (CCM) [120]. Both GCM and CCM, when used with the AES, are part of the TLS 1.2 cipher suite [114], and thus are widely used. Currently, there is a massive ongoing effort in the cryptographic community, channeled by Dan Bernstein, to identify a new portfolio of AEAD schemes which should improve upon the current standards, and in particular, proposals must argue why they are better than GCM used with the AES as the underlying block cipher (this combination is denoted AES-GCM). The Competition for Authenticated Encryption: Security, Applicability, and Robustness (CAESAR) started in 2012 and is still ongoing, with the tentative announcement of the final portfolio set to the end of 2017.

In the following, we formally introduce the concept of authenticated encryption with associated data, and we continue to describe various construction mechanisms to obtain such schemes.

Definition 4 (Authenticated encryption with associated data). *Let κ, η and τ be positive integers. An* authenticated encryption scheme with associated data (AEAD scheme) *is a tuple $\Pi = (\mathscr{K}, \mathscr{E}, \mathscr{D})$ consisting of three algorithms:*

1. *$\mathscr{K}$ is the* key derivation algorithm, *which does not take any input, but returns a key $K \in \mathbb{F}_2^\kappa$ to be used by both $\mathscr{E}$ and $\mathscr{D}$.*

2. *$\mathscr{E}$ is the* encryption function*, and is specified by*

$$\begin{aligned}\mathscr{E} : \mathbb{F}_2^\kappa \times \mathbb{F}_2^\eta \times \mathbb{F}_2^* \times \mathbb{F}_2^* &\to \mathbb{F}_2^* \times \mathbb{F}_2^\tau \\ (K, N, A, M) &\mapsto (C, T),\end{aligned} \tag{1.11}$$

where K is the secret key, N is an η-bit nonce*, A is associated data, which is to be authenticated but not encrypted, and M is the message, the two latter both of arbitrary length. The encryption function outputs a ciphertext C of arbitrary length together with a τ-bit* tag *T, which can be thought of as a MAC on the associated data and message. The nonce is a value which, for a fixed key K,* must not be repeated.

3. *$\mathscr{D}$ is the* decryption function*, and is specified by*

$$\begin{aligned}\mathscr{D} : \mathbb{F}_2^\kappa \times \mathbb{F}_2^\eta \times \mathbb{F}_2^* \times \mathbb{F}_2^* \times \mathbb{F}_2^\tau &\to \mathbb{F}_2^* \cup \{\bot\} \\ (K, N, A, C, T) &\mapsto \begin{cases} M & , T = T' \\ \bot & , T \neq T'. \end{cases}\end{aligned} \tag{1.12}$$

Here, K, N, A, C, T and M are as before, and T' is the verification tag *computed by the decryption function. The symbol $\bot$ is used to indicate decryption failure. This symbol is returned when the verification tag T' computed by the decryption function* does not match *the tag T which was input to the function.*

Note 1. *While Definition 4 specifies $\mathscr{K}$ as an* algorithm*, we will sometimes abuse notation and use $\mathscr{K}$ as the set of possible keys produced by the key derivation algorithm. For instance, we may write $K \in \mathscr{K}$ to mean some derived key, or $K \xleftarrow{\$} \mathscr{K}$ to mean a randomly sampled key.*

When Alice and Bob use an AEAD scheme to communicate, they are presumed to have already exchanged a secret key $K \in \mathbb{F}_2^\kappa$. Now, Alice computes $(C, T) = \mathscr{E}_K(N, A, M)$ for her choices of nonce, associated data and message. She sends the tuple (N, A, C, T) to Bob, who uses his own copy of K to decrypt and obtain $M = \mathscr{D}_K(N, A, C, T)$. When the AEAD scheme Π is secure, it provides the guarantee that Eve is not able to learn the value of M (i.e. the scheme achieves confidentiality), nor can she make modifications to the tuple that will not be detected by Bob on the receiving end (i.e. the scheme provides integrity and authenticity).

Generic Composition

We already mentioned generic composition as a method to construct authenticated encryption schemes by combining an encryption algorithm with an authentication algorithm. The most typical choices of primitives fall on a block cipher (in a particular mode of operation) for confidentiality, and a MAC for providing authenticity and integrity. In the following, we describe three common approaches to generic composition.

Recall that both encryption and MAC computation require secret keys. As such, we assume in the following not one but *two* (independent) keys: a key K_0 for the encryption algorithm and a key K_1 for the MAC algorithm. We assume an encryption function $\mathscr{E}_{K_0} : \mathbb{F}_2^* \to \mathbb{F}_2^*$ and a MAC function $\mathsf{MAC}_{K_1} : \mathbb{F}_2^* \to \mathbb{F}_2^\tau$. We remark that all of the generic compositions described below originally do not allow for associated data. However, there are straightforward extensions that turn them into AEAD schemes, and these are the ones we describe.

Encrypt-then-MAC (EtM). With EtM, Alice first encrypts her message to obtain the ciphertext. She then computes the tag as a MAC on the associated data and ciphertext, which is sent to Bob along with associated data and the ciphertext. On the receiving end, Bob first computes his tag as a MAC on the associated data and ciphertext and compares it to the received tag. If the two agree, he proceeds to decrypt the ciphertext to obtain the message. As such, in the encrypting direction, Alice computes and sends

$$A\|C\|T = A\|\mathscr{E}_{K_0}(M)\|\mathsf{MAC}_{K_1}(A\|\mathscr{E}_{K_0}(M)). \tag{1.13}$$

Bob computes

$$T' = \mathsf{MAC}_{K_1}(A\|C), \tag{1.14}$$

and if $T' = T$, he proceeds to compute

$$M = \mathscr{D}_{K_0}(C), \tag{1.15}$$

and finally obtains the message.

Encrypt-then-MAC has found use in IPsec [182], called *encrypt-then-authenticate*, and has been adopted into the ISO/IEC 19772:2009 standard [130]. In 2014, Gutmann described in RFC 7366 [153] a way to negotiate EtM in TLS 1.2 [114] and DTLS [1], in order to overcome recently identified shortcomings of the MAC-then-Encrypt construction (see below).

From a theoretical standpoint, Bellare and Namprempre showed in [44] that EtM is the only one of the three compositions we describe, which satisfies all the conditions of a secure AE scheme. From the standpoint of efficiency, EtM has the nice feature that the tag can be verified *before* decryption takes place. This is possible, because the tag is computed as a function of the ciphertext (and associated data) directly, rather than the message. This means that forgery attempts by an adversary can be discarded rapidly, without having to spend computational effort on decryption before the tag can be verified. This highly desirable property is not shared by the two next constructions, as we shall see.

MAC-then-Encrypt (MtE). With MtE, a tag is first computed as a MAC on the associated data and message. The message *and* the tag are then encrypted, to obtain the resulting ciphertext which is sent to the receiver together with the associated data. On the receiving end, Bob decrypts and obtains the message and the original tag, and then computes his own version of the tag as a MAC on the associated data and message, and compares the two. As such, the MtE construction operations in the encrypting direction by Alice computing and sending

$$A\|C = A\|\mathscr{E}_{K_0}(M\|\mathsf{MAC}_{K_1}(A\|M)). \tag{1.16}$$

Bob computes

$$M'\|T = \mathscr{D}_{K_0}(C) \quad \text{and} \quad T' = \mathsf{MAC}_{K_1}(A\|M'), \tag{1.17}$$

and accepts the message if $T' = T$ and rejects it otherwise.

MtE has found its widespread use in SSL [137], TLS [114], and DTLS [1]. In [195], Krawczyk shows (by example) that while MtE is *not* generally secure, the construction is secure when the encryption is either a block cipher in CBC mode (with a secure underlying block cipher) or a stream cipher with a pseudorandom pad. Unfortunately, due to the way messages are padded in SSL, this statement turned out in 2014 not to be true for CBC mode encryption [194].

When considering MtE, there are other major drawbacks. For example, and in contrast to EtM above, the tag is not a function of the ciphertext, and as such one can not reject a wrong tag until the whole message has been decrypted. This means that under circumstances where many tags are wrong, a lot of effort is spent on decryption that effectively is useless. There is a potential that this can be leveraged to perform *Denial of Service* (DoS) attacks, in which a server fails to be able to function due to an overload of requests. More critically, the fact that the tag, i.e. the output of the MAC, is part of the ciphertext itself, has shown to be exploitable in what is known as *padding oracle attacks*, when encryption uses a block cipher in CBC mode. The original attack from 2002 is due to Vaudenay [292], and the consequence is essentially that an adversary can decrypt entire ciphertexts, under the assumption that the server leaks information about whether or not decrypted message was padded correctly. More recent examples of padding oracle attacks include the *Lucky Thirteen attack* [14] by AlFardan and Paterson in 2013, which used information about code executing time (a so-called *timing side-channel*) as an oracle and used this to exploit TLS and DTLS. In 2014, Möller, Duong, and Kotowicz found a padding oracle attack called POODLE [240] on SSL 3.0. While most servers no longer use SSL 3.0, it is possible that an attacker can force both the browser and the server to agree on using SSL 3.0, through what is known as a *downgrade attack*.

Encrypt-and-MAC (E&M). When using E&M, the encryption and tag generation are completely separate in the sense that Alice computes the ciphertext as the encryption of the message and then computes the tag as a MAC on the associated data and message, and finally sends the associated data, ciphertext and tag to Bob. On the receiving end, Bob decrypts the ciphertext to obtain the message, and then computes his own tag from the associated data and message, and accepts the message if the two tags agree. As such, in the encrypting direction, Alice computes and sends

$$A\|C\|T = A\|\mathscr{E}_{K_0}(M)\|\mathsf{MAC}_{K_1}(A\|M). \tag{1.18}$$

Bob, on the receiving end, computes

$$M = \mathscr{D}_{K_0}(C) \quad \text{and} \quad T' = \mathsf{MAC}_{K_1}(A\|M), \tag{1.19}$$

and accepts the message *if and only if* $T' = T$.

Like the two other constructions covered, also E&M has found its way into widespread use though the SSH protocol for encrypted text-based shell sessions on remote machines over a network. In their 2002 paper [46], Bellare, Kohno, and Namprempre point out security issues with the SSH Binary Packet Protocol (BPP) from a standpoint of provable security, and also give suggestions for fixes of SSH implementations. Like MAC-then-Encrypt, E&M has the same property that the ciphertext is not protected by the MAC, and like MtE, the tag can only be verified after decryption.

Why Generic Composition? One might ask why it would ever be desirable to use generic composition for AEAD. One obvious upshot to the approach is, that one can essentially, independently, pick the encryption function and the MAC algorithm. The security of the composition of the two relies on the security of either; no ad hoc security analysis of the composed scheme is necessary. Another point is that one can choose components which have excellent performance on the target platform.

With that said, we have clearly seen issues arising with generic compositions for authenticated encryption, some of which we have pointed out above. As the idea of providing data integrity and message authentication dates back to the 70s, generic composition is a very old idea (in a cryptographic timeline). The philosophy has been, already for many years, that integrity and authentication are absolutely necessary, and as such should be designed as part of the algorithm to begin with. We have come a long way with the aforementioned GCM and CCM modes of operation (which we describe in detail below), but more work is actively being done in the area, most prominently with the ongoing CAESAR competition.

Nonce Misuse Resistance and Robustness

Correct use of an authenticated encryption scheme dictates that, for a fixed key K, a particular nonce N is only used for a single query. In other words, a (K,N) pair must never repeat in an encryption query. While this sounds simple enough, the practicality of implementing an actual nonce can be quite difficult. For example, a hardware device may keep a counter variable initialized to zero, which is incremented for each message encrypted, and use this as the nonce. Then, if the key is changed, the counter can be reset to zero. However, it is not unthinkable that such a device is manufactured with a single, static key K to be used during its operational lifetime. If an attacker is able to force the device to restart, she can reset its counter, thereby make nonces repeat for the same key.

Arguably, the assumption of a nonce in an AE scheme is somewhat convenient for the cryptographer, but much less so for the implementer who actually has to make sure not to falsify the nonce assumption. To that end, some AE schemes attempt to achieve some degree of *nonce misuse resistance* (NMR), i.e. to maintain some security guarantees even in the case where a (K,N) pair is used more than once. The question of what exactly the term *nonce misuse resistance* should be taken to mean forms the basis of a quite heated debate in the cryptographic community at the time of writing. Already with our commitment to this term, we risk stepping on a few toes. At the root of the argument lies the fact that a plethora of AE schemes exist, all with various features and properties that make them suitable for many different applications. As such, various degrees of failure, in the event of nonce misuse, exist in different schemes. We highlight that we use the term not as a binary property, but rather to mean that a scheme can achieve some *level* of nonce misuse resistance.

While we will not go into the exact details of the security claims with respect to different levels of nonce misuse resistance, we will describe them informally to the best of our ability in the following. As a reference for this discussion, we use the recent work of Hoang, Reyhanitabar, Rogaway, and Vizár [158].

First and foremost, the concept of *nonce-based authenticated encryption* (NAE) was defined by Rogaway in [269]. We use NAE to denote the property that the nonce assumption *must be maintained*: under nonce misuse, all security guarantees are forfeit. In [271], Rogaway and Shrimpton give the definition of *misuse resistant authenticated encryption* (MRAE). Their idea is that an AE scheme should do the best it can with the nonce it is given: nonces should not repeat for a fixed key, but *if it does*, it should do as good as possible in such a circumstance. The MRAE notion promises that, in the case of nonce misuse, the authenticity of the scheme remains, while privacy is compromised in the sense that an adversary can detect the repetition of an (A, M) pair if such pair has previously been used with the particular nonce [271, p. 383].

The concept of an *online cipher* is first described by Bellare, Boldyreva, Knudsen, and Namprempre in [45]. The idea is to have a cipher which requires only constant memory, so it does not depend on neither the length of message, associated data nor ciphertext. In particular, message blocks $M_1, \ldots, M_\ell$ should be processed left-to-right, outputting ciphertext blocks along the way, such that ciphertext block C_i, $1 \leq i \leq \ell$, depends only on message blocks $M_1, \ldots, M_i$. In [128], Fleischmann, Forler, and Lucks build upon this idea for an AE scheme, and define a corresponding security notion *online authenticated encryption*, which we denote OAE1[2] in line with [158]. Returning to the aforementioned debate on the meaning of nonce misuse resistance, a point of critique on the OAE1 notion is the somewhat blurred understanding of exactly what the notion is supposed to mean. This is reflected by the different variants given in [158, Section 8]: OAE1 and OAE1a through OAE1d. In this thesis, we consider OAE1a as equivalent to OAE1, as the only difference is the primitive underlying the mode: one is a block cipher, the other a permutation. In the event of nonce misuse, the promise made by OAE1 is, that if an adversary observes the result of encrypting messages $M = X \| Y$ and $M' = X \| Y'$, where $Y \neq Y'$, then the fact that M and M' are equal on the first $|X|$ bits is revealed. As such, we remark that the OAE1 notion lies between the NAE notion and the MRAE notion, with respect to how robust the scheme is under nonce misuse. While [158] discusses other notions of security, the concepts of MRAE, NAE and OAE1 suffice for our treatment of nonce misuse resistance.

Among the authenticated encryption schemes considered in this thesis, we will not be dealing with any schemes offering MRAE security. Under the OAE1 banner, we describe below the schemes COPA and APE as well as our own proposals PRØST-COPA and PRØST-APE (based on the two former modes), which we introduce in Section 3.2. In the NAE category, we describe GCM, CCM, OCB and OTR below. Furthermore, another of our own proposals, PRØST-OTR, is introduced in Section 3.2 as well.

So far, we have discussed the robustness of an authenticated encryption scheme under nonce misuse. However, other types of robustness are worth discussing as well. An AE scheme Π should be implemented such that, if $\mathscr{D}_K$ returns $\perp$, either due to tampering with the associated data, ciphertext or tag, or due to erroneous reception, the implementation must safely discard *all temporary values* related to decryption. In particular, if the verification tag T' does not match the presented tag T, the implementation must not leak any computed message bits. This topic is being treated formally under the notion of *release of unverified plaintext* (RUP). While we do not go into the robustness in the RUP scenario in this thesis, we remark that the topic is seeing an increasing amount of attention at the time of writing, see e.g. [6, 25, 136].

Block Cipher Modes of Operation

The AEAD block cipher modes allow to choose an arbitrary block cipher and use it for authencated encryption with associated data. We describe in the following some of the most well-known

[2]Originally called OAE in [128] but renamed OAE1 in [158] due to the correction of an error in the security definition

AEAD block cipher modes, starting with the aforementioned GCM and CCM. Besides GCM and CCM, we describe three modes: OCB, OTR and COPA. The treatment of the two latter modes will be useful when we discuss our CAESAR competition proposal PRØST in Section 3.2.

Note that, unlike generic composition, the AEAD modes described here have authenticity and integrity built in to begin with. This also means that, opposed to the generic compositions above, the schemes require a *single* key $K \in \mathbb{F}_2^\kappa$, which we assume was provided by the key derivation algorithm $\mathscr{K}$. They use a single block cipher, which we denote $\mathscr{E} : \mathbb{F}_2^\kappa \times \mathbb{F}_2^n \to \mathbb{F}_2^n$. We assume associated data blocks $A_1, \ldots, A_k$ and message blocks $M_1, \ldots, M_\ell$.

Galois/Counter Mode (GCM). GCM is a design by McGrew and Viega [224]. At the time of writing, it is the most commonly used AEAD block cipher mode of operation, and is, in almost all cases, used with AES-128 as the underlying block cipher. It is *one-pass*, meaning it iterates over the data once (as opposed to e.g. CCM below), and offers no security in the event of nonce misuse, i.e. it falls into the NAE category. GCM is used in a wide range of applications including TLS 1.2 [114, 278] and NSA Suite B Cryptography [8]. Furthermore, it has been standardized by NIST in Special Publication 800-38D in 2007 [121].

Besides a block cipher, GCM uses multiplication by elements of the finite field $\mathbb{F}_{2^n}$, specified by a particular irreducible polynomial. Encryption essentially works in CTR mode, so GCM maintains a counter variable J which is initialized based on the nonce N. For every message block, the counter J is increased, and each ciphertext block is obtained as $C_i = \mathscr{E}_K(J_i) \oplus M_i$ for $i = 1, \ldots, \ell$. For authentication, one first computes $H = \mathscr{E}_K(0^n)$. In a procedure called GHASH, a tag is computed over the associated data and ciphertext by adding each block to a running variable X, which is then updated by multiplication over $\mathbb{F}_{2^n}$ by the constant H. Finally, the authentication tag is computed as $T = H \oplus \mathscr{E}_K(X \oplus (|A| \| |C|))$, thus making the tag depend also on the length of associated data and ciphertext.

An appealing part of AES-GCM is that implementations in high-end general-purpose CPUs can achieve very high performance. This is mainly due to the hardware acceleration on Intel platforms since their *Westmere* microarchitecture launched in 2010, which introduced *AES New Instructions* (AES-NI) [146]. These are very fast instructions for encryption using AES. Furthermore, the `pclmulqdq` instruction allows for highly optimized implementations of the finite field multiplication used by GCM. In fact, these hardware optimizations mean that other block cipher modes for AEAD, as we shall see, also recommend use with the AES, and also employ multiplication in $\mathbb{F}_{2^n}$. We go deeper in the discussion of fast implementations, also for GCM, in Chapter 4. While GCM using the AES as the underlying block cipher has really nice implementations on high-end CPUs, it is in general very complicated to implement on platforms not offering special hardware acceleration instructions. This is especially true when the implementation must be *constant-time*, i.e. it should not leak any timing information. For a discussion on this topic, see e.g. the attack by Bernstein [49] and resistant implementations by Käsper and Schwabe [198].

Given its popularity and widespread use, there are several cryptanalytic results on GCM that we should mention at this point. In the line of weaknesses arising due to the polynomial hashing in GCM, i.e. the GHASH function, the first observation by Joux [176] shows that one can recover the hashing key H by using pairs of messages authenticated using the same nonce N.

Since GCM was not designed to be secure under nonce misuse, this is referred to as the *forbidden attack*. In [125], Ferguson showed that when encrypting messages of 2^ℓ blocks, GCM provides in fact only $\tau - \ell$ bits of authentication security, where τ is the tag size. Handschuh and Preneel propose in [154] a method to recover the hashing key in polynomial hashing (including GHASH). Furthermore, they formalize weak keys for GCM. In [275], Saarinen uses a similar concept of weak keys which lead to a hashing key H of low order, to provide cycling attacks on GHASH. This concept was taken further and formalized by Procter and Cid in [263]. Meanwhile, most of the results described were not constructive, in the sense that they show only the *existence* of weak keys and polynomials allowing forgeries. This issue was addressed in 2015 by Abdelraheem, Beelen, Bogdanov, and Tischhauser [3], providing the first universal forgery attacks on GCM that does not require the misuse of nonces.

Concerning the security proofs for GCM, Iwata, Ohashi, and Minematsu revisited the security proofs for GCM in 2012 [167]. They pointed out flaws in the original security proofs for GCM, but also provided means to fix them. Also, they show that GCM has better security bounds when the nonce length is $\eta = 96$ bits.

On the implementation side, a bug in OpenSSL pointed out by Gueron and Krasnov [149] which allowed message forgeries was thankfully identified in time before making it into production code.

Offset Code Book (OCB). OCB is a one-pass AEAD mode supporting a block size of $n = 128$ bits. Again, it falls in the NAE category, so it is not secure in the event of nonce misuse. It is based on the *xor-encrypt-xor* (XEX) tweakable block cipher construction by Rogaway [270]. It exists in three versions: OCB1 by Rogaway, Bellare, Black, and Krovetz [273]; OCB2 by Rogaway [270]; and OCB3 by Krovetz and Rogaway [196]. We remark that throughout this thesis, we take OCB to mean the latest version, OCB3. The AEAD mode AES-OCB (OCB3 with AES-128 as the underlying block cipher) is, at the time of writing, a second-round candidate in the CAESAR competition. AES-OCB implemented on high-end CPUs supporting AES-NI, obtains a performance even better than AES-GCM. This can be accredited to the fact that OCB is parallelizable in *both* encryption and authentication, whereas GCM is parallelizable in encryption (due to the CTR mode of operation) but sequential in the GHASH part. Furthermore, OCB has extremely low overhead, which means it performs well even for short messages. We discuss the performance of OCB in greater detail in Chapter 4, where we also elaborate on how short and long messages impact performance.

OCB is a patented design, but in 2013 the licensing was relaxed, allowing now use in open-source software, general use in non-military software and also including a particular license for OpenSSL [134], see [268].

Ferguson presented in [124] a collision attack on OCB1 which implies that the data encrypted under a single key should be limited to 2^{32} blocks, or about 68 GB. It is debatable whether this is much of a limitation in practice.

Counter with CBC-MAC (CCM). CCM is a design by Whiting, Housley, and Ferguson [120, 296], and is part of the IEEE 802.11i standard [163] (as a variant called CCMP for use with WPA2), IPsec [160] and TLS 1.2 [114, 222]. The motivation behind CCM was to provide a candidate for IEEE 802.11i which, unlike OCB, was not encumbered by patent licensing issues.

As indicated by the name, it is a combination of CTR mode encryption and CBC-MAC for authentication, combined in a MAC-then-Encrypt manner. Like GCM, it is in the NAE category, so security fails under nonce misuse. Contrary to GCM, it is two-pass, meaning a pass is first made over the message to authenticate and then again over the message and tag to encrypt, as is characteristic for the MtE approach. Note however, that unlike the MtE generic composition, CCM uses a *single* key K for both CTR mode encryption and the CBC-MAC. This is secure, provided the same nonce (which is used as the initial counter for CTR mode encryption) is not repeated for the same key K.

Being two-pass, it is clear that CCM lacks performance in comparison to both GCM and OCB. Besides this obvious deficit, other issues concerning CCM are discussed by Rogaway and Wagner in [272]. As an example critique, the authors point out that CCM is not online (this is a consequence of being two-pass). Other problems concerning more subtle details such as byte orientation, parameters and nonce lengths are also discussed. As an alternative to CCM, Bellare, Rogaway, and Wagner propose the EAX mode of operation in [47].

Offset Two-Round (OTR). The OTR mode is a design by Minematsu [232]. It is, like OCB, at the time of writing a second-round CAESAR competition candidate. As already mentioned, OTR too is in the NAE category of modes, not giving any security guarantees under nonce misuse. Encryption works by considering two consecutive message blocks as a unit, and these are encrypted in a two-round Feistel cipher (hence the name), in which the round function is composed of a block cipher call combined with a masking, thus following the *xor-encrypt* (XE) tweakable block cipher construction by Rogaway [270]. As such, each pair of two consecutive message blocks are encrypted completely independently from any other message blocks, implying that the scheme is easily parallelizable. In OTR, associated data is processed in a PMAC-like fashion to generate a separate tag for associated data. The final authentication tag is computed using an XEX construction on a checksum Σ computed over every second message block, combined with masking by the associated data tag and some additional masking. It makes a single block cipher call per block of associated data and message, and as such is a one-pass scheme, like GCM and OCB. An interesting feature of OTR is, that due to the Feistel structure, it does not need to implement the block cipher decryption function to obtain the AEAD scheme decryption function. This is a property not found in OCB, but also found in GCM and CCM, which employ encryption in CTR mode.

Due to the parallizable structure of OTR, it is able to obtain a performance nearly as high as OCB on high-end CPUs. Again, we refer to Chapter 4 for the details and performance figures. The OTR mode, together with COPA (described next), form two out of three of the underlying modes for our CAESAR proposal PRØST. As such, we shed more light on those two modes, and describe them in greater detail, when we cover the PRØST proposal in Section 3.2.

COPA. Finally, COPA is an AEAD block cipher mode of operation by Andreeva, Bogdanov, Luykx, Mennink, Tischhauser, and Yasuda [21]. COPA makes two block cipher calls per block of message, and can be seen as the composition of two XEX tweakable block ciphers. Indeed this formulation is used for the security proof in [21]. Also, COPA is an online scheme, in the sense that ciphertext blocks can be released even though all message blocks have not yet been encrypted. Like OTR, the way that COPA processes associated data is very much like the PMAC construction. COPA too, when combined with AES-128, is at the time of writing a second-round candidate in the CAESAR competition, under the name AES-COPA. When nonces are misused, COPA offers the OAE1 notion of security, thus leaking equality of block-aligned prefixes for $M = X\|Y$ and $M' = X\|Y'$.

COPA has received attention from the cryptographic community which has resulted in a couple of observations. In the COPA paper, the success probability for a forgery is shown to be 2^{-n}, where n is the block size of the cipher. In [214], Lu presents universal forgery attacks nearly meeting the birthday bound, i.e. with a probability slightly larger than the claimed 2^{-n}. In [242], Nandi gives a forgery attack of about $2^{n/3}$ queries, breaking the security claims of the design. He also revisits the security proofs and gives a new proof of integrity meeting that same bound.

Sponge Constructions

In Section 1.2.5 below, we describe *cryptographic hash functions* which is an unkeyed primitive for providing a fixed-length fingerprint of a variable-length input. Two very common and NIST-standardized hash functions are Secure Hash Algorithm-1 (SHA-1) and the SHA-2 family, both defined in [253]. The latter being from 2001, NIST found in 2007 that SHA-2 needed replacing by a new standard to be called SHA-3, which was to be identified through a public competition, much similar to the AES competition.

During the SHA-3 competition [243], a novel design approach to symmetric primitives saw the light of day. So far, the constructions we have seen have all been based on block ciphers, but with KECCAK [51], the eventual winner of the SHA-3 competition, the designers used a new approach based on a fixed size permutation in their *sponge* constructions. Unlike block ciphers, these constructions do not require designing a key schedule algorithm, and can easily consume arbitrary length inputs and provide arbitrary length outputs. As such, the sponge constructions are very versatile, and can be used to define block ciphers, stream ciphers, hash functions (as with KECCAK), and also AEAD schemes, as we describe here. With KECCAK, permutations have become tremendously popular building blocks for symmetric schemes in general, many of which more or less resemble the sponge construction, which we describe next.

Conventionally, a sponge is a construction which consists of two phases: the *absorption phase* and the *squeezing phase*. During absorption, all the input is mixed into the sponge state over several rounds. During squeezing, bits of the state are extracted over several rounds, until the required amount of output has been obtained. In order to allow for continuous absorption and squeezing, a *duplex mode* for the sponge construction has been defined. Here, essentially one block is consumed and one block is squeezed, per round. This is useful for AEAD schemes, as this allows them to be online. In the following, we take a closer look at a sponge construction in duplex mode, when we describe the APE scheme for AEAD. As was the case with OTR and COPA, one of the PRØST proposals for the CAESAR competition described in Section 3.2 uses APE as a mode of operation for a permutation to obtain an AEAD scheme, which is why we introduce it here.

Duplex Mode. The primary component of a sponge construction (in duplex mode) is a permutation P specified by

$$P : \mathbb{F}_2^n \to \mathbb{F}_2^n. \tag{1.20}$$

Furthermore, we need a *padding function*

$$\mathsf{pad}_r : \mathbb{F}_2^k \to \mathbb{F}_2^r, \quad 0 \le k < r, \tag{1.21}$$

which maps inputs of *strictly less* than r bits to outputs of *exactly* r bits. There are some requirements that pad_r must satisfy in order to be secure, and in particular it must be *sponge-compliant* (see [52, Definition 1]). We remark at this point that the padding function used for our PRØST proposal for AEAD, which we introduce in Section 3.2, is sponge-compliant, but has a slightly different signature than that of Eq. (1.21).

We refer to an n-bit input to- and output from P as a *state*, denoted by S. During operation, the state is partitioned into two parts: the *rate* part of the most significant r bits, and the *capacity* part of the least significant c bits, and as such, $n = r + c$. The rate part of the state is the only part which is ever directly modified (when absorbing) or exposed (during squeezing), at any time during operation. Informally, this means that one can tweak the security of the construction depending on how one chooses r and c, and particularly the ratio between them. A round of the duplex construction consists of an application of P to the state. For an n-bit value X, we use X_r to refer to the rate part and X_c to refer to the capacity part.

For an input to the duplexed sponge X and its corresponding output Y, we write

$$X = X_1 \| \cdots \| X_\ell \quad \text{and} \quad Y = Y_1 \| \cdots \| Y_\ell, \tag{1.22}$$

with X_i and Y_i all less than r bits. We initialize the duplexed sponge by setting $S = 0^n$. One now proceeds by *duplexing*, i.e. applying rounds to absorb the X_i and squeeze the Y_i in the order $i = 1, \ldots, \ell$ as

$$S = P((S_r \oplus \mathsf{pad}_r(X_i)) \| S_c) \quad \text{and} \tag{1.23}$$

$$Y_i = \mathsf{lsb}_{l_i}(S_r). \tag{1.24}$$

As such, with Eq. (1.23) we absorb X_i and with Eq. (1.24) we squeeze from the state to obtain Y_i, where l_i is the required bit-length of Y_i. Figure 1.6 shows a general sponge construction in duplex mode.

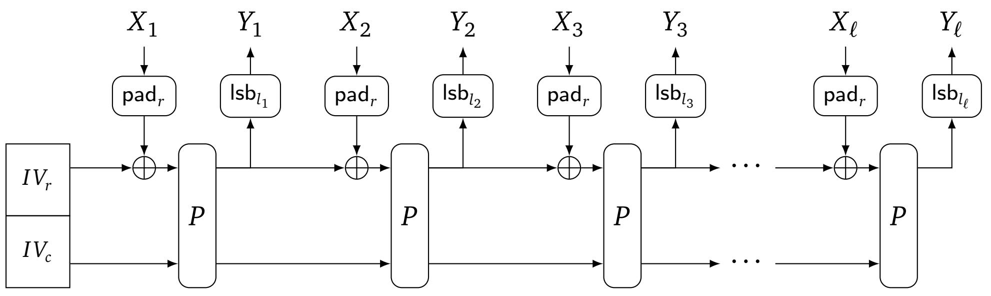


Figure 1.6: General duplex mode sponge construction

APE. The APE scheme is a design by Andreeva, Bilgin, Bogdanov, Luykx, Mennink, Mouha, and Yasuda [24]. It was marketed as "the first permutation-based authenticated encryption scheme that is resistant against nonce misuse". Again, it is debatable how this should be understood. In the notions introduced above, APE offers OAE1 security, just the same as COPA.

APE, as specified in [24], supports *fractional data*, i.e. it can handle cases where $|A|$, $|M|$ or both are not a multiple of r bits. As such, the scheme is in compliance with our notion of an AEAD scheme of Definition 4. However, in the following, we describe APE where $|A|$ and $|M|$ are both assumed to be multiples of r. This allows us to describe APE in a simpler and more elegant way, and in our application of the APE mode in the context of PRØST in Section 3.2, we avoid these complications by always applying padding to the inputs A and M, thus handling fractional data in a much simpler way than having special cases for the mode itself.

APE follows a sponge construction in duplex mode. The size of the secret key K is $\kappa = c$ bits, so it fits in the capacity part of the state. The length of the authentication tag is $\tau = c$ bits. The nonce N is considered to be part of the associated data A. In the terminology of the duplexed sponge, APE processes associated data and message by setting $X = A\|M$, such that $|X_i| = r$ for $i = 1, \dots, k + \ell$. There are, however, two modifications:

1. Instead of initializing S as 0^n, it is initialized as

$$S = 0^r\|K. \tag{1.25}$$

   This change is made to make the state depend on the secret key K to begin with.

2. After processing all associated data, i.e. after absorbing X_k, one computes

$$S = S \oplus (0^{r+c-1}\|1), \tag{1.26}$$

   i.e. the least significant bit of the state is flipped before X_{k+1} is absorbed. This is done to create a so-called *domain separation* between the processing of blocks from A and blocks from M. Without this, an attacker could trivially provide a valid forgery by setting e.g. the last block of A to the first block of M, i.e. by using $A' = A_1, \dots, A_{k-1}$ and $M' = A_k, M_1, \dots, M_\ell$.

The squeezing lengths are set to

$$l_i = \begin{cases} 0 & , 1 \le i \le k \\ r & , k < i \le k + \ell, \end{cases} \tag{1.27}$$

and ciphertext is obtained as

$$C = Y_{k+1}\| \cdots \|Y_{k+\ell}. \tag{1.28}$$

As such, nothing is effectively squeezed during the processing of associated data, while ciphertext blocks are squeezed during the processing of blocks from M. Finally, the authentication tag is obtained as

$$T = S_c^{\text{final}} \oplus K, \tag{1.29}$$

where S^{final} is the output state from which $Y_{k+\ell}$ was obtained. In Section 3.2, we give a pictorial description of APE, when instantiated using the PRØST permutation.

1.2.5 Cryptographic Hash Functions

A commonly used analogy in cryptography is that between *cryptographic hash functions* and the Swiss army knife. Of course, there is a lot of truth in this comparison. Cryptographic hash functions are highly versatile, even more so than block ciphers, and are useful in countless of applications.

While this thesis will not deal with cryptographic hash functions in great detail, we nevertheless give a brief introduction below. In particular, we focus on ways they can be constructed from block ciphers, as this aspect from cryptographic hash function design is of particular interest in relation to our key-less cryptanalysis of the block cipher PRESENT in Section 2.2.

Definition 5 ((Cryptographic) hash function)**.** *Let n be a positive integer. A* (cryptographic) hash function *$\mathscr{H}$ is a function mapping bit-strings of arbitrary length to bit-strings of a fixed length n,*

$$\mathscr{H} : \mathbb{F}_2^* \to \mathbb{F}_2^n. \tag{1.30}$$

Henceforth, we talk simply about *hash functions* with the *cryptographic* prefix implied. The *cryptographic* adjective of Definition 5 implies that the function must satisfy three main security goals, problems which should be hard to solve for a particular function $\mathscr{H}$:

1. **Pre-image resistance**: Given $Y \in \mathbb{F}_2^n$, it should be hard to find $X \in \mathbb{F}_2^*$ such that $Y = \mathscr{H}(X)$,

2. **Second pre-image resistance**: Given $X \in \mathbb{F}_2^*$ and $Y \in \mathbb{F}_2^n$ such that $Y = \mathscr{H}(X)$, it should be hard to find $X' \neq X \in \mathbb{F}_2^*$ such that $Y = \mathscr{H}(X')$, and

3. **Collision resistance**: It should be hard to find $X, X' \in \mathbb{F}_2^*$ with $X \neq X'$ such that $\mathscr{H}(X) = \mathscr{H}(X')$.

One can think of the output of $\mathscr{H}$ as a *fingerprint* or *digest* of the arbitrary-length input. As such, hash functions can be used to provide integrity of data, in the sense that if the input if modified even the slightest, the hash will be significantly different. This is useful in applications where integrity is desired, but authenticity is assumed. For example, if a user downloads a file from a web server, the web server can provide a hash of the file while is assumed to be correct, and the user can compute his own hash of the downloaded file to verify its integrity.

When a hash function does not exhibit any analytical weaknesses with respect to (second) pre-image resistance, the expected effort required to provide a (second) pre-image is $O(2^n)$. However, due to the birthday paradox (see e.g. [192, Chapter 4]), there is a simple algorithm to find collisions for a hash function $\mathscr{H}$ in time $O(2^{n/2})$. One simply computes two lists containing the hash values of randomly chosen inputs. If the lists have size $2^{n/2}$, there is a very high probability that the two lists have at least one element in common, implying a collision. Thus, the digest size n should be large enough that $2^{n/2}$ computations of $\mathscr{H}$ is computationally infeasible. Typical hash sizes are $n = 160$ bits, as is the case for SHA-1 [253], or one of $n \in \{224, 256, 384, 512\}$ bits, which is the case for both the SHA-2 and SHA-3 families [253, 254].

Construction of Hash Functions from Block Ciphers

At this point, we turn our discussion of hash functions to their design, and in particular we describe how they can be constructed from block ciphers. First, we give the definition of *one-way collision-resistant compression functions*.

Definition 6 (One-way collision-resistant compression function). *Let w, n and m be positive integers. A* one-way collision-resistant compression function *is a function*

$$F : \mathbb{F}_2^w \times \mathbb{F}_2^n \to \mathbb{F}_2^m, \tag{1.31}$$

which has the properties that it is easily computable, and is resistant to pre-image attacks, second pre-image attacks and collision attacks.

For simplicity, we leave out the *one-way collision-resistant* prefix for the remainder of this thesis. Compression functions are named so, because typically $w + n > m$, and as such, F compresses the inputs into fewer output bits.

As mentioned, hash functions should be able to consume inputs of arbitrary length and provide an output of a fixed length. Thus, let $M \in \mathbb{F}_2^*$ be one such input and let F be a compression function as specified by Definition 6, where we set $w = m$. We can split M into blocks of n bits each, denoted by $M_1, \ldots, M_\ell$. If the last block M_ℓ has a length $|M_\ell| < n$, we apply a padding function such that $|\mathsf{pad}_n(M_\ell)| = n$. Let $IV \in \mathbb{F}_2^m$ be an initialization vector, as we know from block cipher modes of operation. We can construct a hash function $\mathscr{H}$ from the compression function F by applying what is called the *Merkle-Damgård construction*, an approach first described in the thesis of Merkle [230], and later proven secure in independently by Damgård [109] and Merkle [231]. The Merkle-Damgård construction works as follows. We iteratively apply F to a chaining value which we denote H_i, $1 \le i \le \ell + 1$, together with M_i, to obtain the next value H_{i+1}. We initially set $H_1 = IV$, and each time we compute H_{i+1}, we *absorb* the message block M_i. Finally, we obtain the value $H_{\ell+1}$, which we take as the output of the hash function $\mathscr{H}$. We illustrate the Merkle-Damgård construction in Figure 1.7. The construction is widely used, and indeed is the basis for the hash functions MD5 [266], SHA-1 and SHA-2 [253].

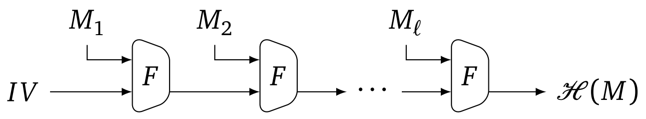
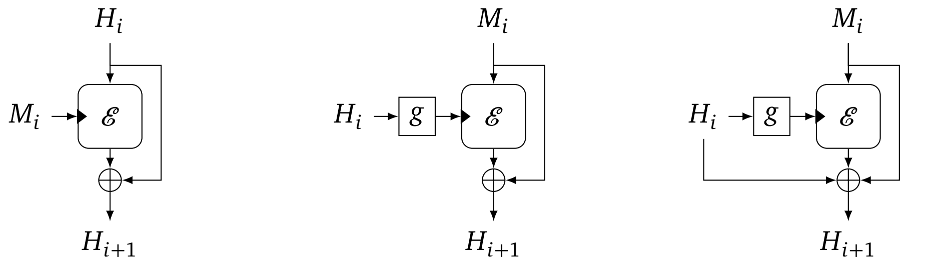


Figure 1.7: The Merkle-Damgård construction

We have now seen how a hash function $\mathscr{H}$ can be constructed from a compression function F, but we have not discussed how to construct a compression function. As it turns out, using a block cipher to construct F is a very popular approach, and we describe three common ways to do it next. An illustration of the three approaches is given in Figure 1.8. For the following three constructions, let $\mathscr{E} : \mathbb{F}_2^\kappa \times \mathbb{F}_2^n \to \mathbb{F}_2^n$ be a block cipher and let $M \in \mathbb{F}_2^*$ be a message to be hashed.

Davies-Meyer (DM). With the *Davies-Meyer* (DM) construction, the message blocks are used as the *key* for the block cipher. Thus, the input message M is split into blocks of κ bits rather than



(a) Davies-Meyer **(b)** Matyas-Meyer-Oseas **(c)** Miyaguchi-Preneel

Figure 1.8: Three common compression function constructions from block ciphers. The black triangle on the block cipher indicates key input.

n bits, i.e. we have $|M_i| = \kappa$. The chaining values H_{i+1} are computed from M_i and the previous value H_i by first encrypting H_i using the block cipher under the key M_i, and adding the result using XOR to H_i itself. As such, we have

$$H_{i+1} = \mathcal{E}_{M_i}(H_i) \oplus H_i, \quad 1 \leq i \leq \ell. \tag{1.32}$$

The final hash $H_{\ell+1}$ has a length equal to the block size n of $\mathcal{E}$. The construction is attributed to Davies by Winternitz in [297, 298] and to Meyer by Davies and Price in [110]. When $\kappa > n$, it was shown by Black, Rogaway, and Shrimpton [66] that it requires approximately 2^n encryptions for a (second) pre-image attack and about $2^{n/2}$ encryptions for a collision attack, for the hash function constructed using DM.

Matyas-Meyer-Oseas (MMO). The *Matyas-Meyer-Oseas* (MMO) construction was proposed in 1985 by Matyas, Meyer, and Oseas [221]. It can be considered the dual to the Davies-Meyer construction, in the sense that the chain values H_i are used as key values and message blocks M_i are used as inputs to the block cipher. As such, we split M into n-bit blocks $M_1, \ldots, M_\ell$, and we compute H_{i+1} from H_i and M_i as

$$H_{i+1} = \mathcal{E}_{g(H_i)}(M_i) \oplus M_i, \quad 1 \leq i \leq \ell. \tag{1.33}$$

Here, $g : \mathbb{F}_2^n \to \mathbb{F}_2^\kappa$ is a function mapping the chaining value from the size of the output block to the size of the key required by the block cipher. If $n = \kappa$, we take g to be the identity function. The MMO construction has been adapted into the ISO/IEC 10118-2 standard [131].

Miyaguchi-Preneel (MP). The *Miyaguchi-Preneel* (MP) construction was proposed independently by Miyaguchi, Iwata, and Ohta [234] and Preneel, Govaerts, and Vandewalle [262] in 1989. The MP construction can be thought of as an extension to MMO, because the only difference is that to compute H_{i+1} we also XOR the previous chaining value H_i. As such, we split the input message M into n-bit blocks $M_1, \ldots, M_\ell$, and we compute H_{i+1} from H_i and M_i as

$$H_{i+1} = \mathcal{E}_{g(H_i)}(M_i) \oplus M_i \oplus H_i, \quad 1 \leq i \leq \ell, \tag{1.34}$$

where, again, $g : \mathbb{F}_2^n \to \mathbb{F}_2^\kappa$ is a function that allows the output of $\mathcal{E}$ to be mapped to a κ-bit value which can be used as key for $\mathcal{E}$.

1.3 Cryptanalysis

In Section 1.1, we introduced the cryptographic goals of confidentiality, integrity and authenticity. The art of *cryptanalysis* essentially evolves around trying to break one or more of those three properties in a cryptographic scheme. This is done by uncovering properties which allow an attacker to achieve some adversarial goal (we consider types of goals below), with a computational complexity below that which would be considered ideal for the primitive in question.

At a very high level, we consider two separate schools of cryptanalysis. First, we have *conventional attacks,* in which the cryptanalyst attempts to identify some particular structure or property of the design, which can be leveraged to accomplish some adversarial goal. Second, *implementation attacks* is a type of cryptanalysis that somehow exploits a particular implementation of a cryptographic primitive. Attacks under the *side-channel* umbrella cover e.g. timing attacks, where one precisely measures the time required to perform specific parts of the algorithm; power analysis attacks, in which the attacker uses measurements of the power consumed by the implementation; or acoustic attacks, where measurements of the sounds emitted by hardware components can be used to determine what is happening in the implementation. Some implementation attacks are more intrusive. For example, one can use specialized and very expensive equipment to modify in a controlled manner what happens during operation. An example would be differential fault analysis, where one can inject a fault into the operation of the primitive in order to obtain (parts of) the internal state. A recommendable introductory book to the topic of power analysis attacks is by Mangard, Oswald, and Popp [216]. In this thesis, we focus on cryptanalysis of the conventional kind. However, we shall briefly discuss *countermeasures* to side-channel cryptanalysis, i.e. steps that can be taken in implementations to avoid such attacks, when we introduce the AEAD scheme PRØST in Section 3.2.

Before we can discuss what is meaningful cryptanalysis, we need to provide a setting for the discussion. In particular, we must define in what ways the adversary can interact with the primitive, and what knowledge she has about the system as a starting point (the adversarial model). We must also define the goals the adversary can try to obtain, and introduce metrics related to the cryptanalytic effort an adversary must make to compromise a system. We cover all of these points in the following.

1.3.1 Adversarial Models

In order to properly assess the severity of an attack on a cryptographic algorithm, we must necessarily consider in what ways the adversary is assumed to be able to interact with (an implementation of) the primitive.

First and foremost, we must assume that the cryptanalyst knows all details of the cryptographic primitive in question, *except* the user-supplied secret key K (when such is present). As already mentioned, this requirement is known as Kerckhoffs' principle, and is a reinterpretation of requirements for a cipher defined in 1883 by Kerckhoffs [179].

Besides Kerckhoffs' principle, we consider five types of adversarial models, listed by the strength of their assumptions (going from weaker to stronger):

1. **Ciphertext-only attack:** The attacker is able to obtain ciphertexts produced by $\mathscr{E}_K$. In this scenario, the attacker must combine this information with some knowledge about the

message being encrypted, in the form of redundancy.

2. **Known-plaintext attack (KPA):** The attacker is capable of obtaining ℓ message/ciphertext pairs (M_i, C_i) such that $C_i = \mathcal{E}_K(M_i)$ for $i = 1, \ldots, \ell$.

3. **Chosen-ciphertext attack (CCA):** In this scenario, the attacker is able to obtain messages $M_1, \ldots, M_\ell$ corresponding to $M_i = \mathcal{D}_K(C_i)$, where the C_i are ciphertexts of her choosing.

4. **Chosen-plaintext attack (CPA):** The CPA attack is the dual to the CCA, in the sense that the attacker can obtain ℓ ciphertexts corresponding to ℓ chosen messages, i.e. she obtains $C_i = \mathcal{E}_K(M_i)$, for $i = 1, \ldots, \ell$, where the M_i are of her choosing.

5. **Adaptive CCA and CPA:** The adaptive version of CCA is like a regular CCA, except the attacker can choose the value of C_j *depending* on the obtained pairs (M_i, C_i) with $1 \leq i < j \leq \ell$. The adaptive CPA is symmetric to the adaptive CCA, but in the encryption direction.

Note that the adversarial models described here can be considered as generalizations of the assumptions described when we discussed the security properties of a MAC in Section 1.2.3. The distinction between the practicality of attacks in the different models listed should be clear. Simply being able to observe ciphertexts, as in the ciphertext-only attack, is a *much more* practical assumption than that of e.g. the CPA. As we move down the list, the control the attacker has over the primitive in question greatly increases. Thus, it is to be expected that this control can be leveraged to obtain more powerful cryptanalytic attacks. These two factors, the adversarial model versus the power of the attack presented, must be weighed against each other, when assessing the severity of the attack.

Separate from the models listed above, we also consider different models as to what the attacker knows about the secret key (if present), before the attack commences. To that end, when describing a cryptanalytic attack, we use one of the models described above in conjunction with a description of what the attacker knows about the key material. The latter can be one of the following:

1. **Secret-key model:** The attacker is assumed to know nothing about the secret key K which is used to provide the security goals for the cryptographic primitive. Doubtless, this is the most common model of assumption on the key in practice, and definitely also the most practical one.

2. **Related-key model:** In the related-key model, an attacker is allowed to analyze the cryptographic primitive under not a single key K, but under a range of different keys $K_1, \ldots, K_m$. As indicated by the name, the keys are related in some sense. We make a distinction between whether the attacker can *choose* this relation herself, or whether she simply *knows* how the keys are related. The key relations themselves can take various forms. For example, keys K_1 and K_2 may be related by $\alpha \in \mathbb{F}_2^\kappa$ by $K_2 = K_1 \oplus \alpha$. Other types of relations could be rotations so e.g. $K_2 = K_1 \lll v$ or arithmetic relations such as $K_2 = K_1 \boxplus v$.

3. **Known-key model:** In the known-key model of attacks, the attacker is assumed to actually know the key K. While this assumption may seem a little odd, as effectively everything has

been given up to the attacker already, there are examples (as we shall see in Section 2.2) where such a model is meaningful. In the known-key model, the adversarial goal necessarily becomes to show some sort of structural weakness in the design, which does not depend directly on the particular key.

4. **Chosen-key model:** Like the known-key model, the attacker also has knowledge of the key K in the chosen-key model. The only difference between the two is, that rather than being given the key, she can freely choose the key herself, when she wants to exhibit the weakness.

5. **(Key-less model):** What we refer to as the *key-less model* is nothing but the union of the known-key and chosen-key models above. In Section 2.2, we present results in the key-less model with particular focus on the standardized block cipher PRESENT. Effectively, cryptanalysis for primitive in the key-less model will attempt to achieve much the same goals as cryptanalysis for a hash function, with it also being a key-less primitive by design. We remark that some literature, e.g. the work by Lamberger, Mendel, Rechberger, Rijmen, and Schläffer [203] refer to this model as the *open-key setting*.

1.3.2 Adversarial Goals

When we consider a cryptanalytic attack, perhaps the most important thing about it, and indeed what concerns the designers and users of the primitive the most, is the *adversarial goal* achieved. From the attackers point of view, she wants to break the primitive as badly as possible. Of course, for a good design this should not be possible. However, there are other goals that she might be able to achieve, which are less severe. To that end, we give a hierarchy of adversarial goals for block ciphers, ranked by severity, as introduced by Knudsen and Robshaw [192]:

1. **Key recovery/total break:** The attacker is able to recover the secret key K for the symmetric primitive in question.

2. **Global deduction:** The attacker is able to determine a function F which is functionally equivalent to either $\mathscr{E}_K$ or $\mathscr{D}_K$, without knowing the secret key K.

3. **Local deduction:** The attacker is able to determine a *single* pair (M, C) s.t. $C = \mathscr{E}_K(M)$, where neither M nor C have previously been observed.

4. **Distinguisher:** The attacker is able to efficiently distinguish between the cryptographic primitive in question, and an ideal version of the same primitive. Using a block cipher as an example, the attacker must be able to decide, given the obtained data, if she is interacting with a specific instantiation of the block cipher, or a random permutation on n bits.

We remark that the goals for other primitives may differ slightly. For example, for stream ciphers, we can consider the goal of recovering the internal state which, for such a primitive certainly is devastating, but not nearly as bad as recovering the secret key. As already hinted, the order above is by severity. This also means, that a key recovery implies all the other goals. In particular,

$$\text{Key recovery} \Rightarrow \text{Global deduction} \Rightarrow \text{Local deduction} \Rightarrow \text{Distinguisher}. \tag{1.35}$$

Thus, a shortcut for a designer to argue for the security of a cryptographic primitive is to show that a distinguishing attack is not possible. If this is true, then obviously none of the other adversarial goals can be achieved. This observation also forms the basis of a topic in cryptography called *provable security*, where constructions using primitives can be shown to be secure, provided that the underlying primitive is indistinguishable from an ideal primitive. The prime example of an application of provable security is block cipher modes of operation. In Section 3.2, when we introduce the PRØST AEAD scheme, we will touch further upon provable security when we show that the constructions are secure assuming that the underlying PRØST permutation is ideal.

1.3.3 Complexity Metrics

Once a cryptanalytic attack has been found on a primitive, the most interesting question is: what is the attack complexity? To compare the attack against a *brute-force attack* (i.e. an attack simply trying all possible key values K, see below) obtaining the same adversarial goal, we need to first answer this question. Indeed, if the attack complexity turns out to be higher than that required by a brute-force approach, we do not consider it a valid attack to begin with. As already hinted, we want to assess the attack on how much data it requires, e.g. in terms of the number of observed ciphertexts or chosen messages. In general, we consider three metrics when evaluating the complexity of a cryptanalytic attack:

1. **Time:** The first thing usually considered for an attack is the time required to perform it. Of course, the time complexity naturally requires an attached unit. Sometimes, if an attack can be practically verified, the unit is universal time. However, cryptanalytic attacks tend to have a theoretically estimated time complexity, and to that end, one uses other units such as basic CPU instructions, or perhaps evaluations of the underlying primitive. It is important to note that time complexity alone does not properly describe the complexity of an attack. For example, it is not unthinkable that an attack may require a much lower time complexity than e.g. the required memory. When analyzing time complexity, we refer to *offline* time complexity and *online* time complexity, respectively. The offline time complexity describes the time required for the attack *before* any message/ciphertext pairs are observed (this is also referred to as the *pre-computation phase*), while the online time complexity refers to the time spent using the obtained data to achieve the cryptanalytic goal (also referred to as the *online phase*). We will see examples of both offline- and online time complexity below, when we describe a generic type of attack called time-memory trade-offs.

2. **Memory:** Like the time complexity, the amount of memory required by an attack is extremely important. One can not precisely say whether a higher time complexity is worse than a higher memory complexity. For example, a time complexity of about 2^{40} operations may be completely practical for a single, high-end desktop machine. Meanwhile, even though 2^{40} bytes of memory, amounting to about 1 TB, is a very reasonable size for a hard drive these days, an attack requiring random read/write access to a data structure of such a size on the disk could render the memory complexity the bottleneck of the attack, even with improving technology such as solid state disks. Typically, the memory complexity is measured in some concise type, such as bytes. Other times, the complexity will occur

without a unit, e.g. as the number of elements in a hash table. In such cases, one can roughly estimate the concise complexity by using e.g. the block size of a block cipher as the size of single element.

3. **Data:** When describing the adversarial models above, we have already touched upon the concept of data complexity. It refers to the data points, i.e. either messages, ciphertexts, or a combination of both, which the attacker has obtained, whether the attack model is ciphertext-only, KPA, CCA, etc. When the data complexity covers all pairs of message and ciphertext for the cryptographic primitive, we say that the attack uses the *full code book*. For example, if an attack requires that all 2^n pairs (M_i, C_i) are available for an n-bit block cipher, it uses the full code book.

1.3.4 Brute-Force Attacks

An attack that *theoretically* will always succeed at finding the secret key for a symmetric primitive is the brute-force attack or *exhaustive search*. Thus, the first item on the primitive designer's checklist is to make sure that such an attack is computationally infeasible.

As the computational power with a fixed monetary cost increases over time, brute-force attacks become increasingly feasible for a fixed key size κ. The estimation due to Moore's law is that the computational power doubles every 18 months. While Moore's law certainly is a valid approach to deciding on a safe key size, a more in-depth study such as the parameter recommendations by the European Network and Information Security (ENISA) [286] is recommendable for designers. For another overview of recommended key sizes, see [70].

It is also worth pointing out that the problem of exhaustive search for the secret key allows straightforward parallelization. Such initiatives have been seen e.g. with the DES Cracker [133] and COPACOBANA [197], which are dedicated hardware for exhaustive key search in the DES block cipher [251].

The *expected* time complexity of a brute-force attack is $2^{\kappa-1}$. The unit corresponds to the time required to initialize the primitive, e.g. running the key schedule for a block cipher or initial clocking for a stream cipher, plus the time required to encrypt a couple of messages, that will be used to check whether the key guess was correct. The data and memory complexities for a brute-force attack are negligible, as only a couple of message/ciphertext pairs are required to verify a potential key guess.

Time-memory Trade-offs

In the following, we describe a special class of brute-force attacks called *time-memory trade-offs*. With this approach, we trade off some of the high time complexity with an increased memory complexity, compared to the standard brute-force approach above. While the brute-force approach has a high time complexity and negligible memory complexity, we can imagine another extreme, where the attacker prepares a *dictionary* containing $(\mathscr{E}_K(M), K)$ for all possible $K \in \mathbb{F}_2^\kappa$. Both the offline time complexity and the memory complexity is $O(2^\kappa)$. Then, when observing $C = \mathscr{E}_K(M)$, the attacker can perform a lookup into the dictionary with an online time complexity of $O(1)$ to determine the correct value of K.

A trade-off between the two extremes, first described by Hellman [155] is possible, which allows an attacker to reduce the time complexity of an exhaustive search by sacrificing memory. Like the dictionary attack, the time-memory trade-off also requires pre-computation, i.e. offline computations. The trade-off works by constructing a table T of pairs from $\mathbb{F}_2^\kappa \times \mathbb{F}_2^\kappa$. Let M denote a fixed, chosen message. The attacker computes s *chains* of key guesses, with t key guesses in each chain. These key guesses are denoted $K_{i,j}$ for $1 \le i \le s$ and $1 \le j \le t$. We start by setting randomly chosen initial guesses $K_{i,1}$ for all chains i. Now, we compute each chain by using the recursion

$$\forall i \in \{1,\dots,s\}, \forall j \in \{2,\dots,t\} : K_{i,j} = F\big(\mathscr{E}(K_{i,j-1}, M)\big), \tag{1.36}$$

where $F : \mathbb{F}_2^n \to \mathbb{F}_2^\kappa$ is a *reduction function*. The table T is composed of the set

$$T = \big\{(K_{i,1}, K_{i,t}) \mid 1 \le i \le s\big\}, \tag{1.37}$$

i.e. we store in the pair consisting of the first and last key guess of each chain. The offline time complexity involved in this pre-computation is $2st$ calls to $\mathscr{E}$ with, presumably, a new key each time, and a memory complexity of $2s\kappa$ bits.

Now, for the online phase of the attack, when the attacker intercepts the ciphertext $C = \mathscr{E}_K(M)$ for a chosen message M, she computes $K' = F(C)$. The key insight is, that if K' appeared as any of the $K_{i,j}$ values in the chains from the offline phase, with $j \ge 2$, she is able to determine a key candidate. To see why this is true, assume that $K_{i,j} = K'$ for some i and $j \ge 2$. Then, by definition,

$$F(\mathscr{E}_K(M)) = F\big(\mathscr{E}(K_{i,j-1}, M)\big), \tag{1.38}$$

and thus $K_{i,j-1}$ is a candidate for the secret key K. Of course, if F is many-to-one, there is a certain risk that Eq. (1.38) holds, but $K_{i,j-1} \ne K$. In such a case, we have what we call a *false alarm*. With this observation, the approach of the attacker is to first check if K' appears anywhere among the $K_{i,t}$ values stored in T. If it does, this means she can compute the key candidate $K_{i,t-1}$ by starting from the beginning of the chain $K_{i,1}$ (which is stored in the table), and compute the chain forwards, just like in the pre-computation, to obtain the candidate. However, if K' did not appear as any of the $K_{i,t}$ values, she will move on to check the penultimate key guess in each chain, $K_{i,t-1}$, $1 \le i \le s$, and see if any of those equal K'. This can be done by checking if $F(\mathscr{E}_{K'}(M))$ appears as any $K_{i,t}$ where $1 \le i \le s$. If so, she can again obtain the key candidates by applying $\mathscr{E}$ now $s-2$ times from the chain starting point. She proceeds like this until, at some key guess in some chain $K_{i,j}$, with $j \ge 2$, she hopefully finds K'.

The details of analyzing the complexity and success probability of the attack is beyond the scope of this thesis. For the details, we refer to e.g. [192, Chapter 5]. We state that both the online time complexity and the memory complexity are in the order of $2^{2\kappa/3}$.

Various extensions and modifications of the original time-memory trade-off have appeared over the years. For example, Rivest introduced the use of *distinguished points*, in which the end-points are required to take a particular form, e.g. that the last significant bits should equal zero [267]. Such points are chosen as the $K_{i,1}$, and the endpoints are obtained not necessarily after t computations, but until a new distinguished point is obtained. The distinguished points technique yields fewer memory accesses in the online phase of the attack, since distinguished points are fewer and further apart. However, the very same fact makes the analysis of the approach harder, since the chains do not all have the same length, contrary to those of Hellman's basic approach. Another refined approach by Oechslin [249] called *rainbow tables* use not the same reduction function F in the chain computations, but rather several functions $F_1, \dots, F_t$, one for each link of the chain. Compared to Hellman's approach, the implications are that the online time complexity is halved, while keeping the same key coverage, pre-computation time and memory complexity [192, Chapter 5].

1.3.5 Differential Cryptanalysis

One of the most powerful tools available to the cryptanalyst is *differential cryptanalysis*. It was introduced by Biham and Shamir in the early 90s as a way of analyzing block ciphers, but much of its appeal lies in the applicability to essentially any symmetric primitive, including stream ciphers, authenticated encryption schemes and hash functions. Biham and Shamir noted that the block cipher DES, designed by IBM, would be vulnerable to differential cryptanalysis with a few modifications. This is particularly interesting, as Coppersmith, one of the co-designers of DES, wrote a paper in 1994 making it evident that IBM were aware of the technique even in 1974.

In a nutshell, the idea of differential cryptanalysis is to consider *pairs* of messages (M, M') and their corresponding ciphertexts (C, C'), and utilize a correlation between how M is related to M' and how C is related to C'. The correlation can be used to define a distinguisher, which in turn can be used to perform a key recovery attack, as we shall see. Our treatment in the following will be somewhat formal, as we introduce the necessary concepts and describe distinguishers, key recovery attacks and variants of differential cryptanalysis. In Section 2.1, we turn the theory to practice when we analyze the recent block cipher SIMON using techniques introduced in this section.

Definition 7 (Difference). *Let n be a positive integer and let $X, X' \in \mathbb{F}_2^n$. We define the* (XOR-) difference *between X and X' as*

$$\alpha = X \oplus X'. \tag{1.39}$$

For a difference α, we refer to the bits of α that equal 1 *as the* active *bits.*

We remark that in some applications of differential cryptanalysis, other notions than the XOR-difference make sense, e.g. one might consider the difference defined by $X \boxminus X'$.

Definition 8 (Differential). *Let n and m be positive integers and let F be a vectorial Boolean function $F : \mathbb{F}_2^n \to \mathbb{F}_2^m$. A* differential *is a pair of differences $(\alpha, \beta) \in \mathbb{F}_2^n \times \mathbb{F}_2^m$ where α is called the* input difference *between two inputs to F, and β is called the* output difference *for the two corresponding outputs from F.*

We also denote by $\alpha \xrightarrow{F} \beta$ a differential (α, β) over F, and we write $\alpha \xrightarrow[X]{F} \beta$ to denote the event $F(X) \oplus F(X \oplus \alpha) = \beta$ for some $X \in \mathbb{F}_2^n$. A pair of inputs (X, X') is said to be a *right pair* if and only if $\alpha \xrightarrow[X]{F} \beta$; otherwise, we refer to the pair as a *wrong pair*. In differential cryptanalysis, the main problem for the attacker is to determine differentials which yield an unexpected correlation. To that end, we need to discuss probabilities for differentials.

Definition 9 (Differential probability). *Let n and m be positive integers. Let $F : \mathbb{F}_2^n \to \mathbb{F}_2^m$ be a vectorial Boolean function, and let $(\alpha, \beta) \in \mathbb{F}_2^n \times \mathbb{F}_2^m$ be a differential. The* differential probability *of (α, β)* over *F, denoted $\mathsf{DP}_F(\alpha, \beta)$, is defined as*

$$\mathsf{DP}_F(\alpha, \beta) = \Pr_X\left[\alpha \xrightarrow[X]{F} \beta\right], \tag{1.40}$$

where the probability is taken over $X \in \mathbb{F}_2^n$.

Definition 10 (Impossible differential). *Let n and m be positive integers. Let $F : \mathbb{F}_2^n \to \mathbb{F}_2^m$ be a vectorial Boolean function, and let $(\alpha, \beta) \in \mathbb{F}_2^n \times \mathbb{F}_2^m$. The pair (α, β) is said to be an* impossible differential *with respect to F* if and only if *$\mathsf{DP}_F(\alpha, \beta) = 0$.*

We also use the notation $\alpha \overset{F}{\not\to} \beta$ to denote an impossible differential (α, β) for F.

Definition 11 (Expected differential probability). *Let κ and n be positive integers. Let $\mathscr{E} : \mathbb{F}_2^\kappa \times \mathbb{F}_2^n \to \mathbb{F}_2^n$ be a block cipher and let $(\alpha, \beta) \in \mathbb{F}_2^n \times \mathbb{F}_2^n$ be a differential. The* expected differential probability *of (α, β) over $\mathscr{E}$, denoted $\mathsf{EDP}_{\mathscr{E}}(\alpha, \beta)$, is defined as*

$$\mathsf{EDP}_{\mathscr{E}}(\alpha, \beta) = 2^{-\kappa} \sum_{K \in \mathbb{F}_2^\kappa} \Pr_X\left[\alpha \xrightarrow[X]{\mathscr{E}_K} \beta\right], \tag{1.41}$$

where the probabilities are taken over $X \in \mathbb{F}_2^n$.

As we already discussed in our introduction of block ciphers, most modern designs apply iteratively a round function which is cryptographically weak, in order to define the full encryption function. In the context of differential cryptanalysis, this allows an adversary to consider also differences in the internal states of the block cipher between rounds. In particular, the cryptanalyst can consider how differences behave over the round function, and extrapolate how the behavior extends to the full encryption function. To that end, we introduce the notion of a *differential characteristic* next.

Definition 12 (Differential characteristic). *Let T and n be positive integers, denote by $F_0, \dots, F_{T-1}$ vectorial Boolean functions*

$$F_t : \mathbb{F}_2^n \to \mathbb{F}_2^n, \quad 0 \le t < T, \tag{1.42}$$

and let $F = F_{T-1} \circ \cdots \circ F_0$. A tuple $(\alpha_0, \dots, \alpha_T) \in (\mathbb{F}_2^n)^{T+1}$ is called a differential characteristic *or* differential trail *for the function F, which we also denote by*

$$\alpha_0 \xrightarrow{F_0} \alpha_1 \xrightarrow{F_1} \cdots \xrightarrow{F_{T-2}} \alpha_{T-1} \xrightarrow{F_{T-1}} \alpha_T. \tag{1.43}$$

Fix a key $K \in \mathbb{F}_2^\kappa$ and let $\mathscr{E}_K$ be a T-round iterated cipher using round keys $K_0, \dots, K_{T-1}$. As such, each $F_t(K_t, \cdot)$, $0 \le t < T$, is an n-bit vectorial Boolean function. We illustrate a T-round differential characteristic for $\mathscr{E}_K$ in Figure 1.9.

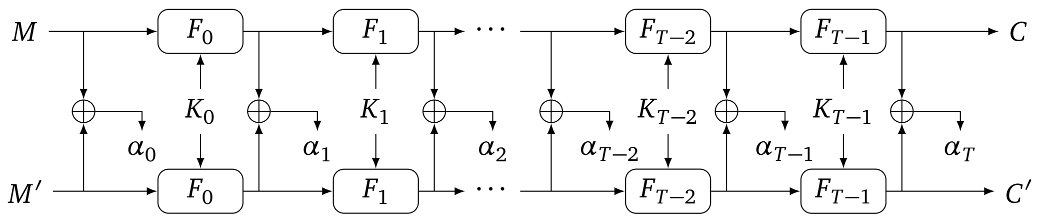


Figure 1.9: A differential characteristic $(\alpha_0, \ldots, \alpha_T)$ for a T-round iterated block cipher

Definition 13 (Differential characteristic probability). *Let T, n and κ be positive integers and denote by $F_0, \ldots, F_{T-1}$ vectorial Boolean functions*

$$F_t : \mathbb{F}_2^n \to \mathbb{F}_2^n, \quad 0 \le t < T, \tag{1.44}$$

and let $F = F_{T-1} \circ \cdots \circ F_0$. Let $(\alpha_0, \ldots, \alpha_r)$ be a T-round differential characteristic over F and let $\tilde{F}_t$ be shorthand for the composition of the first $t+1$ functions, i.e.

$$\tilde{F}_t = F_t \circ \cdots \circ F_0, \quad 0 \le t < T. \tag{1.45}$$

The differential characteristic probability *of $(\alpha_0, \ldots, \alpha_T)$ over F, denoted $\mathsf{DCP}_F(\alpha_0, \ldots, \alpha_T)$, is defined as*

$$\begin{aligned} &\mathsf{DCP}_F(\alpha_0, \ldots, \alpha_T) = \\ \Pr_X\big[\alpha_1 = \tilde{F}_0(X) \oplus \tilde{F}_0(X') \wedge \cdots \wedge \alpha_T &= \tilde{F}_{T-1}(X) \oplus \tilde{F}_{T-1}(X') \mid \alpha_0 = X \oplus X'\big], \end{aligned} \tag{1.46}$$

where the probability is taken over $X \in \mathbb{F}_2^n$. When F is an iterated cipher, the probability is furthermore taken over the key $K \in \mathbb{F}_2^\kappa$.

Definition 14 (Markov chain). *Let $X_1, \ldots, X_k$ be a series of discrete random variables. We say that $X_1, \ldots, X_k$ form a* Markov chain if and only if *for all $i = 1, \ldots, k-1$, it holds that*

$$\Pr\big[X_{i+1} = x_{i+1} \mid X_i = x_i \wedge \cdots \wedge X_1 = x_1\big] = \Pr\big[X_{i+1} = x_{i+1} \mid X_i = x_i\big]. \tag{1.47}$$

In words, a Markov chain has the property that the probability distribution of the random variable X_{i+1} does not depend on the outcome of $X_1, \ldots, X_i$. This property is usually referred to as the memory-less property. *Furthermore, if it holds that for all x, x' in the sample space, the probability*

$$\Pr\big[X_{i+1} = x' \mid X_i = x\big] \tag{1.48}$$

is independent of the index i, then the Markov chain is said to be homogeneous.

Definition 15 (Markov cipher). *Let $\mathscr{E}_K = F_{T-1}(K_{T-1}, \cdot) \circ \cdots \circ F_0(K_0, \cdot)$ be a T-round iterated cipher. We say that $\mathscr{E}_K$ is a* Markov cipher if and only if *for all $t = 0, \ldots, T-1$, it holds that*

$$\mathsf{DCP}_{F_t}(\alpha, \beta) \tag{1.49}$$

is independent of the choice of input X for all $\alpha, \beta \in \mathbb{F}_2^n$ when $K \xleftarrow{\$} \mathbb{F}_2^\kappa$.

If $\mathscr{E}_K$ is a Markov cipher and the round keys are independent and drawn uniformly at random, then the sequence of differences $\tilde{F}_t(M) \oplus \tilde{F}_t(M')$ for $t = 0, \dots, T-1$ form a homogeneous Markov chain [202]. Thus, in a Markov cipher, no matter what particular one-round characteristic we consider, the DCP does not depend on the particular input, assuming the key was chosen uniformly at random. The differential characteristic probability of Definition 13 can be extremely difficult to estimate in modern block ciphers. However, in a Markov cipher where the round keys are independent and generated uniformly at random, the DCP can be computed as a product of the DCP for each round:

$$\mathsf{DCP}_{\mathscr{E}}(\alpha_0, \dots, \alpha_T) = \prod_{t=0}^{T-1} \mathsf{DCP}_{F_t}(\alpha_t, \alpha_{t+1}). \tag{1.50}$$

While modern block ciphers are typically not designed to be Markov ciphers, it turns out that computing the DCP as the product of the single-round characteristic probabilities is, in many cases, a very good approximation [56]. Indeed, the DES and the AES with independent round keys are Markov ciphers [192, Chapter 8]. While this inaccuracy is somewhat unsatisfying from an academic point of view, it is the best tool we have to estimate the DCP of a single differential characteristic for an iterated cipher.

At this point, we note that there is a natural relationship between the concept of a characteristic and that of a differential. In a sense, a differential for an iterated cipher captures a *set* or *collection* of characteristics, in the sense that the differential (α_0, α_T) corresponds to the characteristic $(\alpha_0, \star, \dots, \star, \alpha_T)$, where a $\star$ denotes any value in $\mathbb{F}_2^n$. As such, $\mathsf{DP}_{\mathscr{E}}(\alpha_0, \alpha_T)$ over an iterated cipher $\mathscr{E}$ can be computed in terms of the characteristic probabilities as

$$\mathsf{DP}_{\mathscr{E}}(\alpha_0, \alpha_T) = \sum_{\alpha_1 \in \mathbb{F}_2^n} \sum_{\alpha_2 \in \mathbb{F}_2^n} \cdots \sum_{\alpha_{T-1} \in \mathbb{F}_2^n} \mathsf{DCP}_{\mathscr{E}}(\alpha_0, \alpha_1, \dots, \alpha_{T-1}, \alpha_T), \tag{1.51}$$

which, when $\mathscr{E}$ is a Markov cipher, can be computed as

$$\mathsf{DP}_{\mathscr{E}}(\alpha_0, \alpha_T) = \sum_{\alpha_1 \in \mathbb{F}_2^n} \sum_{\alpha_2 \in \mathbb{F}_2^n} \cdots \sum_{\alpha_{T-1} \in \mathbb{F}_2^n} \prod_{t=0}^{T-1} \mathsf{DCP}_{F_t}(\alpha_t, \alpha_{t+1}). \tag{1.52}$$

In the secret-key model under CPA assumptions, only messages and ciphertexts are available to the attacker. In particular, she has no knowledge about the internal state, i.e. values of the state during encryption. To that end, the differential probability rather than the probability of a single characteristic is what matters to the cryptanalyst. However, even under the assumptions of a Markov cipher, accurately determining the differential probability can be extremely hard. This is in part because of the possible choices of the input mask and output mask (α_0, α_T), but also because it is hard to get a clear view of how many characteristics with the same input/output masks that contribute with a positive probability towards the differential probability.

Distinguishers

While an attacker using differential cryptanalysis usually faces the problem of key-recovery of a single, secret key, the differential probability is taken over all possible keys $K \in \mathbb{F}_2^\kappa$. To overcome this issue, one usually has to make an assumption. We state this assumption in the words of Knudsen and Robshaw [192].

Assumption 1 (Hypothesis of stochastic equivalence)**.** *The* hypothesis of stochastic equivalence *states that for essentially all T-round differentials (α, β) of high probability, it holds that*

$$\Pr_{M}\left[\alpha \xrightarrow[M]{\mathscr{E}_K} \beta\right] \approx \Pr_{M,K'}\left[\alpha \xrightarrow[M]{\mathscr{E}_{K'}} \beta\right] \tag{1.53}$$

is true for a significant fraction of the key values $K \in \mathbb{F}_2^\kappa$, where the probabilities are taken over $M \in \mathbb{F}_2^n$ and $K' \in \mathbb{F}_2^\kappa$.

Informally, the hypothesis states that for a vast majority of the keys, differential probabilities behave as the probabilities taken over all keys.

Consider now a block cipher $\mathscr{E}$, let (α, β) be a T-round differential for $\mathscr{E}$, and let $p = \mathsf{DP}_{\mathscr{E}}(\alpha, \beta)$. For an attacker to provide a distinguisher on $\mathscr{E}$, we require that $p \gg 2^{-n}$. Given that this holds, a very simple approach now lets an adversary distinguish $\mathscr{E}_K$ from an ideal permutation. It works as follows. We consider an *oracle* denoted $\mathscr{O}$ as a function that the attacker can interact with by asking it to provide the output $C = \mathscr{O}(M)$ for some input M. From the beginning, the oracle commits to either returning outputs from an ideal permutation on each input, or it returns the actual encryptions of the inputs obtained using $\mathscr{E}_K$. In the end, the goal of the adversary is to correctly guess which of the two strategies the oracle has committed to. The first step of the adversary is to ask the oracle for outputs on p^{-1} pairs of inputs $(M, M \oplus \alpha)$ and obtains corresponding outputs $(C, C') = (\mathscr{O}(M), \mathscr{O}(M \oplus \alpha))$ from the oracle. If the adversary finds that $C \oplus C' = \beta$ for some pair (C, C'), she guesses that the oracle was interacting with $\mathscr{E}_K$, and otherwise that the output was from an ideal permutation. To see why this approach makes sense, note that in p^{-1} queries, we expect to see one pair (C, C') for which $C \oplus C' = \beta$, as per the probability of the differential. Next, we describe how such a distinguisher can be turned into a key-recovery attack.

Key Recovery

We consider a T-round key-alternating block cipher $\mathscr{E}$, with a secret key $K \in \mathbb{F}_2^\kappa$. We use a $(T-1)$-round differential (α, β) of probability $p = \mathsf{DP}_{\mathscr{E}}(\alpha, \beta)$ and we assume that $\mathscr{E}$ is a Markov cipher. In the following, we describe how to turn a distinguisher on $T - 1$ rounds of $\mathscr{E}$, using the differential (α, β), into a key-recovery attack.

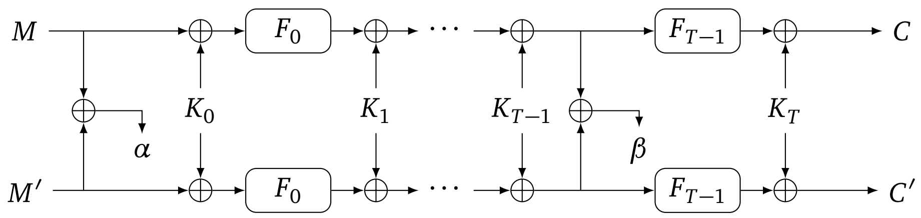


Figure 1.10: Key recovery for a T-round key-alternating block cipher using a $(T-1)$-round differential (α, β)

Consider the illustration of $\mathscr{E}$ with the differential indicated in Figure 1.10. Let us denote by γ the number of possible candidates for the last round key K_T. We assume a CPA scenario where the attacker has obtained N ciphertext pairs (C_i, C'_i), corresponding to the encryption under $\mathscr{E}_K$ of N chosen message pairs (M_i, M'_i), $1 \le i \le N$.

The attacker initializes a list of γ counters denoted $T_0, \dots, T_{\gamma-1}$; one for each possible guess of K_T, and sets them all to zero. Now, the attacker starts by guessing the value of the last round key. Letting $\tilde{K}_T$ denote the guessed value, she iterates over $\tilde{K}_T = 0, \dots, \gamma - 1$. She uses each key guess to decrypt *each ciphertext pair* (C_i, C'_i) for one round, i.e. she computes for $i = 1, \dots, N$ the values

$$X_i = F_{T-1}^{-1}(C_i \oplus \tilde{K}_T) \quad \text{and} \quad X'_i = F_{T-1}^{-1}(C'_i \oplus \tilde{K}_T). \tag{1.54}$$

For each fixed guess $\tilde{K}_T$, she increases the counter $T_{\tilde{K}_T}$ by one each time she observes a pair (X_i, X'_i) where $X_i \oplus X'_i = \beta$, because each such pair corresponds to a right pair (M_i, M'_i) for that particular key guess. This process is repeated for each possible key guess $\tilde{K}_T$.

As the differential is assumed to have probability p for the correct key, the expected counter value for the correct round key K_T is $\mathbb{E}[T_{K_T}] = Np$. If we denote by p' the probability that a given pair is right with respect to the differential (α, β), for a wrong key $K'_T \ne K_T$, then for such a key the expected counter value is $\mathbb{E}[T_{K'_T}] = Np'$. If the attacker found a good differential, then $p \gg p'$, and hence the counter value for the correct round key K_T is expected to be significantly higher than that of the wrong key guesses (and in fact close to Np), and the attacker should be able to distinguish the correct key guess K_T from the wrong ones, based on this difference.

Assuming that the attacker has been able to correctly determine the last round key K_T, she can proceed in exactly the same way to try to recover the penultimate round key K_{T-1} (now using a $(T-2)$-round differential), and so on, until enough round key material has been recovered to reconstruct the master key $K \in \mathbb{F}_2^{\kappa}$ (assuming the key scheduling algorithm is injective, and reminding that the algorithm generally expands few bits to many). The process of recovering the round key K_T, using a $(T-1)$-round differential (α, β), is summarized as Algorithm 1.

Algorithm 1: DIFFERENTIALKEYRECOVERY

```
   Data: Pairs $(M_i, M'_i)$ and $(C_i, C'_i)$ with $1 \le i \le N$, s.t. $M_i \oplus M'_i = \alpha$, $C_i = \mathscr{E}_K(M_i)$ and
         $C'_i = \mathscr{E}_K(M'_i)$ and differential $(\alpha, \beta)$
   Result: Guess $\hat{K}$ for the round key $K_T$
 1 Initialize counters $T_0, \dots, T_{\gamma-1}$ to zero
 2 for $\tilde{K}_T = 0, \dots, \gamma - 1$ do                      // Iterate over key guesses
 3 |   for $i = 1, \dots, N$ do
 4 |   |   if $F_{T-1}^{-1}(C_i \oplus \tilde{K}_{T-1}) \oplus F_{T-1}^{-1}(C'_i \oplus \tilde{K}_{T-1}) = \beta$ then
 5 |   |   |   $T_{\tilde{K}_{T-1}} \leftarrow T_{\tilde{K}_{T-1}} + 1$
 6 |   |   end
 7 |   end
 8 end
 9 $\hat{K} \leftarrow \min_v \{|T_v - Np|\}$
10 return $\hat{K}$
```

Consider a pair of inputs (M, M') used by the attacker, where $M \oplus M' = \alpha$. Of course, the attacker does not know whether this is a right pair or a wrong pair, but we know that *if* it is a right pair, it will suggest the correct key by increasing the corresponding counter, while if it is a wrong pair, it *will not* increase the counter. For wrong pairs, we generally assume that the key guesses $\tilde{K}_T$ for which the counter was increased, are randomly distributed. In particular, we assume that the counters for wrong keys will be sampled from $\mathscr{B}(N, \gamma^{-1})$. This is known as the *hypothesis of wrong-key randomization*. We do not fully justify it in the thesis at hand, but one can think of trying to peel off the last round F_{T-1}, using a wrong key guess, as essentially applying another round of the cipher. For a more detailed discussion on the wrong-key randomization hypothesis, see e.g. [13, 192, 202].

The event that a right pair suggests the correct value for K_T can be thought of as a *signal*. When we identify a pair which suggests a wrong value for K_T, we can think of it as *noise*, because it clouds the separation of the correct key from the wrong ones. In order to establish a successful attack, the signal to noise ratio needs to be diverge sufficiently from 1. A rule of thumb, when using a standard differential attack as outlined above to recover a single round key, is that one needs to use $c/\mathsf{DP}_{\mathscr{E}}(\alpha, \beta)$ message pairs (M_i, M_i') such that $M \oplus M' = \alpha$, where c is a small constant, in order to mount a successful attack. As such, this is the magnitude of the data complexity, and the time complexity is in the same order of magnitude. The memory complexity can be significant, even just for maintaining counters, if the number of candidates γ for K_T is very large. We remark that sometimes, differential attacks can be tailored such that the adversary may get away with recovering the last round key only in parts. This facilitates lowering both the time and memory complexity significantly, while the data complexity remains unchanged, as this depends on the probability of the differential used. With all this said, the analysis presented is based on the assumption that $\mathscr{E}$ is a Markov cipher. In reality, this is almost never the case. Furthermore, we assumed the round keys to be independent, which is also not realistic in practice.

The problem of determining characteristics, let alone differentials, of high probability, is generally very hard. While generic approaches do exist, the attacker must usually, at the very least, adapt a generic approach to fit the cipher at hand, if not explicitly exploit some particular structure of it. In Section 2.1, we put the theory introduced here into practice, when we present a range of results using differential cryptanalysis on the recent block cipher SIMON.

Variants

In the following, we describe some variants of differential cryptanalysis that, in some cases, may either decrease the complexities of an attack, or increase the number of rounds an attacker is able to cover with the analysis. We describe *impossible differentials*, *truncated differentials* and *higher-order differentials*. We go into detail with the two former variants, when we consider them with respect to the block cipher SIMON in Section 2.1.

Truncated Differentials. The term truncated differentials comes from the idea, that we can introduce uncertainty into the concept of a difference as we know it from Definition 7.

Definition 16 (Truncated difference)**.** *Let n be a positive integer. An n-bit* truncated difference *is a value $\alpha \in \{0, 1, \star\}^n$. When $\star$ does not occur in α, it has the same meaning as a regular difference. However, a $\star$ indicates an* unknown bit*, i.e. it represents* either *a zero* or *a one.*

As an example with $n = 4$, the truncated difference $\alpha = (0, 1, \star, \star)$ captures four possible differences $\{(0, 1, 0, 0), (0, 1, 0, 1), (0, 1, 1, 0), (0, 1, 1, 1)\}$. In a sense, a truncated difference is nothing else than a stronger and more flexible way to express expected differences between states during the encryption of two messages M, M'. When considering a non-linear part of a cipher, with a truncated difference, we are able to express a differential transition of probability one. This is most easily seen by example, so let us consider a simple S-box S defined by Table 1.1. We usually express S-boxes by a table in hexadecimal notation, but to highlight the point, we give it here in binary notation.

Table 1.1: Example 3-bit S-box given in binary notation

X	000	001	010	011	100	101	110	111
$S(X)$	100	110	011	000	001	010	111	101

Let us consider a difference $\alpha = (0, 0, 1)$ going into S. By inspection, we see that the possible output differences are $\beta \in \{(0, 1, 0), (0, 1, 1)\}$, each occurring with probability $p = 1/2$. In other words, we have $\mathsf{DCP}_S(\alpha, \beta) = 1/2$. However, by noting that the first two bits of the possible output differences agree, we see that $\mathsf{DCP}_S(\alpha, (0, 1, \star)) = 1$. Now, to take the analysis one step further, one could consider how the truncated difference $(0, 1, \star)$ behaves again over S, by considering the same analysis for both values of β as an *input* difference. As such, truncated differentials are on one hand a way of boosting the differential probability, but on the other hand, ultimately extract less knowledge about the system. We remark that with truncated differentials, key recovery attacks would work in the same way as with a regular differential.

Impossible Differentials. As already stated, an impossible differential is a differential (α, β) of probability zero, i.e. $\mathsf{DP}_{\mathcal{E}}(\alpha, \beta) = 0$. One way to obtain an impossible differential is to combine two differentials: (α, δ) which covers the first part of the cipher, and (γ, β) which covers the last part. Both differentials should have probability one, and should be chosen such that the difference δ *conflicts* with the difference γ. Thus, when combined, the overall differential (α, β) has probability zero – it is *impossible*.

Most commonly, but not always, the approach to constructing the differentials of probability one would be using truncated differentials. For example, one could imagine finding truncated differentials (α, δ) and (γ, β), both of probability one, but such that there is a position i where $\delta_i = 0$ and $\gamma_i = 1$, or the other way around, thus creating the conflict to obtain the impossible differential.

To perform a key recovery attack with an impossible differential, the procedure is much the same as with an ordinary differential. The crucial difference (no pun intended) is, that when we make guesses for the last round key $\tilde{K}_{T-1}$, we go backwards one round, and we check for each message pair (M, M'), where $M \oplus M' = \alpha$, and each corresponding ciphertext pair (C, C') observed, whether $F_{T-1}^{-1}(\tilde{K}_{T-1}, C) \oplus F_{T-1}^{-1}(\tilde{K}_{T-1}, C') = \beta$. If this is the case, we know that $\tilde{K}_{T-1}$ *can not* be the correct key K_{T-1}, because under this key, the differential (α, β) is impossible. As already mentioned, we shall see how impossible differentials, implicitly using truncated differentials, can be applied to the block cipher SIMON in Section 2.1.

Higher-Order Differentials. The concept of *higher-order differentials* generalizes that of differentials, as considered so far. Its application in block cipher cryptanalysis is due to work by Lai [201] and Knudsen [188]. It was shown by Lai that the ordinary differential is a special case of a higher-order derivative, in the sense that

$$\Delta_\alpha F(X) = F(X \oplus \alpha) \oplus F(X) \tag{1.55}$$

is the first-order derivative in the definition of a d^{th} order derivative given by

$$\Delta^{(d)}_{\alpha_1,\ldots,\alpha_d} F(X) = \Delta_{\alpha_d}(\Delta^{(d-1)}_{\alpha_1,\ldots,\alpha_{d-1}} F(X)). \tag{1.56}$$

For example, a second-order differential using differences α_1 and α_2 is defined as

$$F(X) \oplus F(X \oplus \alpha_1) \oplus F(X \oplus \alpha_2) \oplus F(X \oplus \alpha_1 \oplus \alpha_2). \tag{1.57}$$

The idea of Knudsen to apply higher-order derivatives to block cipher cryptanalysis is based on the observation that taking derivatives decreases the degree of a function. As such, if $\mathcal{E}$ is a block cipher of degree d, then the $(d+1)^{\text{st}}$ order derivative of $\mathcal{E}_K$ as given by Eq. (1.56) at *any* input X point equals zero.

We do not go into the details of key recovery attacks using higher-order differentials. Instead, we refer the interested reader to e.g. [188, 192, 201]. The application of higher-order attacks in practice are rather uncommon. One example worth mentioning, however, is the KN-cipher [248] by Nyberg and Knudsen, which was designed to be resistant towards ordinary differential cryptanalysis, but which was broken using a higher-order attack, as shown by Jakobsen and Knudsen [170]. MISTY1, a design by Matsui [218], is another example cipher susceptible to higher-order attacks, as shown by e.g. Babbage and Frisch [35] and Bar-On [37].

1.3.6 Linear Cryptanalysis

When introducing differential cryptanalysis above, we mentioned that it is one of the most powerful tools available to the cryptanalyst. The same can be said of *linear cryptanalysis*, which is the topic for this section. While differential cryptanalysis is a CPA attack, since the attacker needs to *choose* message pairs (M, M') with a particular difference, linear cryptanalysis has the advantage of being a KPA.

Linear cryptanalysis was pioneered by Matsui in 1993, when it was applied to the DES block cipher [217]. More recently, a lot of work analyzes the links between differential- and linear cryptanalysis, and we touch briefly on this at the end of this section. While differential cryptanalysis considered *pairs* of messages and ciphertexts (M, M') and (C, C'), linear cryptanalysis uses messages and their corresponding ciphertexts, but not in pairs.

In a nutshell, linear cryptanalysis is about making linear approximations to non-linear components of a cipher. Such a linear approximation will be expressed in terms of the message M, the ciphertext C and the secret key K (or the round keys in the case of an iterated cipher). To properly introduce the concept, we start off with some notation.

Definition 17 (Linear mask)**.** *Let n be a positive integer. A* linear mask*, or just* mask*, is a value $\alpha \in \mathbb{F}_2^n$. The bits of a mask α which equal* 1 *are called the* active bits.

An n-bit linear mask serves the purpose of *selecting* particular bits (the active bits) from an n-bit value $X \in \mathbb{F}_2^n$. Letting $\alpha = \alpha_{n-1}\|\cdots\|\alpha_0$ and $X = X_{n-1}\|\cdots\|X_0$, this is done when we compute the *inner product*, denoted $\langle\alpha,X\rangle$, where the parity of the *masked bits* is computed as

$$\langle\alpha,X\rangle = \bigoplus_{i=0}^{n-1} \alpha_i X_i. \tag{1.58}$$

Definition 18 (Linear trail). *Let T and n be positive integers and denote by $F_0,\ldots,F_{T-1}$ vectorial Boolean functions*

$$F_t : \mathbb{F}_2^n \rightarrow \mathbb{F}_2^n, \quad 0 \leq t < T, \tag{1.59}$$

and let $F = F_{T-1} \circ \cdots \circ F_0$. A tuple $(\alpha_0,\ldots,\alpha_T) \in (\mathbb{F}_2^n)^{T+1}$ is called a linear trail*, or simply* trail*, for the function F. We refer to α_0 as the* input mask *and to α_T as the* output mask*. The values $\alpha_t, 0 < t < T$, are referred to as* intermediate masks.

Informally, a linear trail expresses for $t = 0,\ldots,T-1$ that the parity of particular bits of the input to F_t is equal to the parity of particular bits of the output of F_t. This is expressed in the masks α_t and α_{t+1}. Naturally, we are interested in the probability that such linear approximations hold, not just for each of the rounds but also for the trail as a whole. To that end, we make the following definition.

Definition 19 (Trail correlation). *Let T, n and κ be positive integers and denote by $F_0,\ldots,F_{T-1}$ vectorial Boolean functions*

$$F_t : \mathbb{F}_2^n \rightarrow \mathbb{F}_2^n, \quad 0 \leq t < T, \tag{1.60}$$

and let $F = F_{T-1} \circ \cdots \circ F_0$. Let $(\alpha_0,\ldots,\alpha_T)$ be a T-round linear trail over F and let $\tilde{F}_t$ be shorthand for the composition of the first $t+1$ functions, i.e.

$$\tilde{F}_t = F_t \circ \cdots \circ F_0, \quad 0 \leq t < T. \tag{1.61}$$

For $t = 0,\ldots,T-1$, we define the correlation *of (α_t,α_{t+1}) over F_t, denoted* $\mathsf{Corr}_{F_t}(\alpha_t,\alpha_{t+1})$*, as*

$$\mathsf{Corr}_{F_t}(\alpha_t,\alpha_{t+1}) = 2 \cdot \Pr_X\big[\langle X,\alpha_t\rangle = \langle F_t(X),\alpha_{t+1}\rangle\big] - 1, \tag{1.62}$$

where the probability is taken over $X \in \mathbb{F}_2^n$. When F_t is the round function of an iterated cipher, the probability is furthermore taken over the round key.

Under the assumption that for $t = 0,\ldots,T-1$, the round correlations $\mathsf{Corr}_{F_t}(\alpha_t,\alpha_{t+1})$ are independent, a method called the *piling-up lemma* (see e.g [217, Lemma 3] by Matsui) is useful for determining the combined *linear trail correlation* over F which, when written in terms of correlations, is given by

$$\mathsf{Corr}_F(\alpha_0,\ldots,\alpha_T) = \prod_{t=0}^{T-1} \mathsf{Corr}_{F_t}(\alpha_t,\alpha_{t+1}). \tag{1.63}$$

Next, we define what we mean by a Markov cipher, but this time in the framework of linear cryptanalysis.

Definition 20 (Markov cipher (for linear cryptanalysis)). *Let $\mathscr{E}_K = F_{T-1}(K_{T-1},\cdot)\circ\cdots\circ F_0(K_0,\cdot)$ be a T-round iterated cipher. We say that $\mathscr{E}_K$ is a* Markov cipher, *with respect to linear cryptanalysis,* if and only if *for all $t = 0,\dots,T-1$, it holds that*

$$\left|\Pr_X\left[\langle X,\alpha_t\rangle = \langle F_t(K_t,X),\alpha_{t+1}\rangle\right] - \frac{1}{2}\right|, \tag{1.64}$$

where the probability is taken over $X \in \mathbb{F}_2^n$, is independent of X for any pair of masks $(\alpha_t,\alpha_{t+1}) \in \mathbb{F}_2^n \times \mathbb{F}_2^n$, when $K \xleftarrow{\$} \mathbb{F}_2^\kappa$.

With our introduction of differential cryptanalysis we saw that Markov ciphers are particularly well-behaved when it comes to analyzing the probabilities involved, and thus the success probabilities and complexities associated with the attack. With our definition of Markov ciphers, in terms of linear cryptanalysis, it is no different, because for a Markov cipher with independent round keys, we can obtain the trail correlation as the product of the round correlations as in Eq. (1.63). In general, this assumption does not strictly hold, but the cryptanalyst makes the assumption regardless, as it is the best tool available to assess the resistance of a cipher towards linear cryptanalysis.

When introducing differential cryptanalysis, we saw that an attacker is not particularly concerned with a specific *characteristic*, but rather a *differential*. This, too, can be translated to linear cryptanalysis, where we again are not particularly concerned with the parity of particular bits in the internal state, but rather the correlation between the parity of particular bits of the message M and ciphertext $C = \mathscr{E}_K(M)$. To that end, we introduce *linear hulls* in the following.

Definition 21 (Linear hull). *Let T be a positive integer and let α_0 and α_T be linear masks. A T-round* linear hull *with input mask α_0 and output mask α_T, denoted $\mathsf{LH}_T(\alpha_0,\alpha_T)$, is the* set *of linear trails with input mask α_0 and output mask α_T, but where the intermediate masks can take any value.*

While we defined the notion of correlation for a trail, it extends naturally to a linear hull as well. In particular, the *linear hull correlation* for $\mathsf{LH}_T(\alpha_0,\alpha_T)$ over $\mathscr{E}$ is defined as

$$\begin{aligned}\mathsf{Corr}_{\mathscr{E}}(\mathsf{LH}_T(\alpha_0,\alpha_T)) &= \sum_{s\in\mathsf{LH}_T(\alpha_0,\alpha_T)} \mathsf{Corr}_{\mathscr{E}}(s)\\ &= \sum_{s\in\mathsf{LH}_T(\alpha_0,\alpha_T)} (-1)^{\mathrm{sgn}(s)}|\mathsf{Corr}_{\mathscr{E}}(s)|, \quad \mathrm{sgn}(s) = \begin{cases}0 & ,\mathsf{Corr}_{\mathscr{E}}(s)\geq 0\\ 1 & ,\mathsf{Corr}_{\mathscr{E}}(s) < 0.\end{cases}\end{aligned} \tag{1.65}$$

We also remark that a 1-round linear hull reduces to a 1-round linear trail, as there are no intermediate masks. This reduction carries over naturally to correlations, so in such a case $\mathsf{Corr}_{\mathscr{E}}(\mathsf{LH}_1(\alpha_0,\alpha_1)) = \mathsf{Corr}_{\mathscr{E}}(\alpha_0,\alpha_1)$. When the linear hull input/output masks are understood from the context, we write for simplicity $\mathsf{Corr}_{\mathscr{E}}$ for the linear hull correlation.

For a vectorial Boolean function $F : \mathbb{F}_2^n \to \mathbb{F}_2^m$, note that a linear hull $\mathsf{LH}_T(\alpha,\beta)$, where $(\alpha,\beta)\in\mathbb{F}_2^n\times\mathbb{F}_2^m$, defines a T-round *linear relation* between an input X and $F(X)$, which we denote $\mathscr{R}^F_{\alpha,\beta} : \mathbb{F}_2^n \to \mathbb{F}_2$, where

$$\mathscr{R}^F_{\alpha,\beta}(X) = \begin{cases}1 & ,\langle X,\alpha\rangle = \langle F(X),\beta\rangle\\ 0 & ,\langle X,\alpha\rangle \neq \langle F(X),\beta\rangle.\end{cases} \tag{1.66}$$

When $\mathscr{R}^F_{\alpha,\beta}(X) = 1$ we say the relation is *satisfied* for input X and otherwise it is not. We will use this terminology in Section 2.2, when we consider linear cryptanalysis of the block cipher PRESENT.

In general, an attacker using linear cryptanalysis is interested in determining a linear hull (α, β) which maximizes the absolute correlation $|\mathsf{Corr}_{\mathscr{E}}(\mathsf{LH}_T(\alpha, \beta))|$. While the number of trails in a linear hull, denoted $\sharp\mathsf{LH}_T(\alpha, \beta)$, does not depend on the secret key K, the sign $\mathrm{sgn}(s)$ of each trail $s \in \mathsf{LH}_T(\alpha, \beta)$ does.

Distinguishers and Key Recovery

Obviously, a non-zero correlation of a linear hull $\mathsf{LH}_T(\alpha, \beta)$ leads directly to a distinguishing attack. Consider the scenario where the attacker interacts with an oracle $\mathscr{O}$ by providing inputs and requesting outputs. The oracle either represents the instantiation of $\mathscr{E}$ with a particular fixed key K, or it samples the outputs from an ideal permutation. The attacker should now guess, solely based on N queried inputs M_i and the corresponding outputs $C_i = \mathscr{O}(M_i)$, whether the oracle was interacting with $\mathscr{E}_K$ or not. To do this, the attacker initializes two counters T_0 and T_1, both to zero. She checks each pair (M_i, C_i) and in particular computes $\langle M_i, \alpha\rangle \oplus \langle C_i, \beta\rangle$, and when it equals zero she increases T_0 and otherwise she increases T_1. The distinguisher comes from the fact that if the oracle was *not* interacting with $\mathscr{E}_K$, we expect that both T_0 and T_1 have values very close to $N/2$. However, if one of T_0 or T_1 is close to $\frac{1}{2}N(\mathsf{Corr}_{\mathscr{E}} + 1)$, we expect that the oracle was interacting with $\mathscr{E}_K$. To be able to correctly identify the correct key guess, the number of pairs N used should be in the order of $c/\mathsf{Corr}^2_{\mathscr{E}}$, where c is a small constant [192, Chapter 8].

With differential cryptanalysis, we saw that a distinguisher could be used to define an attack recovering (parts of) the last round key. With linear cryptanalysis, we can do the same thing. Let $\mathscr{E}$ be a T-round key-alternating cipher, i.e. consisting of round functions $F_0, \ldots, F_{T-1}$ using round keys $K_0, \ldots, K_T$. Furthermore, we denote by γ the number of key candidates for (parts of) the last round key K_T and consider a $(T-1)$-round linear hull $\mathsf{LH}_{T-1}(\alpha, \beta)$. We initialize 2γ counters $T_{v,0}$ and $T_{v,1}$, $0 \le v < \gamma$, to zero. Just as the approach with differential cryptanalysis, the attacker iterates over all γ possible key guesses $\tilde{K}_T$. She then computes backwards using that guess for the last round, and computes $\langle M, \alpha\rangle \oplus \langle F^{-1}_{T-1}(C \oplus \tilde{K}_T), \beta\rangle$. If it equal zero, she increases the counter $T_{\tilde{K}_T,0}$, otherwise she increases $T_{\tilde{K}_T,1}$. We make the assumption that under an incorrect key guess $\tilde{K}_T$, either of the counters $T_{\tilde{K}_T,0}$ and $T_{\tilde{K}_T,1}$ have a probability of $1/2$ to be increased. When this procedure is repeated for many message/ciphertext pairs (M, C), the assumptions imply that for the correct key K_T, the magnitude of either $T_{K_T,0}$ or $T_{K_T,1}$ is expected to be significantly higher than $N/2$. The attack is summarized as Algorithm 2.

Like its differential counterpart, linear cryptanalysis too is a very powerful cryptanalytic tool. As described, it can be used to attempt the goal of key recovery, the most severe compromise to a block cipher possible. We also saw that linear cryptanalysis can be used simply to distinguish the block cipher from an ideal primitive. In Section 2.2, we apply linear cryptanalysis to the standardized block cipher PRESENT. Rather than considering the cipher in the secret-key model, we will apply our analysis in the key-less model, as described in Section 1.3.1 above. As we shall see, the key-less model is an interesting case study if the block cipher under analysis is used to construct a compression function in the MMO mode.

Algorithm 2: LinearHullKeyRecovery

```
Data: Pairs $(M_i, C_i)$ with $1 \leq i \leq N$, s.t. $C_i = \mathcal{E}_K(M_i)$ and a linear hull $\mathsf{LH}_{T-1}(\alpha, \beta)$
Result: Guess $\hat{K}$ for the last round key $K_T$
1  for $v = 0, \dots, \gamma - 1$ do
2      Initialize counters $T_{v,0}$ and $T_{v,1}$ to zero
3  end
4  for $\tilde{K}_T = 0, \dots, \gamma - 1$ do                 // Iterate over key guesses
5      for $i = 1, \dots, N$ do
6          $w \leftarrow \langle M_i, \alpha \rangle \oplus \langle F_{T-1}^{-1}(C_i \oplus \tilde{K}_T), \beta \rangle$ ;   // Compute parity
7          $T_{\tilde{K}_T, w} \leftarrow T_{\tilde{K}_T, w} + 1$
8      end
9  end
10 $\hat{K} \leftarrow \max_v \{T_{v,0}, T_{v,1}\}$
11 return $\hat{K}$
```

Variants

There are not as many widely applied variants of linear cryptanalysis as there is for its differential counterpart. However, we mention briefly a few in the following.

In [177], Kaliski and Robshaw propose to use not a single, but *multiple* linear approximations for the same internal key bits. Thus, with each approximation, we get more suggested values for the targeted key bits, and can thus reduce the data complexity. Hermelin, Cho, and Nyberg introduced in [157] a new application of linear cryptanalysis using *multidimensional linear approximations*, with an application to a version of the block cipher SERPENT [57] reduced to 4 rounds. One interesting property of their approach is, that one does not need the assumption of statistical independence of the approximations used, contrary to the approach of Kaliski and Robshaw.

Above, we saw impossible differentials as a variant where we consider differentials of probability zero, and use these to filter out round key candidates that are certainly wrong. The dual to this idea, in the setting of linear cryptanalysis, was first investigated by Bogdanov and Rijmen in [74]. Under the term *zero-correlation attacks*, they show how linear approximations of correlation zero can be used to mount key-recovery attacks on reduced-round AES-192, AES-256 and CLEFIA-256.

As we have already seen several times, there are similarities between concepts and approaches in differential and linear cryptanalysis. At the time of writing, a very active research topic attempts to uncover the links between the two approaches. While we do not go into further details here, we refer the interested reader to e.g. the works by Chabaud and Vaudenay [91] and Blondeau and Nyberg [67, 68]. With this said, there are also places where the two concepts differ significantly. For example, we have seen that the differential probability increases with each characteristic contained, while for a linear hull this is not true, as the linear hull correlation depends also on the particular sign of each trail correlation.

1.3.7 Meet-in-the-Middle Attacks

Next we describe a very generic attack approach called *meet-in-the-middle* (MitM) attacks. Consider a cipher which takes a key consisting of two *independent* parts, $K = (K_0, K_1)$, and which uses two keyed functions F_0 and F_1 to define encryption as

$$\begin{aligned} \mathscr{E} : \mathbb{F}_2^\kappa \times \mathbb{F}_2^\kappa \times \mathbb{F}_2^n &\to \mathbb{F}_2^n \\ (K_0, K_1, M) &\mapsto F_1(K_1, F_0(K_0, M)). \end{aligned} \tag{1.67}$$

Such an approach is very naturally called *double encryption*. While the classical way of describing a MitM attack (see e.g. [192, Chapter 5]) considers such a scheme, the applicability of such an attack in practice is debatable. A more recent approach to meet-in-the-middle attacks, under a more realistic assumption, was defined by Bogdanov and Rechberger in [73] using the *3-subset meet-in-the-middle attack*. We describe this in the following.

Consider a T-round iterated block cipher $\mathscr{E}_K$ using a secret key $K \in \mathbb{F}_2^\kappa$, consisting of round functions $F_0(K,\cdot), \dots, F_{T-1}(K,\cdot)$. We let $\varphi_{i,j} = F_{j-1}(K,\cdot) \circ \cdots \circ F_i(K,\cdot)$ denote the application of rounds i through $j-1$, with $0 \le i < j \le T$. Consider splitting the encryption function into two parts: $\mathscr{E}_K = \varphi_{\alpha,T} \circ \varphi_{0,\alpha}$, with $0 < \alpha < T$. We consider two sets of key bits,

$$\begin{aligned} K^0 &= \left\{K_i \mid \text{key bit } K_i \text{ used by } \varphi_{0,\alpha}\right\} \quad \text{and} \\ K^1 &= \left\{K_i \mid \text{key bit } K_i \text{ used by } \varphi_{\alpha,T}\right\}. \end{aligned} \tag{1.68}$$

Then we let $A_m = K^0 \cap K^1$ denote the key bits that are used in both $\varphi_{0,\alpha}$ and $\varphi_{\alpha,T}$. We also let $A_0 = K^0 \backslash A_m$ and $A_1 = K^1 \backslash A_m$ denote the key bits which are *only* used in $\varphi_{0,\alpha}$ and $\varphi_{\alpha,T}$, respectively. The attack works in two phases, which we describe next.

MitM Phase

It follows from the definitions that by guessing the bits in A_0 and A_m, one can compute all of $\varphi_{0,\alpha}$, and similarly for $\varphi_{\alpha,T}$. Thus, the MitM phase progresses as specified in Algorithm 3. We consider a single message/ciphertext pair (M, C) and let a_m, a_0 and a_1 denote particular guesses of the bits of A_m,A_0 and A_1, respectively.

The algorithm proceeds by first fixing a guess a_m of the key bits required by *both* $\varphi_{0,\alpha}$ and $\varphi_{\alpha,T}$. For each such guess, we initialize two sets U and V. The set U will hold the values of $\varphi_{0,\alpha}(M)$ which are obtainable after combining each guess for a_0 with the current guess for a_m. Correspondingly, the set V contains the possible values $\varphi_{\alpha,T}^{-1}(C)$ obtainable from any guess a_1 combined with the current guess of a_m. The algorithm MATCH-IN-THE-MIDDLE, which we do not describe in pseudo-code, takes the two sets U and V, and determines which of the key guesses for a_0 are compatible with which key guesses for a_1, in the sense that the corresponding values of $\varphi_{0,\alpha}(M)$ and $\varphi_{\alpha,T}^{-1}(C)$ can match at the point where round F_α is about to commence. The compatible combinations of a_0 and a_1 are returned in a list by the MATCH-IN-THE-MIDDLE algorithm, and the resulting combinations are paired with the current value for a_m to specify a full key candidate, which is stored in $\hat{K}$.

Algorithm 3: MEET-IN-THE-MIDDLE

```
   Data: A single message/ciphertext pair $(M, C)$, s.t. $C = \mathscr{E}_K(M)$
   Result: List of key candidates $\hat{K}$
1  $\hat{K} \leftarrow \emptyset$
2  foreach guess $a_m$ of key bits in $A_m$ do
3      $U, V \leftarrow \emptyset$
4      foreach guess $a_0$ of key bits in $A_0$ do $U \leftarrow U \cup \{(\varphi_{0,\alpha}(M), a_0)\}$
5      foreach guess $a_1$ of key bits in $A_1$ do $V \leftarrow V \cup \{(\varphi_{\alpha,T}^{-1}(C), a_1)\}$
6      $\{(a_{0,i}, a_{1,i})\}_{i=1}^{\ell} \leftarrow$ MATCH-IN-THE-MIDDLE$(U, V)$
7      $\hat{K} \leftarrow \hat{K} \cup \{(a_m, a_{0,i}, a_{1,i}) \mid 1 \leq i \leq \ell\}$
8  end
9  return $\hat{K}$
```

Key Testing Phase

While the MitM phase above uses only a *single* message/ciphertext pair, the key testing phase requires the availability of several pairs. In this phase, the key candidates $\hat{K}$ remaining from the MitM phase are tested in a brute-force manner, using N message/ciphertext pairs $(M_i, C_i), 1 \leq i \leq N$, until the single correct key K has been found. The number of pairs N needed, and hence the data complexity of the attack, is determined by (see [73])

$$N = \left\lceil \frac{\kappa}{n} \right\rceil, \tag{1.69}$$

while the memory complexity is defined by that required by the MATCH-IN-THE-MIDDLE algorithm. We remark that if $\sharp A_0 + \sharp A_1 > 2$, and both are non-empty, then the attack is more efficient than a simple brute-force attack, given that the probability for a wrong key candidate to survive the matching is sufficiently low.

Further Developments

In recent years the MitM attack has evolved in different directions, especially with applications to the AES. In this line, Demirci and Selçuk showed in 2008 an attack on 7 rounds of AES-192, and 8 rounds of AES-256 [111]. The result on AES-192 is improved in 2010 to cover 8 rounds by Dunkelman, Keller, and Shamir [118]. In [77], Bogdanov, Khovratovich, and Rechberger introduce a new sophisticated variant of a MitM attack coined the *biclique attack*. With their approach, they for the first time present key-recovery attacks on the *full* version of each AES variant, albeit with a marginal time complexity, effectively shaving off around 2 bits of security for each variant. Derbez, Fouque, and Jean improve in [112] the best known attacks on 7-round AES-128 and 8 rounds of AES-192 and AES-256, respectively.

Meanwhile, we remark that the improvements in MitM attacks have not only been applicable to block ciphers, but also hash functions (see e.g. the pre-image attacks on Skein-512 and the SHA-2 family [186]).

1.3.8 Algebraic Attacks

While we will not be applying *algebraic attacks* in this thesis, we nevertheless describe the attack vector briefly here, due to its interesting and very different nature, compared to other attack vectors described so far.

Let K be a field and let $K[X_1,\dots,X_m]$ be a polynomial ring in m variables over K. With algebraic attacks, the first step is to express the behavior of the cipher by k non-linear polynomials $f_1,\dots,f_k \in K[X_1,\dots,X_m]$. In the second step, one solves the system of equations

$$\begin{aligned} f_1(X_1,\dots,X_m) &= 0 \\ f_2(X_1,\dots,X_m) &= 0 \\ &\vdots \\ f_k(X_1,\dots,X_m) &= 0 \end{aligned} \tag{1.70}$$

in order to determine the values of the variables representing the secret key of the cipher, thereby breaking it. In general, solving such a system of multivariate non-linear polynomials is an NP-hard problem (see e.g. [138, Appendix A7.2] for the case $K = \mathbb{F}_2$). In practice, one typically uses either a *SAT solver* or *Gröbner basis* computations to solve the system. If one uses a SAT solver, the system of equations is modeled as a Boolean satisfiability problem, which is an NP-complete decision problem, and an off-the-shelf solver is used to try to break the system. A Gröbner basis is a special mathematical structure that relates to the polynomials $f_1,\dots,f_k$ in such a way, that if this basis can be found, the set of solutions to Eq. (1.70) can be efficiently found.

In general, algebraic cryptanalysis is not a successful attack vector for block ciphers, and certainly does not compare to differential- and linear cryptanalysis. However, counterexamples do exist, such as the attack on KeeLoq by Courtois, Bard, and Wagner [103]. Algebraic attacks have been more successfully applied to stream ciphers. The work of Courtois and Meier gives a high-level overview of application to stream ciphers [102] with examples on the Toyocrypt and LILI-128 ciphers. In [30], Armknecht provides improvements on previous work by Courtois [101] on fast algebraic attacks, and provides results on the E_0 key stream generator from the Bluetooth standard [162].

2 Cryptanalysis of Symmetric Primitives

Having introduced many important concepts of symmetric cryptography in Chapter 1, covering the goals and purposes of symmetric cryptography, over design approaches to different primitives, to cryptanalysis including adversarial assumptions, we now turn to applying cryptanalysis to specific primitives. In particular, in the following three sections, we present recent cryptanalytic results on two block ciphers. The first, SIMON, is a recent design by the NSA. We show some attacks using differential cryptanalysis, and show connections to linear cryptanalysis for these. Secondly, we consider the standardized block cipher PRESENT in the adversarial model of the key-less setting. We define a new model for distinguishers using linear cryptanalysis under this assumption, and apply it to PRESENT. Finally, the last section covers cryptanalysis of AE(AD) schemes. We show first a very generic attack that can be used in our case to completely recover the secret key for the round-one CAESAR candidate AVALANCHE. Finally, we perform a key-recovery attack against an AE scheme RBS, using a very interesting design property which, as we show, introduces a fatal weakness.

At first, it may seem strange to cover cryptanalysis of symmetric primitives in this chapter, before we turn to covering their *design* in Chapter 3. However, as a designer of a new symmetric primitive it is of utmost importance to understand which attacks might be applicable to the design, to be able to choose design parameters leading to a good combination of security and performance. Arguably, designing an excellent new primitive is the hardest task possible for a researcher in symmetric cryptography. The cryptanalytic results presented in this chapter should make the reader more comfortable with the concepts of differential- and linear cryptanalysis, as well as some generic techniques from Section 2.3, to better appreciate the design choices made in Chapter 3.

2.1 Cryptanalysis of SIMON

In June 2013, researchers from the NSA published on the ePrint archive two lightweight block cipher designs named SIMON and SPECK. In this section, we apply the concepts of differential cryptanalysis, with a variant of impossible differentials as introduced in Section 1.3.5, to the block cipher SIMON. We also describe a connection to linear cryptanalysis of SIMON, and give an overview of how analysis of SIMON has progressed since its publication in 2013.

Publications

The results presented in this section are from:

[16] Javad Alizadeh, Hoda A. AlKhzaimi, Mohammad Reza Aref, Nasour Bagheri, Praveen Gauravaram, Abhishek Kumar, Martin M. Lauridsen, and Somitra Kumar Sanadhya. Cryptanalysis of SIMON Variants with Connections. In Nitesh Saxena and Ahmad-Reza Sadeghi, editors, *Radio Frequency Identification: Security and Privacy Issues - 10th International Workshop, RFIDSec 2014, Oxford, UK, July 21-23, 2014, Revised Selected Papers*, volume 8651 of *Lecture Notes in Computer Science*, pages 90–107. Springer, 2014.

[19] Hoda A. Alkhzaimi and Martin M. Lauridsen. Cryptanalysis of the SIMON Family of Block Ciphers. Cryptology ePrint Archive, Report 2013/543, 2013.

The RFIDsec paper [16] is the merged result of the ePrint report [19] and the work of Alizadeh, Bagheri, Gauravaram, Kumar, and Sanadhya [15].

Author Contribution

Among the results presented in this section, the author is partly responsible for all results on: differential cryptanalysis; impossible differentials; analysis of differential effect in SIMON; and search heuristic implementation. The observation and results described in Section 2.1.9, which describes connections between differential- and linear cryptanalysis in SIMON, is the work of [15].

2.1.1 Specification of SIMON

SIMON is a family of lightweight block ciphers designed by the NSA, with the aim of providing a cipher optimized for hardware performance [40, 41]. The design of SIMON is a classical Feistel construction as introduced in Section 1.2, operating on two $(n/2)$-bit halves in each round, thus the block size is n bits. Each round of SIMON applies a non-linear, non-bijective, and hence non-invertible function

$$\begin{aligned} F : \mathbb{F}_2^{n/2} &\rightarrow \mathbb{F}_2^{n/2} \\ X &\mapsto ((X \lll 8) \odot (X \lll 1)) \oplus (X \lll 2) \end{aligned} \tag{2.1}$$

to the left half of the state. The output of F is added, using XOR, to the right half along with a round key, and the two halves are swapped.

Variants of SIMON exist for different parameters of key size, block size and number of rounds. The name of each SIMON variant with its parameters are presented in Table 2.1. The meaning of the number of key words m, and index into z, will be made clear when we describe the key schedule for SIMON next.

Table 2.1: Members of the SIMON family and their parameters

Cipher	Block size n	Key words m	Key size κ	Rounds T	Seq. z^j
SIMON32/64	32	4	64	32	z^0
SIMON48/72	48	3	72	36	z^0
SIMON48/96	48	4	96	36	z^1
SIMON64/96	64	3	96	42	z^2
SIMON64/128	64	4	128	44	z^3
SIMON96/92	96	2	92	52	z^2
SIMON96/144	96	3	144	54	z^3
SIMON128/128	128	2	128	68	z^2
SIMON128/192	128	3	192	69	z^3
SIMON128/256	128	4	256	72	z^4

Key Schedule

A T-round variant of SIMON uses T round keys $K_0, \dots, K_{T-1}$ that are extracted from a secret master key K of $\kappa = \frac{mn}{2}$ bits through the SIMON key schedule algorithm. We refer to m as the number of key words for the particular SIMON variant. The key schedule works on $m \in \{2,3,4\}$ $(n/2)$-bit word registers. To start off, the m key words of the master key K are loaded into registers $K_0, \dots, K_{m-1}$, which also equal the first m round keys. Given the key words $K_{t-1}, \dots, K_{t-m}$, the next round key K_t is determined in slightly different ways, depending on the value of m. For $m \in \{2,3\}$, K_t is computed by cyclically rotating K_{t-1} by 3 and 4 positions to the right, respectively. These two rotated versions of K_{t-1} are now XORed together with K_{t-m}, a constant c and another constant z_t^j to obtain K_t. Specifically, when $m \in \{2,3\}$, we have for $t = m, \dots, T-1$ that

$$K_t = (K_{t-1} \ggg 3) \oplus (K_{t-1} \ggg 4) \oplus K_{t-m} \oplus c \oplus z_t^j, \tag{2.2}$$

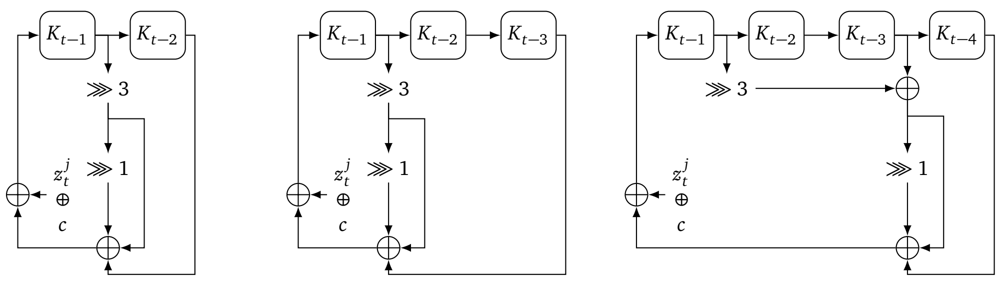
where $c = 2^{n/2} - 4 = \texttt{fff}\cdots\texttt{fc}$ is a fixed constant and z_t^j is the t^{th} bit of the binary string z^j (see Table 2.2). The constants z^j are derived using three 5×5 matrices over $\mathbb{F}_2$. For the details, see the specification [40]. When the number of key words is $m = 4$, the computation is slightly different, as we compute

$$K_t = (((K_{t-1} \ggg 3) \oplus K_{t-3}) \ggg 1) \oplus ((K_{t-1} \ggg 3) \oplus K_{t-3}) \oplus K_{t-m} \oplus c \oplus z_t^j. \tag{2.3}$$

The three variants of the SIMON key schedule are illustrated in Figure 2.1.

2.1.2 Differential Attacks

To begin our analysis of SIMON, we consider ordinary differential cryptanalysis as we described it in Section 1.3. In particular, we start by considering *iterated* differentials, i.e. where one finds



(a) $m = 2$ key words **(b)** $m = 3$ key words **(c)** $m = 4$ key words

Figure 2.1: The SIMON key schedule for the cases $m \in \{2,3,4\}$. The computation of round key K_t depends on K_{t-1} and K_{t-m}, and also K_{t-3} in the case of $m = 4$.

Table 2.2: The z^j vectors used in the SIMON key schedule

j	z^j
0	11111010001001010110000111001101111101000100101011000011100110
1	10001110111110010011000010110101000111011111001001100001011010
2	10101111011100000011010010011000101000010001111110010110110011
3	11011011101011000110010111100000010010001010011100110100001111
4	11010001111001101011011000100000010111000011001010010011101111

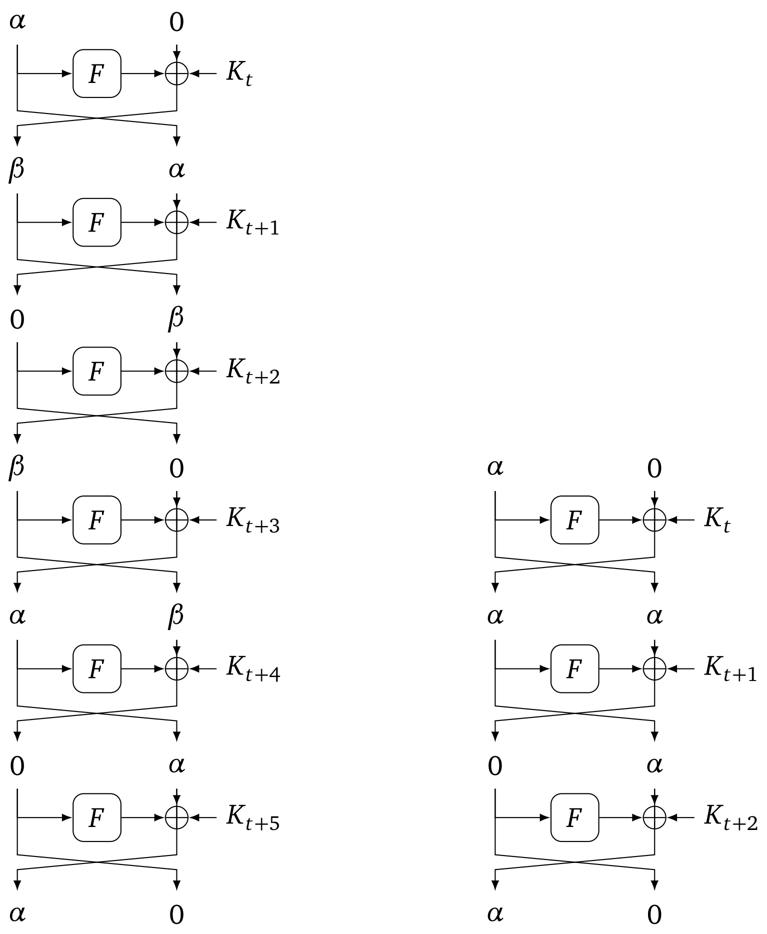
some differential (α, α), and then uses the fact that the input difference equals to the output difference, in order to repeat it several times over more rounds of the cipher.

For the SIMON family of block ciphers, we are interested in one of two properties of F for constructing the iterated differentials. Firstly, we consider pairs of $(n/2)$-bit differences (α, β), for which the combined probability $\mathsf{DCP}_F(\alpha, \beta) \cdot \mathsf{DCP}_F(\beta, \alpha)$ is maximized. We refer to this as a *type-1* iterated characteristic. Secondly, we may consider looking for a characteristic using a *single* difference α, for which $\mathsf{DCP}_F(\alpha, \alpha)$ is maximized. We refer to this as a *type-2* iterated characteristic. For type-1 characteristics, we can construct a 6-round iterative characteristic, while for type-2 we get a similar 3-round characteristic. We illustrate both a type-1 and type-2 iterative characteristic in Figure 2.2.

With our introduction in Chapter 1, we saw that a designer of a block cipher uses a non-linear component to introduce non-linearity in the cipher. We saw that this could be accomplished in SP-networks using S-boxes. In Feistel schemes, and indeed also in SIMON, this non-linearity is introduced using the bitwise AND operation denoted $\odot$. When performing differential cryptanalysis on SIMON, we are thus interested in how an input difference behaves over F with respect to the non-linearity in F. Consider two inputs X and $X \oplus \alpha$ to F. We would like to analyze the probability that the output difference is β given the input difference α, i.e. we consider a characteristic (α, β) over F. Due to the specification of F of Eq. (2.1), we know that a single bit of β depends on 2 bits of the input X and 3 bits of the input difference α, due to the relation

$$\forall i \in \{0, \ldots, n/2-1\} : \beta_i = \alpha_{i-8} X_{i-1} \oplus \alpha_{i-1} X_{i-8} \oplus \alpha_{i-1} \alpha_{i-8} \oplus \alpha_{i-2}, \qquad (2.4)$$

where all indices are computed modulo $n/2$. The canonical way of determining the differential probabilities over a non-linear component is to construct what is known as a *difference distribution*



(a) A 6-round iterated characteristic using two input/output relations

(b) A 3-round iterated characteristic using a single input/output relation

Figure 2.2: Type-1 and type-2 iterated differential characteristics for SIMON. The values indicated in the states are differences.

table (DDT). Essentially, this is a matrix D where the element $D_{\alpha,\beta}$ gives the number of inputs X to the function F, such that $F(X) \oplus F(X \oplus \alpha) = \beta$. When the sizes of the domain and co-domain of the non-linear component are not too big, we can construct such a DDT. For S-boxes, this is almost always feasible, as they usually take inputs of at most 8 bits. However, for the SIMON F function we can not in general construct the DDT, as the complexity of doing so is $O(2^n)$ memory and time. However, for $n = 32$ we are able to construct the DDT for F using 8 GB of memory with an unsigned 16-bit data type for the entries. As such, we can determine the best pairs (α, β) as above for the type-1 characteristic. We find 16 such pairs (α, β) (see Table 2.3) each with the same combined probability

$$\begin{aligned} \mathsf{DCP}_F(\alpha,\beta)\cdot\mathsf{DCP}_F(\beta,\alpha) &= \frac{256}{2^{16}}\cdot\frac{2048}{2^{16}} \\ &= 2^{-13}. \end{aligned} \tag{2.5}$$

By squaring this probability, we find that 2^{-26} is the probability of the 6-round type-1 characteristic shown in Figure 2.2a, when using such an (α,β) pair.

Table 2.3: Best possible (α,β) pairs for type-1 differential characteristics obtained for Simon32/64

α	β	$\mathsf{DCP}_F(\alpha,\beta)$	$\mathsf{DCP}_F(\beta,\alpha)$	α	β	$\mathsf{DCP}_F(\alpha,\beta)$	$\mathsf{DCP}_F(\beta,\alpha)$
0045	051e	2^{-5}	2^{-8}	1401	7814	2^{-5}	2^{-8}
008a	0a3c	2^{-5}	2^{-8}	1e05	4500	2^{-8}	2^{-5}
0114	1478	2^{-5}	2^{-8}	2280	8f02	2^{-5}	2^{-8}
0228	28f0	2^{-5}	2^{-8}	2802	f028	2^{-5}	2^{-8}
028f	8022	2^{-8}	2^{-5}	3c0a	8a00	2^{-8}	2^{-5}
0450	51e0	2^{-5}	2^{-8}	4011	8147	2^{-5}	2^{-8}
08a0	a3c0	2^{-5}	2^{-8}	5004	e051	2^{-5}	2^{-8}
1140	4781	2^{-5}	2^{-8}	a008	c0a3	2^{-5}	2^{-8}

As the difference β does not appear in neither the input- nor output difference of the type-1 characteristic, we may instead think of it as a 6-round differential, where the difference β can take on any possible value. As such, we can search for the best difference α, s.t. the quantity

$$\sum_{\beta\in\mathbb{F}_2^{n/2}} \mathsf{DCP}_F(\alpha,\beta)\cdot\mathsf{DCP}_F(\beta,\alpha) \tag{2.6}$$

is maximized. Doing so, we find that for $n = 32$, there are four such differences

$$\alpha \in \{1111, 2222, 4444, 8888\} \tag{2.7}$$

which maximize Eq. (2.6). These values for α represent 3-round differentials of probability $2^{-11.19}$, where we do not care about the intermediate differences. In other words, they correspond to type-2 characteristics considered as differentials. When putting two such differentials together, we get a 6-round differential of probability at least $2^{-2\cdot 11.19} = 2^{-22.38}$, which is similar to the type-1 characteristic considered as a differential, except that after 3 rounds, we expect a difference of $\alpha \| 0$.

When the block size is $n > 32$, determining the full DDT is computationally infeasible. However, a method by Dinur, Dunkelman, Gutman, and Shamir [115] computes the *diagonal* of the DDT for a function $G : \mathbb{F}_2^m \to \mathbb{F}_2^m$ with a time and memory complexity of $O(2^m)$. Thus, by using this approach, we can obtain results for $n = 48$ as well, with a time and memory requirement of about 2^{24} applications of the Simon F function. The diagonal entries of the DDT represent the iterative characteristics (α,α). The idea of the approach is as follows. A hash table H is used, which maps values $X \oplus F(X)$ to a list holding the X values giving this difference. The table H is constructed by iterating over all $X \in \mathbb{F}_2^{n/2}$. After this, by construction, any pair of distinct $X, X' \in \mathbb{F}_2^{n/2}$ in the list associated with the same difference are values s.t. $X \oplus F(X) = X' \oplus F(X')$. In other words, $\alpha = X \oplus X'$ is the diagonal entry under consideration. However, to compute the actual differential probability, we must again iterate over all $X \in \mathbb{F}_2^{n/2}$ and check how many times $F(X) \oplus F(X \oplus \alpha) = \alpha$. Using this approach, for $n = 32$ and $n = 48$, we obtain a list of the best diagonal differential probabilities, presented in Table 2.4.

Table 2.4: Best diagonal entries of the DDT for $n \in \{32, 48\}$

n	$\mathsf{DCP}_F(\alpha,\alpha)$	Differences α
32	2^{-8}	`5555, aaaa, ac0e, 1d58, ab03, 581d,`
		`3ab0, 6075, 5607, 0eac, b03a, 7560,`
		`c0ea, 03ab, eac0, 81d5, 0756, d581`
48	2^{-12}	`555555, aaaaaa, 0e22ac, 1c4558,`
		`388ab0, 711560, c45581, e22ac0,`
		`88ab03, 115607, 22ac0e, 45581c,`
		`ab0388, b0388a, 560711, 8ab038,` ...

From the probabilities given in Table 2.4, it is evident that already for $n = 32$, the differential characteristic probability $\mathsf{DCP}_F(\alpha,\alpha)$ is too low to be exploitable for an attack. For example, for just a 3-round type-2 differential, we would have a combined probability of $(2^{-n/2})^2 = 2^{-n}$, and the required data complexity would be $c \cdot 2^n$ for a small constant c, thus exceeding the full code book. We remark that the table *suggests* that the best probability for a diagonal entry in the DDT is $2^{-n/4}$, however this is speculation.

2.1.3 Round Function Differentials

Consider an $(n/2)$-bit input difference $\alpha = X \oplus X'$ to F, where $\mathsf{hw}(\alpha) = 1$. As the XOR operation is invariant with respect to rotation, we consider without loss of generality $\alpha = 0 \cdots 01$. Recall that $F(X)$ left rotates X by eight and one positions respectively, applies binary AND to those two, and to the result of that XORs the left rotation of X by two positions. Due to the rotation by two and the XOR, the output difference $\beta = F(X) \oplus F(X')$ will, for this particular α, certainly have an active bit on position 2, i.e. $\beta_2 = 1$. Also, on positions 1 and 8, there *may* be active bits in the output difference (in fact each case occurs, on both bits independently, with probability $\frac{1}{2}$). As the AND operation is non-linear with respect to differences, which of the cases actually occur depends on the inputs X and X'. This observation means that we can describe the output difference β from using an input difference α in truncated form as

$$\beta = 0^{n/2-9} \| \star 000001 \star 0, \tag{2.8}$$

where a $\star$ denotes an unknown bit, just like when we discussed truncated differentials in Section 1.3.

This approach of determining a truncated output difference which holds with probability one can be generalized to arbitrary input differences, and each time we put a $\star$ on a position we lose certainty about that particular bit of the output difference. Note, that this also provides a means of determining all feasible (i.e. with a positive probability) output differences, given some input difference, which in general is very useful for differential analysis. We will use this observation in the following section, and when we consider impossible differentials in Section 2.1.8.

2.1.4 Search Heuristic for Differentials

In the following, we consider organizing differential characteristics in a graph structure. A graph $\Gamma = (V, E)$ consists of a set of vertices V and a set of edges E. Vertices are labeled by an $(n/2)$-bit difference and an edge is an ordered relation between two vertices α and β, denoted $e(\alpha, \beta)$. The edge set is then defined as

$$E = \big\{e(\alpha,\beta) \mid \alpha,\beta \in V \land \mathsf{DCP}_F(\alpha,\beta) > 0\big\}. \tag{2.9}$$

In other words, if we consider a vertex α then there is an edge going from α to a vertex β, *if and only if* β is a feasible output difference over the Simon round function F, when using input difference α.

When considering differential cryptanalysis, such a graph naturally occurs. Consider a fixed input difference α, which we represent by a vertex in V. Over F, we know that there are several possible output differences β, each of them representing a vertex in V. As the induced graph is very large, much larger than we can hope to examine, we define a search heuristic to discover differentials of high probability. We describe this in the following. First of all, we limit the number of rounds T for which we search for differentials. Starting with α, we progress in a depth-first manner, searching through characteristics until we have gone T rounds and end up in an output difference which we denote β. Such a characteristic is modeled by a path of length T from α to β in the graph. Along the way, we keep track of the characteristic probability and add it to the output difference β in a lookup table. At the same time, we keep running score of the best seen output difference, for the fixed α, in terms of differential probability.

Naturally, one can not hope to exhaustively try all input differences α and still examine a significant part of the paths of length T starting from α. To that end, we maintain an array containing the best characteristic probability seen, for each distance from the input difference α in the graph, corresponding to each number of rounds $1, \ldots, T$. We heuristically bound the search at round t, by allowing it only to go to round $t+1$ if the computed characteristic probability for level $t+1$ is within some fraction ω away from what is so far the best observed probability, which is stored in the array. If this is not the case, we cut off that part of the current depth-first search and backtrack to the previous round.

We determine experimentally, for each variant of Simon, the constant fraction ω used in the bounding, which gives the best result. Note that this method of cutting off sub-graphs helps keep the Hamming weight of the differences low, because the characteristic probability drops exponentially with increasing Hamming weight. Furthermore, we considered only input differences α of low Hamming weight, as these have less feasible output differences in the beginning, which are also of low Hamming weight. As such, we do not claim to have found the best differentials for any Simon variant using the approach described, but our results certainly do provide lower bounds. An overview of differential attack parameters and complexities, due to our differentials found with the search heuristic described, can be found in Table 2.5.

Table 2.5: Summary of our differential cryptanalysis results on SIMON

	Rounds		Complexity		
Cipher	Attacked	Total	Data	Memory	Time
SIMON32/64	16	32	$2^{29.48}$	$2^{16.00}$	$2^{26.48}$
SIMON48/72	18	36	$2^{46.42}$	$2^{24.00}$	$2^{43.25}$
SIMON48/96	18	36	$2^{46.42}$	$2^{24.00}$	$2^{43.25}$
SIMON64/96	24	42	$2^{62.01}$	$2^{32.00}$	$2^{58.43}$
SIMON64/128	24	44	$2^{62.01}$	$2^{32.00}$	$2^{58.43}$
SIMON96/92	29	52	$2^{87.53}$	$2^{48.00}$	$2^{83.67}$
SIMON96/144	29	54	$2^{87.53}$	$2^{48.00}$	$2^{83.67}$
SIMON128/128	40	68	$2^{124.80}$	$2^{64.00}$	$2^{120.47}$
SIMON128/192	40	69	$2^{124.80}$	$2^{64.00}$	$2^{120.47}$
SIMON128/256	40	72	$2^{124.80}$	$2^{64.00}$	$2^{120.47}$

2.1.5 Differential Effect in SIMON

Due to the small block- and key size of SIMON32/64, we are able, for a given number of rounds T, to use the search heuristic of Section 2.1.4 above, to determine lower bounds on the expected differential probability $\mathsf{EDP}_{\mathscr{E}}(\alpha,\beta)$ for a particular differential (α,β), as specified in Definition 11. The 12-round differential leading to our 16-round differential attack on SIMON32/64, as described in Table 2.5, is

$$(\alpha,\beta) = (0001\|0000, 0100\|0000). \tag{2.10}$$

For this differential, we found that $\mathsf{EDP}_{\mathscr{E}}(\alpha,\beta) > 2^{-29.481}$. The reason that the bound is not tight (i.e. we have a strict inequality) is twofold. Firstly, due to cutting off parts of the graph during the search, a large portion of characteristics belonging to some differential are never considered. Secondly, the search was, in some cases, stopped before considering all characteristics, even when using the pruning as just described, due to time limitations imposed on jobs run on the high-performance cluster used for the search.

An interesting question we are able to answer using the presented search method, for the smallest version SIMON32/64, is how strong the *differential effect* is. That is, we can determine whether, in one extreme the contribution to the EDP is due to a few (or even a single) characteristics of high probability, or rather, in the other extreme, is the result of clustering of many characteristics of lower probability.

With our search approach, we keep track of the number of characteristics of probability p in the range $]p; 2p]$ belonging to a given differential. This is accomplished by mapping $\lfloor \log_2 p \rfloor$ to a counter, and increasing the counter whenever a new characteristic in the differential of probability p is discovered. We note that the search, and hence the characteristic counting, is stopped at the same point as for differential search, i.e. when obtain the bound $\mathsf{EDP}_{\mathscr{E}}(\alpha,\beta) > 2^{-29.481}$. The

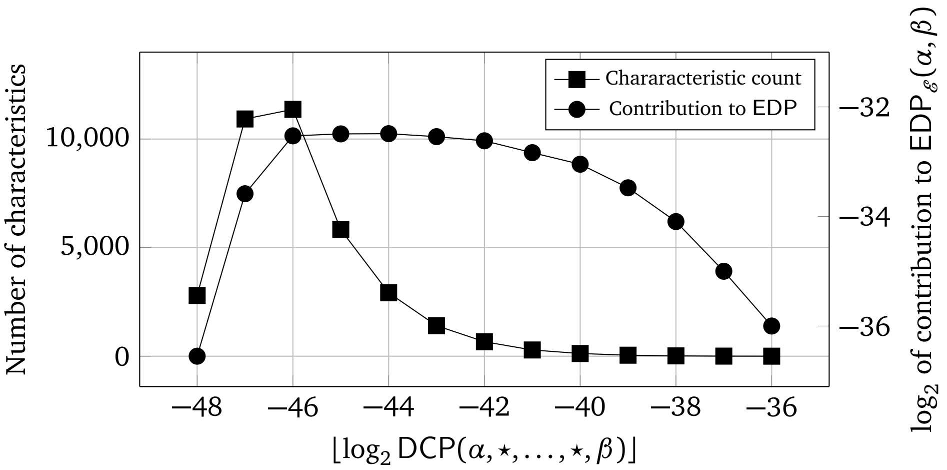


Figure 2.3: Analysis of the differential effect in the 12-round differential $(\alpha,\beta) = (0001||0000, 0100||0000)$ for SIMON32/64

resulting distribution of the number of characteristics and their probabilities are shown in the left axis of Figure 2.3. The figure also shows a division of the characteristics of probability $]p;2p]$ on the bottom axis, and their total contribution to the EDP as the function value on the right axis. We see a low frequency of characteristics of probability 2^{-44} to 2^{-36}. In fact, we find just one characteristic of $\lfloor \log_2 p\rfloor = -36$ and four characteristics of $\lfloor \log_2 p\rfloor = -37$. While these few characteristics do provide an accumulated probability of approximately $2^{-36} + 4\cdot 2^{-37} \approx 2^{-34.42}$, the majority of the $\mathsf{EDP}_{\mathscr{E}}(\alpha,\beta)$ is due to the vast number of characteristics of probability p s.t. $\lfloor \log_2 p\rfloor \in \{-47,\dots,-39\}$. As mentioned, there is only a single characteristic of probability 2^{-36}, which is a factor of approximately $2^{6.5}$ from the bound on $\mathsf{EDP}_{\mathscr{E}}(\alpha,\beta)$. This might give us an indication that the theoretical bound on the EDP, chosen initially by the designers, is based on a provable bound on the characteristic probability, which is close to the 2^{-36} for 12 rounds, as seen above.

In Figure 2.4 we show the results where the same experiment is performed, however with characteristic probability frequencies for *all* differentials for SIMON32/64 with $\mathrm{EDP}_{\mathscr{E}}(\alpha,\beta) > 2^{-33}$ observed during our search. A total of 53 differentials were found. We clearly observe the same substantial differential effect for all 53 cases. Based on this, we conclude that at least for SIMON32/64, there is a prominent clustering of characteristics of lower probability, i.e. a strong differential effect. This could lead to a better understanding of the constraints imposed by the designers of SIMON, especially for smaller block sizes, when considering security bounds against certain attacks such as a differential cryptanalysis.

2.1.6 Generic Extension by Two Rounds on Top

In the following, we describe a generic approach to extend a particular form of differential by two rounds for SIMON, without any loss of differential probability. Consider a $(T-2)$-round differential, where the input difference is of the form $\alpha||0$. As the difference is zero on the right

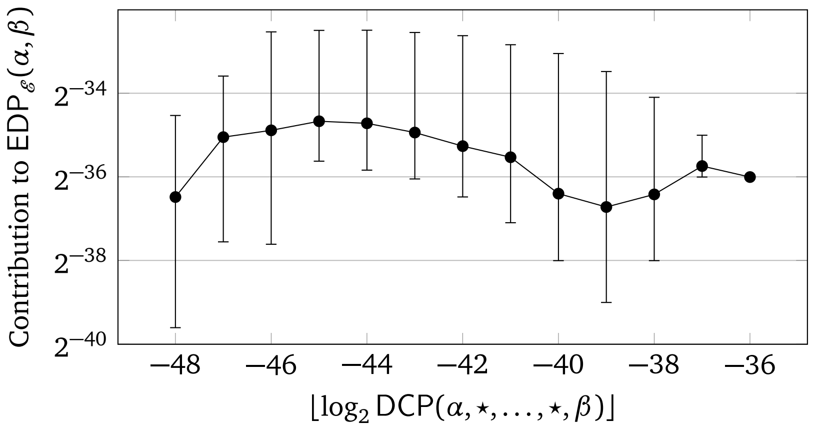


Figure 2.4: Contribution to the $\mathsf{EDP}_{\mathscr{E}}(\alpha,\beta)$ in SIMON32/64 from characteristics of probability in $]p;2p]$ averaged over 53 differentials (α,β) with $\mathsf{EDP}_{\mathscr{E}}(\alpha,\beta) > 2^{-33}$. The error bars show the minimal and maximal contribution to $\mathsf{EDP}_{\mathscr{E}}(\alpha,\beta)$ over all 53 differentials (α,β).

half of the input, the corresponding input difference to F in the previous round is necessarily zero. As such, we can extend the $(T-2)$-round property to a $(T-1)$-round property by using the input difference $0\|\alpha$ instead. Moreover, consider two chosen messages $M = X\|Y$ and $M' = X'\|Y'$, and set $X' = X \oplus \alpha$. By suffering the overhead of computing $F(X)$ and $F(X')$, we can set $Y' = F(X) \oplus F(X')$, thus canceling the differences to get a difference of zero on the left half of the state after the first round. In other words, by choosing a message $M = X\|Y$ and setting

$$\begin{aligned} M' &= X'\|Y' \\ &= (X \oplus \alpha)\|(Y \oplus F(X) \oplus F(X \oplus \alpha)), \end{aligned} \tag{2.11}$$

we have a difference of $\alpha\|0$ after two rounds with probability one. This extends the $(T-2)$-round property to one for T rounds, without reducing the differential probability, but with the overhead of just two applications of F.

2.1.7 Key Recovery

As described in Section 1.3, when using a differential for key recovery, one would normally attack a T-round version of the cipher using a $(T-1)$-round differential. However, as the round key addition is performed after the application of F in each round for SIMON, we will in fact do key recovery on a T-round version of SIMON by using a $(T-2)$-round differential. We refer to Figure 2.5 in our explanation of the key recovery in the following.

Assume that the output difference of the $(T-2)$-round differential is $\alpha\|0$ as shown at the top of the figure. Furthermore, let an output ciphertext pair be $C_L\|C_R$ and $C_L'\|C_R'$, for which the corresponding input message pair have a chosen difference dictated by the differential. We initialize a counter $T_{\tilde{K}_{T-1}}$ for each possible key guess $\tilde{K}_{T-1}$ to zeroes. Referring to Figure 2.5, as

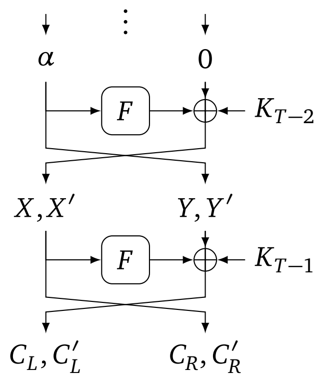


Figure 2.5: Recovery of round key K_{T-1} on SIMON

we can compute F, we may determine

$$Y \oplus Y' = F(C_R) \oplus F(C_R') \oplus C_L \oplus C_L', \tag{2.12}$$

and check whether this difference matches the difference α dictated by the differential. If this is the case, then the message pair is assumed to be a right pair. By trying all possible values $\tilde{K}_{T-1}$ for the last round key, we may partially decrypt a given ciphertext pair for one round to obtain the actual values of $X\|Y$ and $X'\|Y'$, under the assumption that $K_{T-1} = \tilde{K}_{T-1}$. Again, as we can evaluate F, we can check if

$$F(Y) \oplus F(Y') \oplus X \oplus X' \tag{2.13}$$

equals zero. If this is the case, then the current guess $\tilde{K}_{T-1}$ can be considered a candidate for the correct round key, and we increment the counter $T_{\tilde{K}_{T-1}}$.

Denoting the differential probability by p, the process above is repeated using cp^{-1} chosen message pairs, for some small constant c. In the end, a ranking of key candidates by their counter values $T_{\tilde{K}_{T-1}}$ provides the attacker with the most probable key guesses for the attacked last round key, as we described in Section 1.3. Next, we discuss the complexity of the attack described.

Complexity

As described in our introduction of differential cryptanalysis in Section 1.3, the data complexity of the classical differential attack is cp^{-1}, where p is the differential probability and c is a small constant. For the attack presented on SIMON32/64, the data complexity is $2^{29.48}$ chosen pairs for $T = 16$ rounds of the cipher, as shown in Table 2.5. As for the (online) time complexity, it is defined by the total number of encryption queries performed for all filtered pairs, using all possible $2^{n/2}$ values $\tilde{K}_{T-1}$ for the last round key, i.e. it is

$$\frac{c}{p} \cdot \gamma 2^{\kappa} \cdot \frac{2}{T}, \tag{2.14}$$

where γ is the probability that a pair survives the filtering, which is $2^{-n/2}$. This will yield a time complexity of $\frac{2c}{Tp}$ encryption query equivalents for SIMON variants. As for the memory needed for the key recovery attack in the presented cases, it equals the number of key guesses which is $2^{n/2}$ words of memory.

2.1.8 Impossible Differentials

Some applications of impossible differentials to Feistel schemes require that the round function F is a bijection. A prominent example of this is the general 5-round property presented by Knudsen in the proposal for the 128-bit block cipher DEAL [189]. However, the F function of SIMON is not a bijection, and indeed the impossible differentials we present in the following do not rely on it being so.

Previously, we saw how one can determine the feasible output differences of the F function of SIMON, using a fixed input difference, in the sense that we can determine the truncated output difference. We also saw that all possible output differences all occur with equal probability. We are interested in investigating for how many rounds a particular input difference can go before we are uncertain about *all output difference bits*, i.e. before $\star$ symbols appear on all positions so the difference is completely truncated. Intuitively, using an input difference of Hamming weight one will be the best approach, as each active bit in the input difference gives rise to 1, 2 or 3 active bits in the output difference (ignoring the possibility of cancellations, which is less predictable). For $n \in \{32, 48, 64\}$, we exhaustively tried all possible input differences and confirmed this intuition. For $n = 32$ and $n = 64$, there was another pattern of Hamming weight two, namely $0 \cdots 0101$, and any rotation of this, that covered equally many rounds in one direction, before being completely truncated. However, as there was no occurrence of both zeroes and ones in the last truncated difference, the resulting impossible differential would cover less rounds than when using a Hamming weight one input difference.

Table 2.6: Truncated differential pattern propagation for SIMON using block sizes $n \in \{32, 48, 64\}$, with an input difference $0 \cdots 01 \| 0 \cdots 0$

(a) $n = 32$

Rounds	Left halves	Right halves
0	0000000000000001	0000000000000000
1	0000000*000001*0	0000000000000001
2	00000**00001**0*	0000000*000001*0
3	000***0*01*****0	00000**00001**0*
4	0******1******0*	000***0*01*****0
5	****************	0******1******0*

(b) $n = 48$

Rounds	Left halves	Right halves
0	000000000000000000000001	000000000000000000000000
1	000000000000000*000001*0	000000000000000000000001
2	0000000*00000**00001**01	000000000000000*000001*0
3	00000**0000***0*01***0**	0000000*00000**00001**01
4	000***0*0**************1	00000**0000***0*01***0**
5	0***********************	000***0*0**************1
6	************************	0***********************

(c) $n = 64$

Rounds	Left halves	Right halves
0	00000000000000000000000000000001	00000000000000000000000000000000
1	00000000000000000000000*000001*0	00000000000000000000000000000001
2	000000000000000*00000**00001**01	00000000000000000000000*000001*0
3	0000000*00000**0000***0*01***0*0	000000000000000*00000**00001**01
4	00000**0000***0*0******1******0*	0000000*00000**0000***0*01***0*0
5	000***0*0**********************0	00000**0000***0*0******1******0*
6	0*****************************0*	000***0*0**********************0
7	********************************	0*****************************0*

Table 2.6 shows how the truncated differences progress over the rounds of SIMON for block

sizes $n \in \{32, 48, 64\}$. We refer to Appendix A for the remaining cases. All progressions use the same input difference $0\cdots01\|0\cdots0$. Other input differences of Hamming weight one yield a progression of truncated differences that are rotated correspondingly. Taking the $n = 32$ case as an example and referring to Table 2.6a, we see that after 5 rounds of SIMON, we have with probability $p = 1$ the truncated output difference

$$\texttt{****************}\|\texttt{0******1******0*}. \tag{2.15}$$

By left rotating the right half of this truncated difference by 7 or 9 positions, one of the 0-bits will be shifted to the position of the 1-bit. Due to the symmetry of decryption and encryption of the Feistel scheme, we find that this provides us with two impossibility properties,

$$\texttt{0001}\|\texttt{0000} \overset{\mathcal{E}}{\not\rightarrow} (\texttt{0001} \lll 7)\|\texttt{0000} \quad \text{and} \tag{2.16}$$

$$\texttt{0001}\|\texttt{0000} \overset{\mathcal{E}}{\not\rightarrow} (\texttt{0001} \lll 9)\|\texttt{0000}, \tag{2.17}$$

where $\mathcal{E}$ denotes $T = 10$ rounds of SIMON32/64. With this, we find two impossibility properties for each input difference of Hamming weight one, i.e. $n/2$ in total. This property for the rotation by $q = 7$ is depicted in Figure 2.6. In the further description of the attack, we denote by Q the set of indices for such rotations of the output difference, relative to the input difference, and hence $\sharp Q$ is the number of impossible differentials using one input difference. For example with SIMON32/64, we have $Q = \{7, 9\}$.

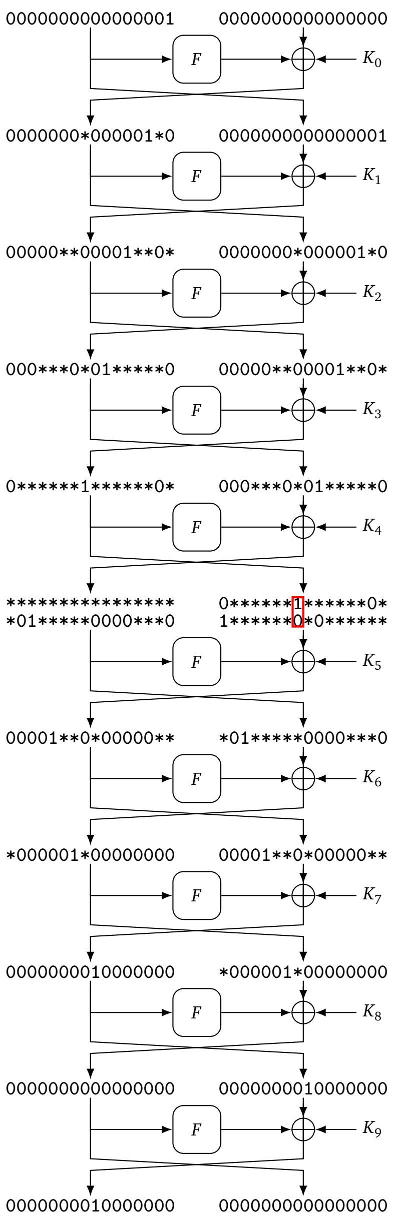


Figure 2.6: A 10-round impossible differential for SIMON32/64. Tracing truncated output differences in respectively forward and backward directions give a contradiction on the right half truncated mask after 5 rounds, where a 0-bit overlaps with a 1-bit.

Note that the analysis described so far uses an input difference of the form $\alpha||0$. Thus, the impossible differentials described in this section can trivially be extended by two rounds on top of probability one, as described in Section 2.1.6, yielding an extra 2 rounds for the impossible differential. Referring to Table 2.6, we see that for other values of n, we do not have both a 0-bit and 1-bit in the last truncated difference. Thus, we can not use this for obtaining an impossibility property, because we need to make a 0-bit overlap with a 1-bit to do so. We can, however, trace back to the last round where the truncated output difference on the right half contains a 1-bit, and match this up with the last truncated output difference containing a 0-bit. Of course, this forfeiture means the impossible differential covers less rounds.

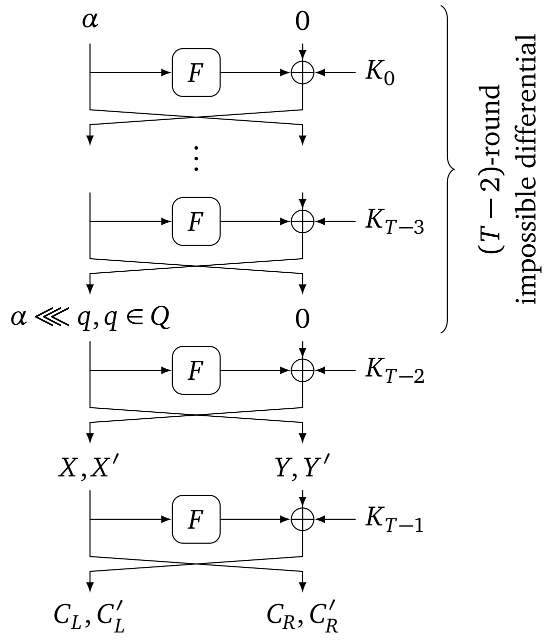


Figure 2.7: Key recovery attack with impossible differentials on SIMON

Key Recovery

As it was described for key recovery using a regular differential, we again attack a T-round variant of SIMON using a $(T-2)$-round property. We refer to Figure 2.7 for an illustration of the attack. Consider a pair of ciphertexts $C_L \| C_R$ and $C_L' \| C_R'$. The first filter in the recovery we can apply, is to test whether

$$\Gamma = F(C_R) \oplus F(C_R') \oplus C_L \oplus C_L' \tag{2.18}$$

equals the right half of one of the $\sharp Q$ impossible differentials, i.e. if it equals some $\alpha \lll q$, where $q \in Q$. If this is the case, we try all values $\tilde{K}_{T-1}$ of the last round key K_{T-1} and partially decrypt for one round to obtain the 1-round decrypted pair $(X\|Y)$ and $(X'\|Y')$. We may now test if

$$F(Y) \oplus F(Y') \oplus X \oplus X' \tag{2.19}$$

equals zero. If this is the case, then the candidate $\tilde{K}_{T-1}$ for K_{T-1} can be discarded forever as a possible last round key. After using many impossible differentials, our hope is that very few candidates for the last round key remain. The attack procedure is presented as Algorithm 4. Next, we turn to analyzing the complexity of the attack.

Algorithm 4: SIMON-IMPOSSIBLEDIFFERENTIALKEYRECOVERY

```
Data: A set of rotation indices $Q$, relative to input difference $\alpha$ yielding impossible
      differentials
Result: A set of remaining key candidates $\mathcal{S}$ for last round key
1  $\mathcal{S} \leftarrow \mathbb{F}_2^{n/2}$
2  Construct a "basis" $\mathcal{X}$ of $2^\ell$ messages
3  for $j = 0, \ldots, n/2-1$ do
4      $\alpha \leftarrow 0\cdots01 \lll j$
5      foreach $M \in \mathcal{X}$ do
6          $M' \leftarrow (M_L \oplus \alpha) \| \big(M_R \oplus F(M_L) \oplus F(M_L \oplus \alpha)\big)$
7          $C \leftarrow \mathcal{E}_K(M), C' \leftarrow \mathcal{E}_K(M')$
8          $\Gamma \leftarrow F(C_R) \oplus F(C'_R) \oplus C_L \oplus C'_L$
9          if $\Gamma \in \{\alpha \lll q \mid q \in Q\}$ then
10             foreach $\tilde{K}_{T-1} \in \mathcal{S}$ do          // Iterate over remaining key guesses
11                 $X \| Y \leftarrow C_R \| \big(F(C_R) \oplus C_L \oplus \tilde{K}_{T-1}\big)$
12                 $X' \| Y' \leftarrow C'_R \| \big(F(C'_R) \oplus C'_L \oplus \tilde{K}_{T-1}\big)$
13                 if $F(Y) \oplus F(Y') \oplus X \oplus X' = 0$ then
14                     $\mathcal{S} \leftarrow \mathcal{S} \setminus \{\tilde{K}_{T-1}\}$       // Remove impossible key candidate
15                 end
16             end
17         end
18     end
19 end
20 return $\mathcal{S}$
```

Complexity

In the following, we give our analysis of the key recovery complexity for the impossible differential attack, in terms of data, memory and time, given in terms of equivalent number of T-round encryption queries. During our analysis, we refer to the line numbers of Algorithm 4, as well as Eq. (2.18) and Eq. (2.19).

As the messages of the basis $\mathcal{X}$ are queried *once* and stored in memory, the data and memory complexity for line 2 is 2^ℓ data and 2^ℓ memory. By choosing $\mathcal{X}$ in a way that we avoid using a particular pair twice in the form of (M, M') *and* (M', M), the total number of plaintext pairs used for the attack is

$$\frac{n}{2} \cdot 2^\ell, \tag{2.20}$$

where the factor $n/2$ comes from the possible rotations of the left half of the input difference $\alpha = 0\cdots01 \lll j, j = 0, \ldots, n-1$. As the number of input differences we iterate over in line 3 is $n/2$, and $\sharp\mathcal{X} = 2^\ell$, the number of M' constructed and queried in lines 6 and 7 is $\frac{n}{2} \cdot 2^\ell$. These M' are used once and not stored in memory, hence the total memory complexity of the attack is 2^ℓ for storing $\mathcal{X}$, and the total data complexity is $2^\ell + \frac{n}{2} \cdot 2^\ell = \left(\frac{n}{2} + 1\right)2^\ell$.

Expected Number of Remaining Keys. When using a particular message pair (M, M') with corresponding ciphertext pair (C, C') in lines 6 through 17, we first check if the difference Γ matches one of the right halves of the $\sharp Q$ impossible differences. Assuming that Γ is uniformly distributed with probability mass function $2^{-n/2}$, the probability of entering the **if** statement of line 9 is

$$\frac{\sharp Q}{2^{n/2}}, \tag{2.21}$$

and as such, the expected number of pairs passing the filtering of Eq. (2.18) is

$$\frac{n}{2} \cdot 2^{\ell} \cdot \frac{\sharp Q}{2^{n/2}}. \tag{2.22}$$

Consider now a wrong guess $\tilde{K}_{T-1}$ for the last round key K_{T-1}. We know already that for the correct key K_{T-1}, the probability of the **if** statement of line 13 being true is zero, due to the miss-in-the-middle property of the impossible differential attack. However, under the assumption that for a wrong key guess $\tilde{K}_{T-1}$, the difference of Eq. (2.19) is uniformly distributed, the probability of discarding a wrong key, using a single pair, is $2^{-n/2}$, and thus the probability of *not* discarding it is $1 - 2^{-n/2}$. Assuming independence of the probabilities of discarding a wrong key, for each of the $\frac{n}{2} \cdot 2^{\ell}$ pairs, the expected number of remaining keys $\sharp\mathscr{S}$ after using all pairs is

$$\mathbb{E}[\sharp\mathscr{S}] = 2^{n/2}\left(1 - 2^{-n/2}\right)^{\frac{n}{2}2^{\ell}\sharp Q 2^{-n/2}}. \tag{2.23}$$

Time Complexity. For every pair used in lines 10 through 16, i.e. those pairs satisfying $\Gamma \in \{\alpha \lll q \mid q \in Q\}$, we must try as many keys as there are currently in $\mathscr{S}$. The fraction of the set $\mathscr{S}$ which is not discarded by using a single such pair equals the probability that some pair does not discard some wrong key. This probability is computed as

$$\begin{aligned}
1 - &\Pr[\text{wrong key } \tilde{K}_{T-1} \text{ discarded by some pair}] \\
&= 1 - \Pr[\text{pair discards } \tilde{K}_{T-1} \mid \Gamma \in \{\alpha \lll q \mid q \in Q\}] \cdot \Pr[\Gamma \in \{\alpha \lll q \mid q \in Q\}] \\
&= 1 - 2^{-n/2} \cdot \frac{\sharp Q}{2^{n/2}} \\
&= 1 - \frac{\sharp Q}{2^{n}}.
\end{aligned} \tag{2.24}$$

As such, the expected number of 1-round partial decryptions made during the course of the attack, using $\frac{n}{2}2^{\ell} = n2^{\ell-1}$ pairs, is determined as

$$\begin{aligned}
& 2^{n/2} + 2^{n/2} \cdot \left(1 - \frac{\sharp Q}{2^n}\right) + 2^{n/2} \cdot \left(1 - \frac{\sharp Q}{2^n}\right)^2 + \cdots + 2^{n/2} \cdot \left(1 - \frac{\sharp Q}{2^n}\right)^{\frac{n}{2}2^{\ell}-1} \\
= {} & 2^{n/2} \sum_{i=0}^{\frac{n}{2}2^{\ell}-1} \left(1 - \frac{\sharp Q}{2^n}\right)^i \\
= {} & 2^{n/2} \cdot \frac{1 - \left(1 - \frac{\sharp Q}{2^n}\right)^{\frac{n}{2}2^{\ell}}}{1 - \left(1 - \frac{\sharp Q}{2^n}\right)} \\
= {} & 2^{3n/2} \cdot \frac{1 - \left(1 - \frac{\sharp Q}{2^n}\right)^{\frac{n}{2}2^{\ell}}}{\sharp Q}.
\end{aligned} \tag{2.25}$$

Evaluating this expression numerically is computationally intensive for larger values of ℓ and n. For the numerator of Eq. (2.25), we can use the fact that

$$\lim_{x \to \infty} \left(1 - \frac{k}{x}\right)^x = e^{-k}. \tag{2.26}$$

We write 2^{ℓ} as $2^{\ell} = c2^n$ for some constant c. Then

$$\begin{aligned}
\lim_{2^n \to \infty} 2^{3n/2} \cdot \frac{1 - \left(1 - \frac{\sharp Q}{2^n}\right)^{\frac{n}{2}2^{\ell}}}{\sharp Q} &= 2^{3n/2} \cdot \frac{1 - e^{-\frac{1}{2}\sharp Q c n}}{\sharp Q} \\
&= 2^{3n/2} \cdot \frac{1 - e^{-\sharp Q n 2^{\ell-n-1}}}{\sharp Q}.
\end{aligned} \tag{2.27}$$

We use the approximation of Eq. (2.27), when computing Eq. (2.25) is too intensive. For the attack, the time complexity is determined as the total effort spent in the 1-round partial decryption phase, converted to the equivalents of T-round encryption queries. As such, the total complexity in terms of T-round encryptions equals the expression from either Eq. (2.25) or Eq. (2.27), scaled by $\frac{2}{T}$.

In Table 2.7 we present our results for recovering the last round key using impossible differentials, for all variants of SIMON. The results given all use the full code book, i.e. a data complexity of 2^n. Note from Table 2.7, that in the best case we expect to reduce the number of candidates for the last round key to 14.1%, while in the worst case we only reduce it to 38.29%. For the cryptanalyst, such reductions of the last round key space are not quite satisfying, and indeed we choose not to label the analyses as *attacks* per se. Nevertheless, we feel the analysis is interesting enough to merit inclusion in this section. In the original paper [19], we give figures for the complexities in the case that the number of remaining keys is expected to be 1% of the round key space, but we note that in this case, the data complexity exceeds that of the full code book.

Table 2.7: Results for key recovery attack on SIMON using $\sharp Q \cdot \frac{n}{2}$ impossible differentials. The complexities indicated with a † are computed using the approximation of Eq. (2.27).

	Rounds						
Cipher	Attacked	Total	Rem. keys	$\sharp Q$	Data	Memory	Time
SIMON32/64	14	32	15.22%	2	2^{32}	$2^{27.91}$	$2^{46.76}$
SIMON48/72	15	36	38.29%	1	2^{48}	$2^{43.36}$	$2^{71.30}$
SIMON48/96	15	36	38.29%	1	2^{48}	$2^{43.36}$	$2^{71.30}$
SIMON64/96	16	42	14.38%	2	2^{64}	$2^{58.96}$	†$2^{94.78}$
SIMON64/128	16	42	14.38%	2	2^{64}	$2^{58.96}$	†$2^{94.78}$
SIMON96/144	19	52	14.10%	2	2^{96}	$2^{90.39}$	†$2^{142.78}$

Experimental Verification

For the case $n = 32$, the block size is small enough that we may actually implement and verify the impossible differentials analysis. Thus, we provide in [18], among other cryptanalytic functionalities, our C++ implementation of the key-recovery attack on $T = 14$ rounds of SIMON32/64, using the 12-round impossible differential.

Table 2.8: Results from key recovery experiments on SIMON32/64 using the parameters of Table 2.7. The column $\sharp\mathscr{S}$ gives the number of remaining key candidates, while the last column gives the same in terms of percentage of $2^{n/2}$.

$\sharp\mathscr{S}$	Time (sec.)	Rem. keys
3805	1619	5.81%
789	1636	1.20%
2455	1655	3.75%
607	1615	0.93%
1600	1634	2.44%
344	1152	0.52%
1536	1190	2.34%
2937	1172	4.48%
3170	1268	4.84%
5259	1207	8.02%

In Table 2.8, we present the results of 10 experimental runs, the time for each run and the size of the output $\mathscr{S}$, with its corresponding percentage of the full round key space. Figure 2.8 shows how the size of $\sharp\mathscr{S}$ progressed over the course of the attack, when using different rotation amounts on the input difference. From our experiment, we actually observe that the attack gives fewer remaining key candidates than expected from our analysis and from Table 2.7.

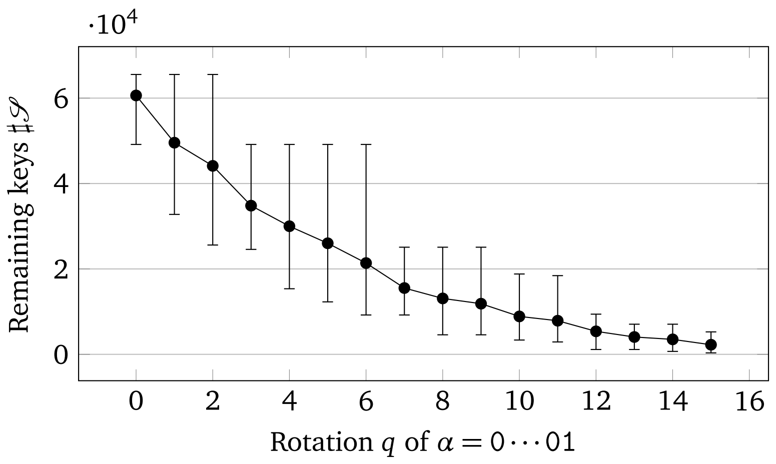


Figure 2.8: Progression of the average size of $\sharp\mathcal{S}$ for the key recovery attack on SIMON32/64 using the parameters of Table 2.7, as a function of the rotation amount on the input difference (input difference used is $\alpha = 0\cdots01 \lll j, j = 0,\ldots,n/2-1$). The progressions are from the experimental results of Table 2.8, and the error bars show the maximum respectively minimum number of keys remaining.

2.1.9 Connections to Linear Trails

Denote by $\Delta_{\{i_1,\ldots,i_k\}} \in \mathbb{F}_2^{n/2}$ a binary string of $n/2$ bits which has active bits in exactly the positions $i_1,\ldots,i_k$. In the following, we compute all indices modulo $n/2$. Consider a difference α input to the round function F. From Section 2.1.3 we know that if $\alpha_i = 1$, then this active bit toggles an active bit in position $i+2$ with probability one, and can also toggle bits in the output difference in positions $i+1$ and $i+8$, where each of the $2^2 = 4$ combinations occur with equal probability. As such, the observation can be described as

$$\begin{aligned} \mathsf{DP}_F(\Delta_{\{i\}},\Delta_{\{i+2\}}) &= 1/4 \\ \mathsf{DP}_F(\Delta_{\{i\}},\Delta_{\{i+1,i+2\}}) &= 1/4 \\ \mathsf{DP}_F(\Delta_{\{i\}},\Delta_{\{i+2,i+8\}}) &= 1/4 \\ \mathsf{DP}_F(\Delta_{\{i\}},\Delta_{\{i+1,i+2,i+8\}}) &= 1/4 \end{aligned}, \qquad 0 \le i < n/2. \tag{2.28}$$

If we analyze the F function with respect to linear cryptanalysis, we remark that bit i of an input can be approximated by a linear combination of bit $i-1$, $i-2$ and $i-8$, but always including bit $i-2$, much like for the differential characteristic. As such, we find that

$$\begin{aligned} \mathsf{Corr}_F(\Delta_{\{i\}},\Delta_{\{i-2\}}) &= 1/2 \\ \mathsf{Corr}_F(\Delta_{\{i\}},\Delta_{\{i-1,i-2\}}) &= 1/2 \\ \mathsf{Corr}_F(\Delta_{\{i\}},\Delta_{\{i-2,i-8\}}) &= 1/2 \\ \mathsf{Corr}_F(\Delta_{\{i\}},\Delta_{\{i-1,i-2,i-8\}}) &= -1/2 \end{aligned}, \qquad 0 \le i < n/2. \tag{2.29}$$

We remark the connection between the differential equations of Eq. (2.28) and the linear approximations of Eq. (2.29): there is a correspondence (in order) of each of the four equations.

This observation allows us to turn any differential characteristic for SIMON into a corresponding linear trail. Define a function

$$\begin{aligned} \mathsf{act} : \mathbb{F}_2^{n/2} &\to \mathscr{P}(\mathbb{Z}_{n/2}) \\ X &\mapsto \{i \mid X_i = 1, i \in \mathbb{Z}_{n/2}\}, \end{aligned} \tag{2.30}$$

where $\mathscr{P}$ denotes the power set. In words, $\mathsf{act}(X)$ is the set of active bit indices of X. We give as Algorithm 5 the procedure for converting a differential characteristic in SIMON to a linear trail.

Algorithm 5: SIMON-DIFFERENTIALCHARACTERISTICTOLINEARTRAIL

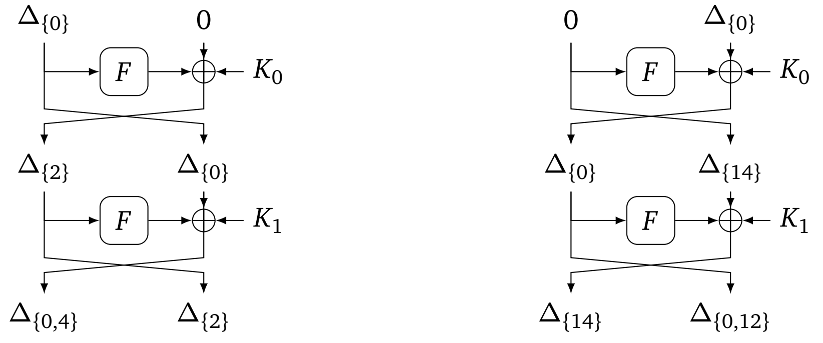
```
   Data: $T$-round differential characteristic $\left(\alpha_0^L \| \alpha_0^R, \dots, \alpha_T^L \| \alpha_T^R\right)$
1  for $t = 0, \dots, T$ do
2      $S_t^L \leftarrow \mathsf{act}(\alpha_t^R)$,    $S_t^R \leftarrow \mathsf{act}(\alpha_t^L)$          // Note the swapping of left and right
3  end
4  if $\mathsf{act}(\alpha_0^R) \neq \emptyset$ then $a \xleftarrow{\$} \mathsf{act}(\alpha_0^R)$ else $a \xleftarrow{\$} \mathsf{act}(\alpha_0^L)$
5  Pick $b \in \{0, \dots, n/2-1\}$
6  for $t = 0, \dots, T$ do                                              // Adjust indices
7      $S_t^L \leftarrow \{b - s + a \bmod n/2 \mid s \in S_t^L\}$
8      $S_t^R \leftarrow \{b - s + a \bmod n/2 \mid s \in S_t^R\}$
9  end
10 for $t = 0, \dots, T$ do
11     $\gamma_t^L \leftarrow \Delta_{\mathsf{act}(S_t^L)}$,    $\gamma_t^R \leftarrow \Delta_{\mathsf{act}(S_t^R)}$
12 end
13 return $\left(\gamma_0^L \| \gamma_0^R, \dots, \gamma_T^L \| \gamma_T^R\right)$
```

Figure 2.9 gives a small example of converting a 2-round differential characteristic for SIMON32/64 into a corresponding linear trail, using Algorithm 5, where we use the values $a = 0$ and $b = 0$. Note that in the differential characteristic, we use two times the first relation of Eq. (2.28) and for the linear trail we use two times the first relation of Eq. (2.29). As such, with the linear trail, we approximate two round key bits using the system of equations

$$\begin{aligned} X_{14}^2 \oplus K_{14}^1 \oplus Y_{14}^1 \oplus X_0^1 &= 0 \\ Y_{14}^1 \oplus X_{14}^0 &= 0 \\ X_0^1 \oplus K_0^0 \oplus Y_0^0 \oplus X_2^0 &= 0, \end{aligned} \tag{2.31}$$

where $X^t \| Y^t$, $0 \leq t < T$, denotes the input to round t, $X^T \| Y^T$ denotes the output from the last round, and K^t denotes the round key of round t. Combining these equations, we finally obtain the equation

$$X_2^0 \oplus X_{14}^0 \oplus Y_0^0 \oplus X_{14}^2 = K_0^0 \oplus K_{14}^1. \tag{2.32}$$



(a) Differential characteristic **(b)** Converted linear trail

Figure 2.9: Sample 2-round differential characteristic for SIMON32/64, i.e. with $n = 32$ (left) and the corresponding converted linear trail (right)

2.1.10 Timeline of Cryptanalysis on SIMON

When the majority of the results presented in this section were made public, it was among the first cryptanalytic results available on the block cipher SIMON. Indeed, our work [19] was published on the ePrint archive the same day as that of Abed, List, Lucks, and Wenzel [5]. Unsurprisingly, due to the high-profile nature of SIMON, it has received much cryptanalytic attention since. We provide here a brief summary of such work, up until the time of writing.

In [5], the authors provide the first differential cryptanalysis on SIMON on up to 14 rounds of SIMON32/64, as well as results on other variants. Building on [63] (see below), the authors updated their results in a later version, and are able to break up to 18 rounds of SIMON32/64. Furthermore, results using linear cryptanalysis were included, but breaking fewer rounds.

Combining the work of [5], Abed, List, Lucks, and Wenzel present in [7] results on both SIMON and SPECK, breaking e.g. up to 18 rounds on SIMON32/64. Parallel to this work, Biryukov, Roy, and Velichkov employ in [63] a technique termed *threshold search*, an automated search method for differential characteristics in ARX ciphers, to SIMON and SPECK. In their paper, the results of [5] are improved, so e.g. a differential attack on 19 rounds of SIMON32/64 is provided. Furthermore, methods to efficiently determine the differential probability of the AND operation was presented, and the differential effect in SIMON was studied further.

In [291], Tupsamudre, Bisht, and Mukhopadhyay show, quite unsurprisingly, that SIMON is susceptible to fault attacks. In particular, the last round key can be recovered, assuming $n/2$ bit-faults or $n/8$ byte faults, depending on the model.

Wang, Wang, Jia, and Zhao give in [293] new differential attacks using a new *dynamic key-guessing technique*, to provide improved results on all SIMON variants, e.g. breaking up to 21 rounds of SIMON32/64.

In [2] by Abdelraheem, Alizadeh, Alkhzaimi, Aref, Bagheri, Gauravaram, and Lauridsen, the linear analysis provided in [16] is improved upon, in particular using correlation matrix techniques. Improved results on a range of SIMON variants are exhibited, including linear hulls which cover up to 21 rounds of SIMON32/64.

Shi, Hu, Sun, Song, Qiao, and Ma provide in [285] *mixed-integer linear programming* (MILP) models to determine differentials and characteristics in SIMON variants, thereby improving on the first results presented in [2] (the latter was later updated to match the number of rounds broken by [285]). This work is in fact somewhat related to what we present in Section 3.1, where we will be employing MILP models to determine bounds on resistance of certain primitives against differential- and linear attacks.

In [84, 85], Boura, Naya-Plasencia, and Suder provide a new generic approach to impossible differential cryptanalysis of block ciphers. The work is motivated by the fact that, due to the complicated nature of impossible differentials, results using these attack vectors often provide incorrect analyses of complexities. With respect to SIMON, the authors point out that our analysis using impossible differentials above use a data complexity exceeding the full code book. While we pointed this out in Section 2.1.8, it mistakenly went unremarked in the original publication [19]. With respect to concrete results on SIMON, the authors provide impossible differential attacks on all variants, covering e.g. 19 rounds of SIMON32/64 and 30 rounds of SIMON128/256. We remark that the impossible differential used for the result on SIMON32/64 is exactly equal to the one we describe in Section 2.1.8 (see [85, Appendix A.3]), but using additional tricks to reduce the data complexity and cover more rounds.

Parallel to [84, 85], the work [288] by Sun, Hu, Wang, Qiao, Ma, and Song uses again MILP models to determine differentials and characteristics in SIMON48/72, among others. With respect to SIMON48/72 and SIMON48/96, the authors provide a 15-round differential characteristic of probability 2^{-53} and a differential of probability $2^{-41.96}$. An incremental work by Sun, Hu, Wang, Wang, Qiao, Ma, Shi, Song, and Fu is given in [289].

In [294], Wang, Liu, Varici, Sasaki, Rijmen, and Todo present impossible differential cryptanalysis of SIMON variants, but perhaps more interestingly, for the first time, zero-correlation attacks (which we mentioned briefly in Section 1.3.6) and *integral attacks* are applied to the SIMON family. Among the most interesting results is the zero-correlation attack covering 20 rounds of SIMON32/64 and the integral attack covering 21 rounds of SIMON32/64.

Interestingly, going away from the trend of solely differential- and linear cryptanalysis, Ahmadian, Rasoolzadeh, Salmasizadeh, and Aref give in [10] results on SIMON32/64 using *dynamic cube attacks*, a variant of algebraic attacks described in Section 1.3.8. While this approach is able to break up to 18 rounds only, the attack vector is nevertheless an interesting take on the analysis of SIMON.

In [115], Dinur, Dunkelman, Gutman, and Shamir consider ways to analyze the difference distribution table of a function, when it can not be completely determined, e.g. when the block size is too large. One of the methods used in the paper is the same as the one described in Section 2.1.2. With respect to SIMON, the authors consider existing differential characteristics of [7, 63, 288], and show that the characteristic probabilities can be improved, using the techniques presented.

Ashur provides in [31] a new approach to computing the correlation of short linear hulls in SIMON. This approach is used to iterate several such linear hulls into a single longer *super-trail* which builds on those of [2]. The correlation approximation of this trail is better than those previously considered, albeit still not tight.

In [94], Chen, Wang, and Wang provide impossible differential attacks on 19 rounds of SIMON32/64 and 20 rounds of SIMON48/72. The attacks use the full code book and time complexities corresponding to about $2^{58.9}$ and $2^{71.2}$, respectively.

Attacks on up to 26 rounds of SIMON64/128 using truncated differentials are described by Mourouzis, Song, Courtois, and Christofii in [238]. For that particular member of the SIMON family, their attacks cover two more rounds than our work described in this section, using a slightly different attack vector.

In [92], Chen and Wang re-use the dynamic key-guessing technique of [293], this time in the framework of linear cryptanalysis, to present improved linear hulls for SIMON. For example, their new linear hull covers up to 23 rounds of SIMON32/64, and up to 53 rounds of SIMON128/256.

The most recent work on SIMON, at the time of writing, is that of Kölbl, Leander, and Tiessen [199]. Here, the authors consider a generalized class of the SIMON round function, and provide an exact analysis of such a function with respect to differential- and linear cryptanalysis. Furthermore, a SAT/SMT solver is employed to search for optimal differential characteristics and linear trails in SIMON. The authors also build on our discussion of the differential effect in SIMON, and an analysis of the choice of rotation constants for the SIMON round function is undertaken.

2.1.11 Discussion and Conclusions

In this section, we have taken some of the concepts introduced in Section 1.3, and applied them to the lightweight block cipher SIMON. In particular, we considered properties of the SIMON round function, especially with respect to the difference distribution table, in order to present some very basic differential characteristics. We then went on to describe a search heuristic, based on a graph theoretic description of characteristics in SIMON, and used this to determine the differentials found that give our best attacks, c.f. Table 2.5. After touching upon the observed differential effect in SIMON32/64, a fact which is further investigated in [199], we used truncated differential characteristics to construct impossible differentials over several rounds of SIMON variants. Meanwhile, our analysis of the complexities show, that using our basic observations and the full code book for the attack, the remaining number of candidates for the last round key is still expected to be as low as 14.1% for SIMON96/144, but as high as 38.29% for SIMON48/72 and SIMON48/96. While the results on impossible differentials given in [19] reduce the expected number of remaining keys to 1% of the space of the last round key, they use a data complexity exceeding the full code book, which was pointed out and corrected in [84, 85]. Due to the tremendous amount of papers giving cryptanalytic results on SIMON, in a relatively short time frame, we gave also a comprehensive overview of such results, up until the time of writing.

With the discussion of the cryptanalysis performed on SIMON since 2013, it should be clear that the attacks presented in this section are not the best published, neither do we claim them so. While it seems that small improvements are constantly be made to attack complexities, and even sometimes increase the number of rounds being attacked, no analysis so far is close to breaking any full variant of the SIMON family, and indeed SIMON seems very solid. Some people in the cryptographic community argue that we should stop analyzing SIMON. For example, Appelbaum said [29]:

> SIMON and SPECK should not even be reviewed by anyone in the community, because it dignifies [the designs] and wastes the cycles – the brain cycles – of intelligent people, by going to look at a thing that is produced by a bad actor agency [(the NSA)].

On this, we take the opposite standpoint: as the specification [40] does not provide much in terms of design criteria and cryptanalysis (and especially for this reason), we believe that analytic results from the community are essential in order to shed light on those aspects of the cipher. This is particularly important if the cipher is proposed for standardization, should such considerations become relevant. With the work presented herein, we believe we have taken a good first step towards better understanding the security of SIMON.

With respect to future work on SIMON, much is still desired. While most works focus on providing better differential- or linear cryptanalysis, or both, some works such as [199] strive to bring forth an understanding of the design choices made. We believe much more of such work is required to draw a fulfilling picture of why SIMON looks the way it does. This is especially true with respect to the SIMON key schedule algorithm, which has been left mostly untouched by cryptanalysts thus far.

2.2 Cryptanalysis of PRESENT

In Section 1.2.5, we saw how compression functions and hash functions can be constructed from a block cipher. The approaches we mentioned involved using one of three compression function constructions: Davies-Meyer, Matyas-Meyer-Oseas or Miyaguchi-Preneel, and iterating it using the Merkle-Damgård construction. In Section 1.3, we discussed what adversarial goals an attacker may want to obtain, and under what assumptions she can do so. We also introduced the topic of linear cryptanalysis, a way of analyzing cipher by utilizing linear approximations to the non-linear components of the cipher. In this section, we combine all of the things mentioned. In particular, we consider the block cipher PRESENT in the key-less setting with the adversarial goal of providing a distinguisher for this cipher. As we shall see when we describe the motivation in more detail below, our analysis is applicable if one uses the PRESENT block cipher in the Matyas-Meyer-Oseas construction for a compression function. While this describes the application itself, the major part of the contribution in this section is the first ever definition of a meaningful framework, in which linear cryptanalysis can be used to define a distinguisher in the key-less setting. We describe this framework, or *model* as we call it, in detail in Section 2.2.2, while Section 2.2.5 describes the application of the model to PRESENT.

Publication

The results presented in this section are from:

[207] Martin M. Lauridsen and Christian Rechberger. Linear Distinguishers in the Key-less Setting: Application to PRESENT. In Leander [210], pages 217–240.

Author Contribution

The author contributed towards defining the model for linear distinguishers in the key-less setting, which we present in Section 2.2.2. Furthermore, the author contributed towards applying the key-less linear distinguisher to the standardized block cipher PRESENT in Section 2.2.5, by using previous results in conjunction with the model and further analysis. The author is responsible for the C++ code for aid of analysis and experimental verification, available as [208].

2.2.1 Motivation

Even though block ciphers have been used for a very long time, either implicitly or explicitly, to construct hash functions, a separate study of the security of block ciphers where the key is either known or under control of the adversary, has started only recently. Knudsen and Rijmen proposed so-called known-key distinguishers [191]. Later Biryukov, Khovratovich, and Nikolic [60] and Lamberger, Mendel, Rechberger, Rijmen, and Schläffer [203] proposed open- or chosen-key models to evaluate the security of block ciphers.

Even though these models often exhibit a rather contrived looking property, and evade a formally rigorous definition[1] (a property they share with collision attacks), cryptanalysts largely agree that these distinguishers are useful and interesting. Indeed, techniques developed to improve the original known-key distinguishers from [191], such as the rebound attack, later led to collision attacks on various hash functions by Khovratovich, Naya-Plasencia, Röck, and Schläffer [184], Mendel, Rechberger, Schläffer, and Thomsen [226] and [203]. Also, the findings in the open-key model from [60] were later used to find the first related-key key-recovery attacks on AES-256 and AES-192 in works by Biryukov, Dunkelman, Keller, Khovratovich, and Shamir [61] and Biryukov and Khovratovich [58].

Sometimes distinguisher descriptions are merely motivated by the fact that they *can* be formulated, as e.g. the 7-round known-key distinguisher on AES from [191], where byte-level zero-sums are used as a distinguishing property. Another example is the rotational rebound attack on reduced Skein by Khovratovich, Nikolic, and Rechberger [185], where the existence of *rotational collisions with errors* is defined as a distinguishing property. Sometimes, however, they are better motivated, e.g. by the construction of near-collisions or the subspace distinguishers of [203, 204] or limited-birthday distinguishers by Gilbert and Peyrin [142], that resemble some generalization of the concept of near-collisions.

The distinguisher we propose below comes with a new motivation that stems from pre-image attacks on hash functions or compression functions[2]. As an example, consider the compression function construction using a single call to a block cipher in Matyas-Meyer-Oseas mode. As we saw in Section 1.2.5, the i^{th} message block M_i is compressed by using it as the plaintext input when computing the next chaining value H_{i+1} using H_i as the cipher key, i.e. $H_{i+1} = E_{H_i}(M_i) \oplus M_i$. If an attacker can determine a relation stating that the jth bit of M_i equals the jth bit of $E_{H_i}(M_i)$ with a high probability, then it is likely that the jth bit of H_{i+1} equals zero. In a pre-image attack, if the target pre-image is zero at position j, this then leads to an advantage over brute-force search.

2.2.2 A Model for Linear Distinguishers in the Key-less Setting

In the following, we give our definition of a model for key-less linear distinguishers. Essentially, the model captures the possibility of distinguishing any block cipher in the key-less setting, given that a linear relation (in the form of a linear hull) of sufficiently high absolute correlation for a reasonable fraction of the key space $\mathbb{F}_2^{\kappa}$, is available. The notions of Definitions 22 and 23

[1]One exception being the work of Andreeva, Bogdanov, and Mennink [22]

[2]We emphasize here that the application to PRESENT in Section 2.2.5 will not be a pre-image attack

are largely inspired by the recent work of Gilbert on pushing further known-key attacks on the AES [141].

The following definition of *α-separability* formalizes how a linear relation, combined with a set of inputs for a permutation $\pi : \mathbb{F}_2^n \to \mathbb{F}_2^n$, can exhibit a *significant* deviation from the behavior of a random permutation.

Definition 22 (α-separability). *Let $\mathscr{P}$ be a set of permutations from $\mathbb{F}_2^n$ to $\mathbb{F}_2^n$ and let $\pi \in \mathscr{P}$ denote a particular, fixed permutation from $\mathscr{P}$. Let $\mathscr{S} \subseteq \mathbb{F}_2^n$ be a subset of size Λ and let $\delta, \gamma \in \mathbb{F}_2^n \backslash \{(0,\ldots,0)\}$ be non-zero linear masks.*

*Without checking each input, each $X_i \in \mathscr{S}$ has an (a priori) associated probability $p_i = \Pr\left[\mathscr{R}^{\pi}_{\delta,\gamma}(X_i) = 1\right]$ that the linear relation is satisfied for that particular input. Let $\mathscr{X} = \sharp\left\{X \in \mathscr{S} \mid \mathscr{R}^{\pi}_{\delta,\gamma}(X) = 1\right\}$, then $\mathbb{E}[\mathscr{X}] = \sum_{i=1}^{\Lambda} p_i$. We say that the tuple $(\mathscr{P}, \pi, \mathscr{S}, \mathscr{R}^{\pi}_{\delta,\gamma})$ is α-*separable if and only if

$$\Pr_{\pi}\left[\left|\mathbb{E}[\mathscr{X}] - \frac{\Lambda}{2}\right| \geq \sqrt{\Lambda}\right] \geq \alpha, \tag{2.33}$$

where the probability is taken over $\pi \in \mathscr{P}$.

Definition 23 ($(\mathscr{T}, \Lambda, \alpha)$-intractability). *Let $\mathscr{P}$ be a set of permutations from $\mathbb{F}_2^n$ to $\mathbb{F}_2^n$ and let $\pi \in \mathscr{P}$ denote a particular, fixed permutation from $\mathscr{P}$. Let $\mathscr{S} \subseteq \mathbb{F}_2^n$ be a subset of size Λ and let $\delta, \gamma \in \mathbb{F}_2^n \backslash \{(0,\ldots,0)\}$ be non-zero linear masks. We say that the tuple $(\mathscr{P}, \pi, \mathscr{S}, \mathscr{R}^{\pi}_{\delta,\gamma})$ is* $(\mathscr{T}, \Lambda, \alpha)$-intractable if and only if *it is impossible, for any algorithm $\mathscr{A}$ to*

1. *Commit to a choice of non-zero linear masks $\delta', \gamma' \in \mathbb{F}_2^n \backslash \{(0,\ldots,0)\}$, and*

2. *When given access to a fixed pair Π, Π^{-1} with $\Pi \xleftarrow{\$} \mathsf{Perm}(n)$, construct a set $\mathscr{S}'$ of size Λ in time $\mathscr{T}$, s.t. the tuple $(\mathsf{Perm}(n), \Pi, \mathscr{S}', \mathscr{R}^{\Pi}_{\delta',\gamma'})$ is α-separable.*

Note 2. *For our distinguisher model, the notion of* one time unit *corresponds to a single evaluation of the respective permutation or its inverse.*

With the definitions of α-separability and $(\mathscr{T}, \Lambda, \alpha)$-intractability in hand, we are ready to formulate our proposed key-less linear distinguisher.

Definition 24 (Key-less linear distinguisher). *Let $\mathscr{E} : \mathbb{F}_2^{\kappa} \times \mathbb{F}_2^n \to \mathbb{F}_2^n$ be a block cipher and let $\hat{\mathscr{E}} = \{\mathscr{E}_K \mid K \in \mathbb{F}_2^{\kappa}\}$ denote the set of permutations due to choices of the key $K \in \mathbb{F}_2^{\kappa}$. Let $\mathscr{E}_K$ denote* some *fixed permutation from $\hat{\mathscr{E}}$. Fix non-zero linear masks $\delta, \gamma \in \mathbb{F}_2^n \backslash \{(0,\ldots,0)\}$ and let $\mathscr{A}$ be an algorithm producing in time $\mathscr{T}$ a set $\mathscr{S} \subseteq \mathbb{F}_2^n$ of size Λ. Then the tuple $(\mathscr{A}, \hat{\mathscr{E}}, \mathscr{E}_K, \mathscr{S}, \mathscr{T}, \mathscr{R}^{\mathscr{E}_K}_{\delta,\gamma}, \alpha)$ is said to be a* key-less linear distinguisher if and only if *$(\hat{\mathscr{E}}, \mathscr{E}_K, \mathscr{S}, \mathscr{R}^{\mathscr{E}_K}_{\delta,\gamma})$ is both α-separable and $(\mathscr{T}, \Lambda, \alpha)$-intractable.*

Note 3. *In all of the definitions above, the fixed linear masks $\delta, \gamma \in \mathbb{F}_2^n \backslash \{(0,\ldots,0)\}$ are* chosen *by the algorithm $\mathscr{A}$, but the choice must be made* before *the production of the input set $\mathscr{S}$ commences. In the context of distinguishing a block cipher, the adversary commits to δ and γ and then obtains access to $\mathscr{E}_K$ upon which the production of the subset $\mathscr{S}$ in time $\mathscr{T}$ begins. The parameter α directly expresses a lower bound on the fraction of the permutations $\pi \in \mathscr{P}$ for which the key-less linear distinguisher is valid. The time $\mathscr{T}$ allowed to construct $\mathscr{S}$ is a parameter chosen by the adversary.*

Analysis

In the following, we analyze and argue that the key-less linear distinguisher is meaningful. First, informally, the notion of α-separability expresses that for a concrete permutation $\pi : \mathbb{F}_2^n \to \mathbb{F}_2^n$, one can provide a linear relation which captures, for some constructed set of inputs, a *significant non-random behavior* in a permutation which is supposed to behave randomly. The *significant* part is captured by the requirement that the number of inputs satisfying the relation $\mathscr{R}_{\delta,\gamma}^{\pi}$ should deviate from what is expected in the ideal case by at least $\sqrt{\Lambda}$. This reflects the usual requirement in linear cryptanalysis, that the data complexity is inversely proportional to the squared correlation, as we saw in Section 1.3.6. Second, on top of that, Definition 23 captures the notion that for a random permutation $\Pi \xleftarrow{\$} \mathsf{Perm}(n)$, it should not be possible, in the same amount of time, to provide such a relation with a set of inputs which exhibits the same significant non-random behavior.

With respect to Definition 23, one of the components to analyzing our proposed key-less linear distinguisher is to answer the following question: what is the *upper bound* on the probability α' that an algorithm $\mathscr{A}$, when given access to the fixed pair Π and Π^{-1}, can produce in time $\mathscr{T}$ a set $\mathscr{S}' \subseteq \mathbb{F}_2^n$ of size Λ, together with a pre-determined relation $\mathscr{R}_{\delta,\gamma}^{\Pi}$, such that $(\mathsf{Perm}(n), \Pi, \mathscr{S}', \mathscr{R}_{\delta,\gamma}^{\Pi})$ is α'-separable? Our analysis answers this question in the following, and it implicitly provides a *lower bound* on α for when a concrete permutation $\pi : \mathbb{F}_2^n \to \mathbb{F}_2^n \in \mathscr{P}$ (in the notation of Definitions 22 and 23) can be shown to be $(\mathscr{T}, \Lambda, \alpha)$-intractable, for fixed $\mathscr{T}$ and Λ. We begin our analysis with Lemma 1.

Lemma 1. *In the notation of Definition 23, let $\delta', \gamma' \in \mathbb{F}_2^n \backslash \{(0, \dots, 0)\}$ be fixed non-zero linear masks, and let then an algorithm $\mathscr{A}$ be given access to Π, Π^{-1}, where $\Pi \xleftarrow{\$} \mathsf{Perm}(n)$. The optimal way for $\mathscr{A}$ to construct $\mathscr{S}' \subseteq \mathbb{F}_2^n$ of size Λ in time $\mathscr{T}$ is the following:*

1. *Construct an arbitrarily chosen set $\mathscr{Q} \subseteq \mathbb{F}_2^n$ of size $\mathscr{T}$*

2. *Partition $\mathscr{Q}$ into $\mathscr{Q}_1 = \left\{X \in \mathscr{Q} \mid \mathscr{R}_{\delta',\gamma'}^{\Pi}(X) = 1\right\}$ and $\mathscr{Q}_0 = \left\{X \in \mathscr{Q} \mid \mathscr{R}_{\delta',\gamma'}^{\Pi}(X) = 0\right\}$ by querying $\Pi(X)$ for all $X \in \mathscr{Q}$ (this has time complexity $\mathscr{T}$).*

3. *Set $\mathscr{S}'$ equal to the larger of the sets $\mathscr{Q}_0$ and $\mathscr{Q}_1$.*

4. *Fill up $\mathscr{S}'$ with arbitrarily chosen inputs from $\mathbb{F}_2^n \backslash \mathscr{Q}$ until $\sharp\mathscr{S}' = \Lambda$.*

Proof. As $\Pi \xleftarrow{\$} \mathsf{Perm}(n)$, the particular choice of $\delta', \gamma' \in \mathbb{F}_2^n \backslash \{(0, \dots, 0)\}$ does not affect the analysis. The most information $\mathscr{A}$ can learn about Π in time $\mathscr{T}$ is to obtain $\mathscr{T}$ pairs $(X, \Pi(X))$, as is done when determining $\mathscr{Q}$ and its image under Π. In order to optimally shift the balance of the expected number of inputs of $\mathscr{S}'$ satisfying $\mathscr{R}_{\delta',\gamma'}^{\Pi}$ away from $\Lambda/2$, $\mathscr{A}$ should take the larger of $\mathscr{Q}_1$ and $\mathscr{Q}_0$ and pool it with randomly chosen inputs $X \in \mathbb{F}_2^n$ for which the value of $\mathscr{R}_{\delta',\gamma'}^{\Pi}(X)$ is not known. □

Continuing our analysis, assuming an algorithm $\mathscr{A}$ constructs $\mathscr{S}'$ as in Lemma 1, we determine an upper bound on the value α' as a function of Λ and $\mathscr{T}$, such that the resulting tuple $(\mathsf{Perm}(n), \Pi, \mathscr{S}', \mathscr{R}_{\delta',\gamma'}^{\Pi})$ is α'-separable. We give this result in Theorem 1.

Theorem 1 (Generic success probability)**.** *Let $\mathscr{A}, \Pi, \delta', \gamma', \mathscr{S}'$ and $\mathscr{T}$ be as in Lemma 1, where $\mathscr{T} \leq 4\sqrt{\Lambda}$, and let $\mathscr{X} = \sharp\{X \in \mathscr{S}' \mid \mathscr{R}^{\Pi}_{\delta,\gamma}(X) = 1\}$. Then*

$$\Pr\left[\left|\mathbb{E}[\mathscr{X}] - \frac{\Lambda}{2}\right| \geq \sqrt{\Lambda}\right] = 2^{-\mathscr{T}} \cdot \left[\sum_{k=0}^{\mathscr{T}-2\sqrt{\Lambda}} \binom{\mathscr{T}}{k} + \sum_{k=2\sqrt{\Lambda}}^{\mathscr{T}} \binom{\mathscr{T}}{k}\right]. \tag{2.34}$$

Proof. First, note that $\sharp\mathscr{Q}_1 \sim \mathscr{B}\left(\mathscr{T}, \frac{1}{2}\right)$. We want to determine the probability that we have $\left|\mathbb{E}[\mathscr{X}] - \frac{\Lambda}{2}\right| \geq \sqrt{\Lambda}$. The consideration is split into two cases depending on whether or not $\sharp\mathscr{Q}_1 \geq \mathscr{T}/2$.

Case $\sharp\mathscr{Q}_1 \geq \mathscr{T}/2$. In this case, we know that at least $\sharp\mathscr{Q}_1$ of the Λ inputs satisfy the relation. Thus, $\mathbb{E}[\mathscr{X}] = \mathbb{E}[Z] + \sharp\mathscr{Q}_1$ where $Z \sim \mathscr{B}\left(\Lambda - \sharp\mathscr{Q}_1, \frac{1}{2}\right)$. As such, $\mathbb{E}[\mathscr{X}] = (\Lambda + \sharp\mathscr{Q}_1)/2$, and the requirement $\left|\mathbb{E}[\mathscr{X}] - \frac{\Lambda}{2}\right| \geq \sqrt{\Lambda}$ is equivalent to either $\sharp\mathscr{Q}_1 \geq 2\sqrt{\Lambda}$ or $\sharp\mathscr{Q}_1 \leq -2\sqrt{\Lambda}$, the latter not being possible as $\sharp\mathscr{Q}_1$ is non-negative.

Case $\sharp\mathscr{Q}_1 < \mathscr{T}/2$. In this case, we know that there are at least $\mathscr{T} - \sharp\mathscr{Q}_1$ of the Λ inputs that *do not* satisfy the relation. Thus, $\mathbb{E}[\mathscr{X}] = \mathbb{E}[Z]$ where $Z \sim \mathscr{B}\left(\Lambda - \mathscr{T} + \sharp\mathscr{Q}_1, \frac{1}{2}\right)$. As such, $\mathbb{E}[\mathscr{X}] = (\Lambda - \mathscr{T} + \sharp\mathscr{Q}_1)/2$, and the requirement $\left|\mathbb{E}[\mathscr{X}] - \frac{\Lambda}{2}\right| \geq \sqrt{\Lambda}$ is equivalent to either $\sharp\mathscr{Q}_1 \geq \mathscr{T} + 2\sqrt{\Lambda}$ or $\sharp\mathscr{Q}_1 \leq \mathscr{T} - 2\sqrt{\Lambda}$, the former not being possible as $\sharp\mathscr{Q}_1 \leq \mathscr{T}$.

In both cases considered, there is one event which makes the inequality $\left|\mathbb{E}[\mathscr{X}] - \frac{\Lambda}{2}\right| \geq \sqrt{\Lambda}$ true. The combined probability of those two events is

$$\Pr\left[\sharp\mathscr{Q}_1 \geq 2\sqrt{\Lambda}\right] + \Pr\left[\sharp\mathscr{Q}_1 \leq \mathscr{T} - 2\sqrt{\Lambda}\right] = 2^{-\mathscr{T}} \cdot \left[\sum_{k=0}^{\mathscr{T}-2\sqrt{\Lambda}} \binom{\mathscr{T}}{k} + \sum_{k=2\sqrt{\Lambda}}^{\mathscr{T}} \binom{\mathscr{T}}{k}\right], \tag{2.35}$$

and from this, the result follows. □

Note 4. *The requirement $\mathscr{T} \leq 4\sqrt{\Lambda}$ in the statement of Theorem 1 arises because otherwise the two sums would overlap and add the same terms twice. The probability which is derived as a function of Λ and $\mathscr{T}$ provides a lower bound on α for when, in the notation of Definition 23, a tuple $(\mathscr{P}, \pi, \mathscr{S}, \mathscr{R}^{\pi}_{\delta,\gamma})$ can be $(\mathscr{T}, \Lambda, \alpha)$-intractable. By using the normal approximation of $\sharp\mathscr{Q}_1$, i.e. $\sharp\mathscr{Q}_1 \sim \mathscr{N}\left(\frac{\mathscr{T}}{2}, \frac{\mathscr{T}}{4}\right)$, one obtains a very precise and easily-computable approximation of the probability as*

$$1 - \Phi\left(\mathscr{N}\left(\frac{\mathscr{T}}{2}, \frac{\mathscr{T}}{4}\right), 2\sqrt{\Lambda}\right) + \Phi\left(\mathscr{N}\left(\frac{\mathscr{T}}{2}, \frac{\mathscr{T}}{4}\right), \mathscr{T} - 2\sqrt{\Lambda}\right).$$

Corollary 1. *Let $\mathscr{A}$ be an algorithm which, after a choice of non-zero linear masks $\delta, \gamma \in \mathbb{F}_2^n \backslash \{(0, \ldots, 0)\}$ is fixed, is given access to some permutation $\pi : \mathbb{F}_2^n \to \mathbb{F}_2^n$ from $\mathscr{P}$.*

When $\mathscr{T} < 2\sqrt{\Lambda}$ and $\mathscr{P} = \mathsf{Perm}(n)$, it is impossible for $\mathscr{A}$ to produce in time $\mathscr{T}$ a set $\mathscr{S} \subseteq \mathbb{F}_2^n$ of size Λ s.t. the tuple $(\mathscr{P}, \pi, \mathscr{S}, \mathscr{R}^{\pi}_{\delta,\gamma})$ is α-separable for any $\alpha > 0$.

On the other hand, when $\mathscr{T} \geq 4\sqrt{\Lambda}$ and $\mathscr{P} = \hat{\mathscr{E}}$ (in the notation of Definition 24), then it is impossible for $\mathscr{A}$ to produce in time $\mathscr{T}$ a set $\mathscr{S} \subseteq \mathbb{F}_2^n$ of size Λ s.t. the tuple $(\mathscr{A}, \mathscr{P}, \pi, \mathscr{S}, \mathscr{T}, \mathscr{R}^{\pi}_{\delta,\gamma}, \alpha)$ is a key-less linear distinguisher for any $\alpha > 0$.

Proof. The first result follows directly from Theorem 1 when observing that the both sums are zero when $\mathscr{T} < 2\sqrt{\Lambda}$. The second result follows from Theorem 1 when observing that the sums equal one when $\mathscr{T} = 4\sqrt{\Lambda}$, which makes $(\mathscr{T}, \Lambda, \alpha)$-intractability impossible. □

Note 5. *The key-less linear distinguisher specified in Definition 24 does not ask to provide outputs. Thus, it is not ruled out to give a valid key-less linear distinguisher without pre-computation, i.e. to have $\mathscr{T} = 0$. Indeed, one of the concrete applications we show to the block cipher* PRESENT *does not need any computations.*

From Corollary 1 it follows that when no pre-computation is allowed, i.e. when $\mathscr{T} = 0$, any algorithm $\mathscr{A}$ producing a set $\mathscr{S} \subseteq \mathbb{F}_2^n$ together with any relation $\mathscr{R}_{\delta,\gamma}^{\mathscr{E}_K}$ for a permutation $\mathscr{E}_K \in \hat{\mathscr{E}}$, where $\delta, \gamma \in \mathbb{F}_2^n \backslash \{(0,\dots,0)\}$, yields a key-less linear distinguisher $(\mathscr{A}, \hat{\mathscr{E}}, \mathscr{E}_K, \mathscr{S}, \mathscr{T}, \mathscr{R}_{\delta,\gamma}^{\mathscr{E}_K}, \alpha)$ for some $\alpha > 0$. *Note, however, that the parameter α measures how likely such a distinguisher is to succeed for a specific key. For example, when α is very low, one might have a valid key-less linear distinguisher for many rounds, but for a tiny fraction of the key space. As such, when $\mathscr{T} = 0$, such a key-less linear distinguisher is to be taken with a grain of salt, depending on the value α. In the following discussions, we always provide together with our distinguishers the parameter α, to make clear the lower bound on the fraction of the key space for which it is valid.*

Having analyzed the generic case, we move on to stating in Theorem 2 a necessary condition for when, for a particular fixed $\pi \in \mathscr{P}$ and non-zero linear masks $\delta, \gamma \in \mathbb{F}_2^n \backslash \{(0,\dots,0)\}$, an algorithm $\mathscr{A}$ can construct $\mathscr{S} \subseteq \mathbb{F}_2^n$ of size Λ in time $\mathscr{T}$, s.t. the tuple $(\mathscr{P}, \pi, \mathscr{S}, \mathscr{R}_{\delta,\gamma}^{\pi})$ is a α-separable.

Theorem 2. *Let $\pi \in \mathscr{P}$ and fix non-zero linear masks $\delta, \gamma \in \mathbb{F}_2^n \backslash \{(0,\dots,0)\}$. Let $\mathscr{S} \subseteq \mathbb{F}_2^n$ be a subset of size Λ. Then the tuple $(\mathscr{P}, \pi, \mathscr{S}, \mathscr{R}_{\delta,\gamma}^{\pi})$ can be α-separable for $\alpha > 0$* if and only if *the absolute correction $|\mathsf{Corr}_\pi|$ of $\mathscr{R}_{\delta,\gamma}^{\pi}$ satisfies $|\mathsf{Corr}_\pi| \geq 2/\sqrt{\Lambda}$. Furthermore, the largest α for which α-separability is obtained, is given by*

$$\alpha = \Pr\Big[|\mathsf{Corr}_\pi| \geq 2/\sqrt{\Lambda}\Big]. \tag{2.36}$$

Proof. Let $\mathscr{X} = \big\{X \in \mathscr{S} \mid \mathscr{R}_{\delta,\gamma}^{\pi}(X) = 1\big\}$. Then $\mathscr{X} \sim \mathscr{B}\big(\Lambda, \frac{1}{2} + \frac{\mathsf{Corr}_\pi}{2}\big)$. We have α-separability *if and only if* $\Pr\Big[\big|\mathbb{E}[\mathscr{X}] - \frac{\Lambda}{2}\big| \geq \sqrt{\Lambda}\Big] \geq \alpha$. Thus, we require either $\mathbb{E}[\mathscr{X}] \geq \frac{\Lambda}{2} + \sqrt{\Lambda}$ or $\mathbb{E}[\mathscr{X}] \leq \frac{\Lambda}{2} - \sqrt{\Lambda}$. Since $\mathbb{E}[\mathscr{X}] = \frac{\Lambda}{2} + \Lambda \cdot \mathsf{Corr}_\pi/2$, this happens exactly when $|\mathsf{Corr}_\pi| \geq 2/\sqrt{\Lambda}$. From this, the results follow. □

2.2.3 Specification of PRESENT

PRESENT is a 64-bit iterated block cipher, designed by Bogdanov, Knudsen, Leander, Paar, Poschmann, Robshaw, Seurin, and Vikkelsø [75] for use in lightweight applications such as *radio-frequency identification* (RFID) tags and wireless sensor networks, and has been standardized as ISO:29192-2:2012 [132]. Its use in compression function designs, for example using the Davies-Meyer construction, is e.g. studied and advocated for by Bogdanov, Leander, Paar, Poschmann, Robshaw, and Seurin in [76]. The size of the master key κ is either 80 or 128 bits, and the respective block ciphers are denoted PRESENT-80 and PRESENT-128. Both ciphers are iterated constructions of $T = 31$ rounds. The PRESENT key schedule algorithm takes a master key $K \in \mathbb{F}_2^\kappa$ and produces 32 κ-bit round keys, but only the 64 most significant bits are used in the key addition of each round. We denote the 64 most significant bits used by $K_0, \ldots, K_{31}$. For the details of the key scheduling algorithm, we refer to the specification [75].

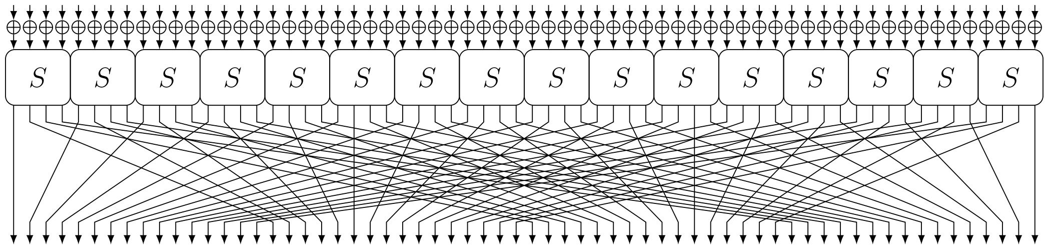


Figure 2.10: A single round of PRESENT

The structure of PRESENT is a key-alternating substitution-permutation network, repeating the round function

$$F_t(K_t, X) = P(S(X \oplus K_t)), \quad 0 \leq t < T, \tag{2.37}$$

where X is the 64-bit state input, S is the parallel application of 16 identical 4-bit S-boxes and P is a fixed bitwise permutation[3]. The full cipher is composed of $T = 31$ applications of the round function followed by addition of a post-whitening key (the last round key K_{31}), so

$$\mathscr{E}_K(M) = F_{30}(K_{30}, \cdot) \circ \cdots \circ F_0(K_0, \cdot)(M) \oplus K_{31}. \tag{2.38}$$

An illustration of a single round of PRESENT is given in Figure 2.10. The PRESENT S-box is specified in Table 2.9 and the bit-permutation P is defined s.t. bit i, $0 \leq i \leq 63$, is moved to bit $P(i)$ where

$$P(i) = 16 \cdot (i \bmod 4) + 4 \cdot \left\lfloor \frac{i}{16} \right\rfloor + \left\lfloor \frac{i \bmod 16}{4} \right\rfloor. \tag{2.39}$$

Table 2.9: The 4-bit PRESENT S-box in hexadecimal notation

X	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
$S(X)$	C	5	6	B	9	0	A	D	3	E	F	8	4	7	1	2

2.2.4 Keys and Linear Hulls in PRESENT

One of the first thorough treatments of linear cryptanalysis on PRESENT is by Ohkuma [255]. This work defines *optimal linear trails*, using solely masks of Hamming weight one. Furthermore, 64 *optimal hulls* using these trails are specified, along with the number of trails in each hull. The absolute correlation for one of Ohkuma's T-round optimal trails s is given by $|\mathrm{Corr}_{\mathscr{E}_K}(s)| = 2^{-2T}$. In our introduction of linear cryptanalysis in Section 1.3.6, we stated with Eq. (1.65) that the linear hull correlation depends on the sign of the trail correlations. To that end, we make the following assumption for our analysis.

[3] S and P are called `sBoxLayer` and `pLayer`, respectively, in the specification [75]

Assumption 2. *Consider a linear hull $\mathsf{LH}_T(\delta,\gamma)$ for a block cipher $\mathscr{E}$. For any fixed key $K \in \mathbb{F}_2^\kappa$, we assume that for any two trails $s, s' \in \mathsf{LH}_T(\delta,\gamma)$, where $s \neq s'$, the signs* sgn(s) *and* sgn(s') *are independent Bernoulli random variables with $p = \frac{1}{2}$.*

We note that this assumption has been experimentally verified for PRESENT, see e.g. the work of Bulygin [87] and Leander [209].

Considering a particular T-round optimal hull $\mathsf{LH}_T(\delta,\gamma)$, let λ_T^+ respectively λ_T^- denote the number of trails s in the hull for which $\operatorname{sgn}(s) = 0$, respectively $\operatorname{sgn}(s) = 1$. We also let $\lambda_T = \sharp\mathsf{LH}_T(\delta,\gamma)$, i.e. $\lambda_T = \lambda_T^+ + \lambda_T^-$. By Assumption 2, for a fixed key $K \in \mathbb{F}_2^\kappa$, we have $\lambda_T^+ \sim \mathscr{B}\big(\lambda_T, \frac{1}{2}\big)$, which for sufficiently large λ_T is well approximated by $\lambda_T^+ \sim \mathscr{N}\big(\frac{\lambda_T}{2}, \frac{\lambda_T}{4}\big)$. Let $Z = \lambda_T^+ - \lambda_T^- = 2\lambda_T^+ - \lambda_T$. Thus, Z is normally distributed with $\mu = 2 \cdot \frac{\lambda_T}{2} - \lambda_T = 0$ and $\sigma^2 = 2^2 \cdot \frac{\lambda_T}{4} = \lambda_T$, so $Z \sim \mathscr{N}(0, \lambda_T)$. When $|Z| \geq N$ for some N, where $0 \leq N \leq \lambda_T$, the absolute linear hull correlation is

$$|\mathsf{Corr}_{\mathscr{E}_K}| \geq N \cdot 2^{-2T}. \tag{2.40}$$

Thus, there is a clear trade-off between the lower bound on $|\mathsf{Corr}_{\mathscr{E}_K}|$ and the probability that a randomly chosen $K \in \mathbb{F}_2^\kappa$ yields such a lower bound. For the λ_T values, we refer to [255] or Table B.1 in Appendix B. For a fixed number of rounds T, using the analysis above, λ_T can be used directly to determine a lower bound on $|\mathsf{Corr}_{\mathscr{E}_K}|$ and the probability that for a random $K \in \mathbb{F}_2^\kappa$ this bound is obtained.

Table 2.10 gives, for various probabilities α and number of rounds T the value β such that $\alpha = \Pr[|\mathsf{Corr}_{\mathscr{E}_K}| \geq \beta]$. Table B.2 in Appendix B gives the same data points for $T \in \{1, \ldots, 31\}$.

Table 2.10: Values $\log_2 \beta$ s.t. $\alpha = \Pr\big[|\mathsf{Corr}_{\mathscr{E}_K}| \geq \beta\big]$ for a T-round variant of PRESENT

	α								
T	0.01	0.05	0.10	0.30	0.50	0.70	0.90	0.95	0.99
7	−9.55	−9.94	−10.20	−10.86	−11.48	−12.29	−13.91	−14.91	−17.23
11	−14.74	−15.14	−15.39	−16.06	−16.68	−17.48	−19.10	−20.10	−22.43
16	−21.27	−21.66	−21.92	−22.58	−23.20	−24.01	−25.63	−26.63	−28.95
24	−31.71	−32.11	−32.36	−33.03	−33.65	−34.46	−36.07	−37.07	−39.40
26	−34.33	−34.72	−34.97	−35.64	−36.26	−37.07	−38.68	−39.69	−42.01
28	−36.94	−37.33	−37.58	−38.25	−38.87	−39.68	−41.30	−42.30	−44.62
31	−40.85	−41.25	−41.50	−42.17	−42.79	−43.60	−45.21	−46.22	−48.54

Example 1. *For $T = 28$, we have $\lambda_{28} = 45170283840$. Thus, with probability $\alpha = 0.30$, a randomly chosen $K \in \mathbb{F}_2^\kappa$ yields that one of Ohkuma's optimal hulls has $|\mathsf{Corr}_{\mathscr{E}_K}| \geq 2^{-38.25}$.*

2.2.5 Application to PRESENT

In this section we give key-less linear distinguishers on PRESENT for varying parameters: the number of rounds T; the pre-computation time $\mathscr{T}$; the size Λ of the set $\mathscr{S}$ produced; and the lower bound α on the fraction of the key space for which they are valid. PRESENT has previously received attention in the context of key-recovery attacks, especially with respect to linear cryptanalysis. For existing results besides [87, 209, 255], we also refer to the work of Cho [95], on which our results build. The attack described is completely independent of the key size used, and hence also of the key schedule algorithm.

Probabilistic Phase

In this section we present key-less linear distinguishers on PRESENT using the model introduced in Section 2.2.2. We refer to approach described here as the *probabilistic phase*, which is combined with a *deterministic phase* below, to extend the distinguishers for three more rounds. The distinguishers we present here do not use any pre-computation, i.e. in the notation of the model, we have $\mathscr{T} = 0$. Corollary 1 implies in this case that when $|\mathsf{Corr}_{\mathscr{E}_K}| > 0$, the tuple produced by any algorithm $\mathscr{A}$ is always $(\mathscr{T}, \Lambda, \alpha)$-intractable for *some* $\alpha > 0$, and hence a valid distinguisher. The results match those of distinguishers used in previous key-recovery attacks, and are as such of limited interest. We hope the discussion below makes it easier to follow (and appreciate) the real use of the model introduced, namely the case with the deterministic phase below, when we do some, albeit very little, pre-computation.

Let $\mathbf{e}_i \in \mathbb{F}_2^n$ denote a binary string with a 1-bit on position i and zeroes elsewhere. In the following, let $\mathscr{R}_{\delta,\gamma}^{\mathscr{E}_K}$ be the linear relation used, where $\delta = \gamma = \mathbf{e}_{21}$, which is one of the optimal linear hulls for PRESENT identified by Ohkuma. Also, let $\mathscr{A}$ be an algorithm constructing $\mathscr{S} \subseteq \mathbb{F}_2^n$ by picking Λ arbitrary $X \in \mathbb{F}_2^n$. In Table 2.11 we give, for various Λ and number of rounds T, lower bounds α on the fraction of the key space, s.t. $(\mathscr{A}, \hat{\mathscr{E}}, \mathscr{E}_K, \mathscr{S}, \mathscr{T} = 0, \mathscr{R}_{\delta,\gamma}^{\mathscr{E}_K}, \alpha)$ are key-less linear distinguishers.

Table 2.11: Lower bounds α on the fraction of the key space $\mathbb{F}_2^\kappa$ susceptible to key-less linear distinguishers using $\mathscr{T} = 0$, and the specified Λ and number of rounds T. A dash indicates that $\alpha < 0.00$.

	Rounds T													
Λ	10	11	12	13	14	15	16	17	18	19	20	21	22	23
2^{40}	0.96	0.89	0.74	0.41	0.04	–	–	–	–	–	–	–	–	–
2^{44}	0.99	0.97	0.93	0.84	0.61	0.21	–	–	–	–	–	–	–	–
2^{46}	0.99	0.99	0.97	0.92	0.80	0.53	0.12	–	–	–	–	–	–	–
2^{52}	1.00	1.00	1.00	0.99	0.97	0.94	0.85	0.63	0.24	–	–	–	–	–
2^{54}	1.00	1.00	1.00	0.99	0.99	0.97	0.92	0.81	0.55	0.14	–	–	–	–
2^{56}	1.00	1.00	1.00	1.00	0.99	0.98	0.96	0.90	0.77	0.46	0.07	–	–	–
2^{62}	1.00	1.00	1.00	1.00	1.00	1.00	1.00	0.99	0.97	0.93	0.82	0.58	0.17	–
2^{63}	1.00	1.00	1.00	1.00	1.00	1.00	1.00	0.99	0.98	0.95	0.87	0.69	0.33	0.02
2^{64}	1.00	1.00	1.00	1.00	1.00	1.00	1.00	0.99	0.99	0.96	0.91	0.78	0.49	0.09

Note, that the α parameter from Table 2.11 gives immediately the probability that such a T-round key-less linear distinguisher without pre-computation for PRESENT is valid in practice, for a fixed chosen- or known key $K \in \mathbb{F}_2^\kappa$. As examples, we see that with $\Lambda = 2^{40}$, the probability of having a valid key-less linear distinguisher for 13-round PRESENT with a fixed key K is *at least* $\alpha = 0.41$. Another example is a key-less linear distinguisher on 22-round PRESENT which is valid for a fraction of at least $\alpha = 0.33$ of the key space, using $\Lambda = 2^{63}$.

Extension by Deterministic Phase

Next, we describe how one can use pre-computation to extend the key-less linear distinguishers from above, to cover three more rounds with no degradation to the valid key space fraction α. In the notation of the model, we now have $\mathcal{T} > 0$, which in turn means that $(\mathcal{T}, \Lambda, \alpha)$-intractability is no longer granted for free by Corollary 1, unless $\mathcal{T}$ is below $2\sqrt{\Lambda}$.

We describe in the following the algorithm $\mathcal{A}$ which will construct the set of inputs $\mathcal{S}$. The algorithm we give will construct $\mathcal{S}$ such that each $X \in \mathcal{S}$ is guaranteed to follow the linear trail $(\mathbf{e}_{21}, \mathbf{e}_{21}, \mathbf{e}_{21}, \mathbf{e}_{21})$ over the first three rounds of the cipher. We remark that this choice of trail is not unique; several others choices are possible, this is but one example. We refer to the approach we describe as the *deterministic phase*.

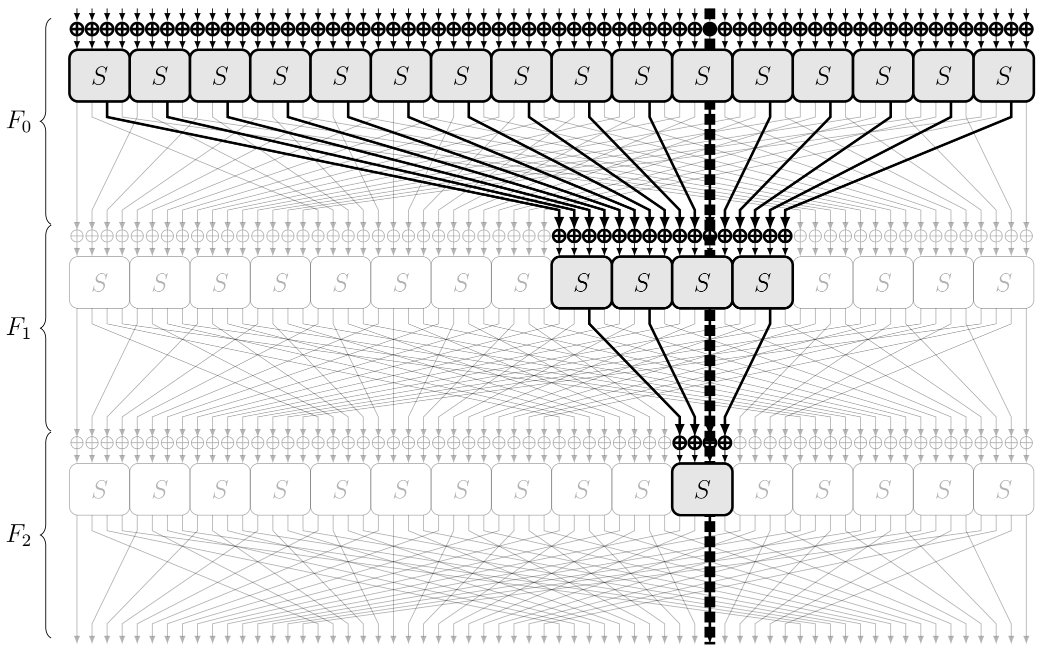


Figure 2.11: Construction of $\mathcal{S}$ for 3-round PRESENT using the trail $(\mathbf{e}_{21}, \mathbf{e}_{21}, \mathbf{e}_{21}, \mathbf{e}_{21})$. The highlighted parts show the S-boxes and key bits involved in the construction. The trail is indicated by the thick dotted line.

For notation, in round $t \in \{0, 1, 2\}$, let $S_{t,j}$ denote the j^{th} S-box of round F_t (counting from right to left) and let $K_{t,j}$ denote the jth least significant bit of the round key K_t, where all indices start from zero. Consider then $S_{2,5}$ which is highlighted in Figure 2.11. By inspection, the PRESENT S-box has 10 inputs X which satisfy $\langle X, (0,0,1,0)\rangle = \langle S(X), (0,0,1,0)\rangle$ and hence follow the trail $(\mathbf{e}_{21}, \mathbf{e}_{21})$ over the round F_2, no matter what the inputs on the other S-boxes are. By adding the key bits $K_{2,23} \| \cdots \| K_{2,20}$ to each X, we can trace those back through the permutation layer of the round F_1. For each value of $X \oplus (K_{2,23}\|\cdots\|K_{2,20})$, we now have a particular value on output bit 1 of each of the S-boxes $S_{1,7},\dots,S_{1,4}$, as indicated in Figure 2.11. By the bijectivity of the S-box, it holds that for each of these S-boxes, half the inputs will give the desired output bit. However, for the S-box $S_{1,5}$ we have the extra requirement that the input bit on position 1 should equal the output bit on position 1, and only 5 inputs satisfy both properties simultaneously. As such, we can trace each of the ten values for X back through round F_1 and also adding the key bits $K_{1,31}\|\cdots\|K_{1,16}$ to obtain $10 \cdot 8^3 \cdot 5 = 25600$ inputs to rounds $F_2 \circ F_1$ which follow the trail $(\mathbf{e}_{21}, \mathbf{e}_{21}, \mathbf{e}_{21})$ by construction. By tracing each of these values back through round F_0 the same way, and adding the full round key K_0, algorithm $\mathscr{A}$ has a construction of the set $\mathscr{S}$ which consists of inputs which follow the trail $(\mathbf{e}_{21}, \mathbf{e}_{21}, \mathbf{e}_{21}, \mathbf{e}_{21})$ over three rounds with probability one. Using this approach to constructing $\mathscr{S}$, the size of the set can be *up to* $\Lambda = 25600 \cdot 8^{15} \cdot 5 = 4503599627370496000 \approx 2^{61.97}$. As such, if one should wish to use a smaller Λ for the key-less linear distinguisher, this is also possible, simply by leaving out elements in the construction of $\mathscr{S}$.

Table 2.12: Tight values α such that $\left(\hat{\mathscr{E}}, \mathscr{E}_K, \mathscr{S}, \mathscr{R}^{\mathscr{E}_K}_{\delta,\gamma}\right)$ is α-separable, where $\mathscr{E}_K$ is T-round PRESENT for a fixed, known $K \in \mathbb{F}_2^\kappa$ (and thus $\mathscr{E}_K \in \hat{\mathscr{E}}$)

Rounds T	18	19	20	21	22	23	24	25	26
α	0.998	0.995	0.988	0.970	0.926	0.819	0.571	0.162	0.001

Consider $\mathscr{E}_K$ being T-round PRESENT for a particular fixed $K \in \mathbb{F}_2^\kappa$, and thus $\mathscr{E}_K \in \hat{\mathscr{E}}$. Let $\mathscr{A}$ be an algorithm for constructing $\mathscr{S}$ using the 3-round deterministic phase described, with $\Lambda \approx 2^{61.97}$ for one of Ohkuma's optimal linear hull relations $\mathscr{R}^{\mathscr{E}_K}_{\delta,\gamma}$. Table 2.12 gives, for various number of rounds T, the highest possible α s.t. $\left(\hat{\mathscr{E}}, \mathscr{E}_K, \mathscr{S}, \mathscr{R}^{\mathscr{E}_K}_{\delta,\gamma}\right)$ is α-separable as per Definition 22. Of course, in order for the key-less linear distinguisher $\left(\mathscr{A}, \hat{\mathscr{E}}, \mathscr{E}_K, \mathscr{S}, \mathscr{T}, \mathscr{R}^{\mathscr{E}_K}_{\delta,\gamma}, \alpha\right)$ to be valid, it also has to hold that the tuple $\left(\hat{\mathscr{E}}, \mathscr{E}_K, \mathscr{S}, \mathscr{R}^{\mathscr{E}_K}_{\delta,\gamma}\right)$ is $(\mathscr{T}, \Lambda, \alpha)$-intractable as per Definition 23, where $\mathscr{T}$ is the time required by $\mathscr{A}$ to construct the set $\mathscr{S}$.

Below, we show that the time $\mathscr{T}$ required to construct $\mathscr{S}$ by $\mathscr{A}$ is equivalent to $\mathscr{T} = \frac{409641}{16T}$ calls to a T-round PRESENT encryption oracle. As such, we have that $\mathscr{T} < 2\sqrt{\Lambda}$, and from Corollary 1, it follows that $\left(\hat{\mathscr{E}}, \mathscr{E}_K, \mathscr{S}, \mathscr{R}^{\mathscr{E}_K}_{\delta,\gamma}\right)$ is $(\mathscr{T}, \Lambda, \alpha)$-intractable. In [207], we give examples of experimental verification of the key-less linear distinguishers presented on 9-round PRESENT. The code for this experimental verification is available as [208].

Time Complexity. In this section we analyze the computational complexity, i.e. the time $\mathscr{T}$ required by $\mathscr{A}$ to construct $\mathscr{S}$ in the deterministic phase of Section 2.2.5. In order to measure the time $\mathscr{T}$ spent in this phase, we determine the number of S-box lookups performed by $\mathscr{A}$ and then compare this to the number of S-box applications for a full call to the encryption oracle.

Let us consider all S-boxes as being different for generality, as the complexity in this case will certainly upper bound the case where they are all equal. In particular, since the key is known, this allows us to consider the key addition as part of the S-boxes. The analysis follows the construction of $\mathscr{S}$ by $\mathscr{A}$ itself, starting from round F_2 and working its way up (referring again to Figure 2.11). To determine the 10 inputs to $S_{2,5}$, $\mathscr{A}$ performs one lookup into this S-box. For each of these 10 values, one bit is traced back to an S-box of round F_1, so this adds $10 \cdot 4$ S-box lookups. Finally, $\mathscr{A}$ has 25600 inputs to round F_1 for which it traces one bit back to each of the 16 S-boxes of round F_0, contributing by $25600 \cdot 16$ S-box lookups.

In total, the number of lookups is $1 + 10 \cdot 4 + 25600 \cdot 16 = 409641$. Now, comparing to the number of S-box lookups involved with a call to a T-round PRESENT oracle, the number of lookups would be $16T$, not counting key scheduling. As such, we find that the time $\mathscr{T}$ spent by $\mathscr{A}$ for constructing $\mathscr{S}$ is $\mathscr{T} = \frac{409641}{16T}$.

Memory Complexity. The memory complexity, though not a formal part of the key-less linear distinguisher model, is at a practical level. The storage of the set $\mathscr{S}$ can be encoded efficiently as follows. We define three sets

$$\begin{aligned} Q &= \{X \mid X_1 = S(X)_1\}, \\ S_0 &= \{X \mid S(X)_1 = 0\}, \quad \text{and} \\ S_1 &= \{X \mid S(X)_1 = 1\}. \end{aligned} \tag{2.41}$$

In a set L we store the 25600 inputs which follow the trail $(\mathbf{e}_{21}, \mathbf{e}_{21})$ over $F_2 \circ F_1$. Let $X = X_{15}\|\cdots\|X_0$ denote one such 16-bit from L. The corresponding set of inputs to F_0 is now determined as the Cartesian product

$$S_{X_{15}} \times \cdots \times S_{X_6} \times Q \cap S_{X_5} \times S_{X_4} \times \cdots \times S_{X_0}. \tag{2.42}$$

The storage of Q, S_0 and S_1 take up 5 bytes, 4 bytes and 4 bytes, respectively. The storage of L takes up 50 KB.

Overview of Selected Distinguishers and Discussion

Here, we consider key-less linear distinguishers applying the deterministic phase combined with the probabilistic phase, using $\Lambda \leq 2^{61.97}$. Let w_2 and w_1 denote the number of inputs to F_2 and F_1 used by $\mathscr{A}$ in the construction of $\mathscr{S}$. Then $w_2 \leq 10$ and w_1 is constrained by w_2 since $w_1 \leq 8^3 \cdot 5w_2$. Further, $\Lambda \leq 8^{15} \cdot 5w_1$ and the time $\mathscr{T}$ required by $\mathscr{A}$ to construct $\mathscr{S}$ is $\mathscr{T} = \frac{1+4w_2+16w_1}{16T}$ for T-round PRESENT. Obviously, for a fixed target size Λ, minimizing w_1 yields the lower time complexity $\mathscr{T}$.

Using these simple observations, we give in Table 2.13 an overview of selected results for key-less linear distinguishers on T-round PRESENT. We give the size Λ of $\mathscr{S} \subseteq \mathbb{F}_2^n$ constructed by $\mathscr{A}$, the time $\mathscr{T}$ required to do so, and the parameter α (implicitly, as we give $\alpha \cdot 2^{128}$) for the distinguisher, i.e. the lower bound on the fraction of the key space for which the distinguisher is valid. As such, the table is representative for PRESENT-128. Entries for PRESENT-80 can be directly determined with the same $\mathscr{T}$ and $\alpha \cdot 2^{80}$. Note, however, that for 27-round PRESENT-80 using $\Lambda = 2^{61.97}$, $\alpha \cdot 2^{80} < 0$, so one can distinguish at most 26 rounds of PRESENT-80.

What is evident from Table 2.13 is, that there is a clear limit to how many rounds can be distinguished using a particular Λ. This shows in the diagonal line through the table. Another observation is that for a fixed Λ, there is a clear drop in the fraction of the key space α for which the distinguisher works between T and $T + 1$ rounds. For example, with $\Lambda = 2^{61}$, we see a drop

Table 2.13: Overview of parameters for key-less linear distinguishers on PRESENT. The entries give, for each Λ and each total number of rounds T, a pair $(\log_2 \mathcal{T}, \log_2(\alpha \cdot 2^{128}))$ s.t. algorithm $\mathcal{A}$ can construct $\mathcal{S}$ in time $\mathcal{T}$ and result in a key-less linear distinguisher for *at least* a fraction α of the key space. Here, we indicate for PRESENT-128 the number of keys supporting the distinguisher. The equivalent number for PRESENT-80 is obtained as $\alpha \cdot 2^{80}$. A dash indicates that $\alpha \cdot 2^{128} < 0$.

	Rounds T						
Λ	14	18	22	25	26	27	28
2^{22}	–	–	–	–	–	–	–
2^{25}	–	–	–	–	–	–	–
2^{28}	$(-3.4, 70.9)$	–	–	–	–	–	–
2^{31}	$(-3.4, 119.2)$	–	–	–	–	–	–
2^{34}	$(-3.4, 126.2)$	–	–	–	–	–	–
2^{37}	$(-3.4, 127.5)$	–	–	–	–	–	–
2^{40}	$(-3.4, 127.8)$	$(-3.8, 107.1)$	–	–	–	–	–
2^{43}	$(-3.4, 127.9)$	$(-3.8, 124.3)$	–	–	–	–	–
2^{46}	$(-3.4, 128.0)$	$(-3.8, 127.1)$	–	–	–	–	–
2^{49}	$(-1.7, 128.0)$	$(-2.1, 127.7)$	$(-2.4, 75.1)$	–	–	–	–
2^{52}	$(0.9, 128.0)$	$(0.5, 127.9)$	$(0.3, 119.8)$	–	–	–	–
2^{55}	$(3.9, 128.0)$	$(3.5, 128.0)$	$(3.2, 126.3)$	–	–	–	–
2^{58}	$(6.9, 128.0)$	$(6.5, 128.0)$	$(6.2, 127.5)$	$(6.0, 103.1)$	–	–	–
2^{61}	$(9.9, 128.0)$	$(9.5, 128.0)$	$(9.2, 127.8)$	$(9.0, 123.7)$	$(9.0, 108.5)$	$(8.9, 21.0)$	–
$2^{61.97}$	$(10.8, 128.0)$	$(10.5, 128.0)$	$(10.2, 127.9)$	$(10.0, 125.4)$	$(9.9, 117.1)$	$(9.9, 71.8)$	–

from $2^{108.5}$ keys supporting the distinguisher for 26 rounds to just 2^{21} keys for 27 rounds. What is also apparent is that in all cases, $\mathcal{T}$ is much less than $2\sqrt{\Lambda}$, indeed sometimes $\mathcal{T} < 1$, so by Corollary 1, $(\mathcal{T}, \Lambda, \alpha)$-intractability is for granted.

One thing worth discussion is the time complexity $\mathcal{T}$. This is the time, converted to equivalent calls to a T-round encryption oracle, required by the key-less linear distinguisher algorithm $\mathcal{A}$ to construct the set $\mathcal{S}$. In a scenario where one would verify the distinguisher for a concrete block cipher $\mathcal{E}_K$, i.e. for a particular value of K, one would need to determine the value of the random variable $\mathcal{X}$ of Definition 22. What we denote as the *verifying complexity* in this case is dominated by Λ, because this is the number of inputs to the permutation that needs to be evaluated in order to determine $\mathcal{X}$.

2.2.6 Discussion and Conclusions

In this section we have formalized the notion of distinguishers for block ciphers using linear cryptanalysis in the key-less setting, i.e. where the block cipher is instantiated with a single known or chosen key. The introduced key-less statistical distinguisher based on linear cryptanalysis led to a wide variety of results on the ISO-standardized block cipher PRESENT. For example, we provide a linear distinguisher of up to 26 and 27 rounds of PRESENT-80 and PRESENT-128, with respective computational complexities of about 2^9 and 2^{10}, and verifying complexities of about 2^{61} and $2^{61.97}$, for both PRESENT variants. The very low computational complexity made a practical verification possible for a reduced number of rounds, but also leaves room for improvements. For example, an open question is whether it is possible to extend the deterministic phase of Section 2.2.5 to cover more rounds, while still keeping the time $\mathscr{T}$ in the order of 2^{30} to allow for a valid distinguisher, c.f. Corollary 1.

While PRESENT was chosen because it is a somewhat high profile cryptanalytic target, and the fact that relatively long useful linear hulls exist, we point out that the new distinguisher model is not specifically tailored for it. For example, KATAN, a cipher design by Cannière, Dunkelman, and Knezevic with a very different round transformation and design philosophy, exhibits linear effects as described in the specification paper [88], which makes it another interesting target for an application of the techniques introduced in this section.

More research is needed on the relations between the use of degrees of freedom and the number of rounds that can be sidestepped, as e.g. in our deterministic phase. Even though there is no good theoretical understanding of this yet, the literature already contains many data points for *differential* properties. The linear counterpart seems different and interesting enough to warrant a separate study, see also Appendix B.

The techniques we developed for the presented distinguisher might also have applications to pre-image attacks that are inspired by linear cryptanalysis, or at least to somewhat speed-up brute-force pre-image search. It will be interesting to see how this approach compares to other such methods, e.g. the biclique approach [77] and the *bruteforce-like cryptanalysis* of Rechberger [264]. Also, the approach presented here naturally and directly applies to permutations which have become an increasingly important primitive in their own right, also due to the popularization of the sponge construction, which we described in Section 1.2.4.

Finally, to round up this section, we highlight a very recent work of Blondeau, Peyrin, and Wang [69], which closely relates to the work described in this section. The authors give distinguishers on full PRESENT (for both key sizes) in the known-key model using differential cryptanalysis. The framework, and differential distinguishers used, are based on the work of Blondeau and Nyberg [68]. As such, the considerable contribution of [69] which allows to reach the full number of rounds, is a new approach to a meet-in-the-middle layer which, in many aspects is very similar to the deterministic phase described for our distinguisher above.

2.3 Forgery and Key Recovery on Selected Authenticated Encryption Schemes

In this section, we present attacks on two separate AE(AD) schemes. The first is AVALANCHE, a first-round proposal for the CAESAR competition, designed by Alomair [20]. The second scheme is a design by Jeddi, Amini, and Bayoumi called RBS [172]. While AVALANCHE supports associated data, RBS does not. The attacks we describe are able to fully recover all the key material, so it is the worst break possible of such a scheme, as it allows to encrypt, decrypt and forge arbitrary data. The attack for AVALANCHE is described in Section 2.3.1, while we describe the attack on

RBS in Section 2.3.2. We remark that AVALANCHE was not among the designs moving two the second round of the CAESAR competition.

Publication

The results presented in this section are from:

[79] Andrey Bogdanov, Christoph Dobraunig, Maria Eichlseder, Martin M. Lauridsen, Florian Mendel, Martin Schläffer, and Elmar Tischhauser. Key Recovery Attacks on Recent Authenticated Ciphers. In Diego F. Aranha and Alfred Menezes, editors, *Progress in Cryptology - LATINCRYPT 2014 - Third International Conference on Cryptology and Information Security in Latin America, Florianópolis, Brazil, September 17-19, 2014, Revised Selected Papers*, volume 8895 of *Lecture Notes in Computer Science*, pages 274–287. Springer, 2014.

Author Contribution

The paper [79] forming the basis for this section is a merged result of analysis on three authenticated encryption schemes: AVALANCHE, CALICO and RBS. Together with Bogdanov and Tischhauser, the author has contributed the analysis of AVALANCHE and RBS, described in Sections 2.3.1 and 2.3.2 respectively. The analysis of CALICO, which we do not cover herein, was contributed by Dobraunig, Eichlseder, Mendel and Schläffer.

2.3.1 Cryptanalysis of AVALANCHE

First, we remind from Section 1.2.4 that we use N, A, M and C to denote nonce, associated data, message and ciphertext for an AE(AD) scheme. As the specification of AVALANCHE leaves quite some room for interpretation, we make the following assumptions on the design relevant to our attacks:

1. The nonce has a length of $\eta \in \{80, 160, 128\}$ bits for key lengths $\kappa = n \in \{128, 256, 192\}$, respectively,
2. The nonce N is randomly generated,
3. The counter c is initialized to $c = 0$,
4. The tag length is $\tau = 128$, and
5. The $(n + 256)$-bit key K consists of three independent parts $K = (K_P, k, p)$, where k and p are 128 bits each.

AVALANCHE uses the AES to process a message M of ℓ blocks and associated data A of arbitrary length, to produce a ciphertext C of $\ell + 1$ blocks and an authentication tag T. All blocks have a length of n bits. It does not support a public message number, rather a nonce N is generated by the encryption algorithm itself. The secret key used in AVALANCHE is a tuple $K = (K_P, k, p)$, where K_P is a n-bit encryption key and k and p keys for authentication of 128 bits each. The input to AVALANCHE is a 3-tuple (K, A, M) of key, associated data and message. The output is a 4-tuple (N, A, C, T) of nonce, associated data, ciphertext, and tag. As such, AVALANCHE does not strictly adhere to our definition on an AEAD scheme of Definition 4. The scheme uses two main algorithms described in the following: PCMAC for message processing and RMAC for processing associated data. The interfaces and outputs of the two algorithms are

$$(N, C, T_P) = \mathsf{PCMAC}(M) \quad \text{and} \quad T_A = \mathsf{RMAC}(A). \tag{2.43}$$

The final tag T is then computed as $T = T_P \oplus T_A$.

PCMAC and RMAC

The encryption with PCMAC is illustrated in Figure 2.12. The padded ℓ-block message is denoted by $M_1, \ldots, M_\ell$ and the ciphertext blocks by $C_0, \ldots, C_\ell$. The number rand is generated at random.

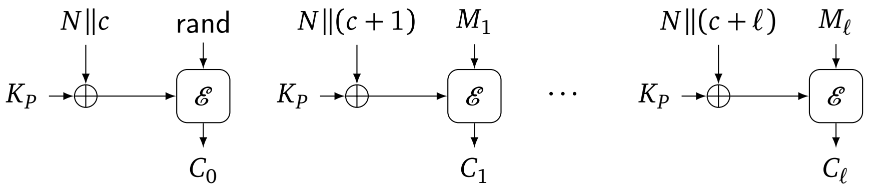


Figure 2.12: Message processing with PCMAC

The output of RMAC is an intermediate tag T_A of 128 bits. RMAC uses the secrets k and p, p being a randomly chosen 128-bit prime and k chosen at random from $\{\lfloor p/2 \rfloor + 1, \ldots, p-1\}$. The intermediate tag T_A is determined as

$$T_A = (1\|A) \cdot k \mod p. \tag{2.44}$$

Recovering the PCMAC key

The critical part of PCMAC is that the encryption key for $\mathcal{E}$ (see Figure 2.12) depends on the nonce and counter. This facilitates key collision attacks, similar to the one by Mendel, Mennink, Rijmen, and Tischhauser on the AEAD scheme McOE-X [227]. Our attack works in an *offline phase* and an *online phase* (see Algorithms 6 and 7). Both are called with the same, arbitrary single-block message M. The offline phase outputs a list L which is used in the online phase. We also note that this technique allows a free trade-off between time and memory by choosing the list size ω accordingly.

```
Algorithm 6: OFFLINE($M$)
  Data: Single-block message $M$
1 $L \leftarrow \emptyset$
2 for $i = 1, \ldots, \omega$ do
3     Choose new $K = (K_P, k, p) \in \mathbb{F}_2^{n+256}$
4     $(N, \varepsilon, C, T) \leftarrow$ AVALANCHE$(K, \varepsilon, M)$
5     $L \leftarrow L \cup \{(C_1, K_P, N)\}$
6 end
7 return $L$
```

```
Algorithm 7: ONLINE($M, L$)
  Data: Single-block message $M$, and list
        $L$ output from Algorithm 6
1 for $i = 1, \ldots, 2^n/\omega$ do
2     Obtain $(N, \varepsilon, C, T)$ for $(M, \varepsilon)$ from
      oracle
3     if $\exists (X, Y, Z) \in L : X = C_1$ then
4         return $Y \oplus ((N \oplus Z) \| 0^{n-\eta})$
5     end
6 end
7 return Failure
```

In the offline phase we build a table of size ω of AVALANCHE encryptions of the same message block, using different keys, mapping the first ciphertext block C_1 to the secret K_P, used in PCMAC, and the nonce N. In the online phase we request the encryption of the same single-block message M in total $2^n/\omega$ times. By the birthday paradox, we expect to see a collision in the oracle output C_1 in the online phase and the list L from the offline phase. As the nonce N is public, we can then recover the secret key K_P by adding it to the stored nonce Z and key Y. We can verify candidate keys using an additional encryption. Obviously, choosing $\omega = 2^{n/2}$ gives the best overall complexity, using just $2 \cdot 2^{n/2}$ time and memory in the order of $2^{n/2}$ to store L.

Recovering the RMAC Secret Parameters

As explained, for RMAC we conservatively assume that the secret parameters k and p are generated as independent 128-bit quantities. In fact, for this attack, these can be of *arbitrary length*, without changing the attack complexity. We explain it in the following.

To recover (k, p), we first use the attack described above to recover the secret K_P. We furthermore ask for encryption and tag of some arbitrary message block: once with empty associated data, i.e. $A = \varepsilon$, and once with $A = 0$, i.e. a single zero bit. Let the corresponding outputs of AVALANCHE be (N, ε, C, T) and $(N', 0, C', T')$, where $T = T_P \oplus T_A$ and $T' = T'_P \oplus T'_A$. With K_P in hand, we can ourselves compute T_P and T'_P using PCMAC. Using the definition of RMAC of Eq. (2.44), we observe that for the case where $A = \varepsilon$ we directly obtain $T_A \equiv k \mod p$, but since $k \in \{\lfloor p/2 \rfloor + 1, \ldots, p-1\}$ we have $k = T_A$. Now, for the case where $A = 0$, we find

$$\begin{aligned} & T'_A \equiv (1\|0) \cdot k \mod p \\ \Leftrightarrow\ & T'_A \equiv 2k \mod p \\ \Leftrightarrow\ & p = 2k - T'_A. \end{aligned} \tag{2.45}$$

We therefore obtain the secret parameters (k, p) of RMAC with a complexity of two one-block encryption queries (on top of the complexity required to obtain K_P, as described above). In summary, we have recovered all $n + 256$ bits of secret key material for AVALANCHE in about $2^{n/2}$ time and memory.

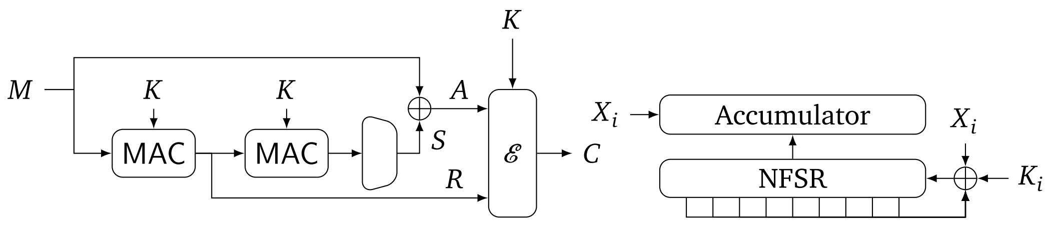
2.3.2 Cryptanalysis of Rbs

Rbs is an authenticated encryption scheme by Jeddi, Amini, and Bayoumi [172] proposed for use in RFID tags. The idea of Rbs is to insert the bits of a MAC, computed on the message, among the message bits in key-dependent positions, to produce the authenticated ciphertext. Note, that Rbs is an AE scheme, and does not support associated data.

Specification

The Rbs scheme is depicted in Figure 2.13. It takes as input a 64-bit message M and a 132-bit key K to produce a 132-bit authenticated ciphertext C. Effectively, the key is split in two parts of sizes which we denote n and m respectively: the least n significant bits are used for clocking the MAC (which we describe in detail later) while the most significant m bits are used for initializing a *non-linear feedback shift register* (NFSR) in the MAC. An NFSR is a component usually employed in stream ciphers. They work by keeping a state of a particular number of bits, and they have the functionality of being *clocked*. Each time the NFSR is clocked, it first computes a *feedback bit* as a non-linear function in the state bits. Then the register is shifted by one position to the left, outputting the most significant bit, and the feedback bit is put in the least significant position. If the feedback function had been linear, we would have a *linear feedback shift register* (LFSR).

Rbs uses $n = 64$ and $m = 68$, but we mostly use n and m for generality in the following. Note that a requirement on the key K is that it has Hamming weight 68, and hence the size of the key space is $\binom{132}{68} \approx 2^{128.06}$. The Rbs MAC takes either a 64-bit or 68-bit input to be processed, along with the key K, and produces a 68-bit output. While Rbs does not specify this, we assume (without influence on our attack) that the second MAC output is truncated by taking the least significant 64 bits to obtain the value denoted S in Figure 2.13a.



(a) Rbs overview **(b)** Rbs MAC(K,X)

Figure 2.13: The Rbs scheme

Consider A and R of Figure 2.13a as registers of 64 bits and 68 bits, respectively. For the function $\mathscr{E}$, the i^{th} ciphertext bit, denoted C_i, is obtained as

$$C_i = \begin{cases} \mathsf{lsb}_1(A) & ,K_i = 0 \\ \mathsf{lsb}_1(R) & ,K_i = 1. \end{cases} \tag{2.46}$$

Each time a bit is taken from either A or R, to produce a ciphertext bit, the corresponding register is right-rotated one position. As 132 bits are produced for the ciphertext, $\mathscr{E}$ effectively obtains C

by inserting the bits of R (the MAC of the message), in order, among the bits of A at key-dependent positions.

The RBS MAC. The MAC used in RBS, which we denote $\mathsf{MAC}(K,X)$, depicted in Figure 2.13b where the input is denoted X, is a Grain-like design based on the MAC of Ågren, Hell, Johansson, and Meier [9]. It is composed of a 68-bit NFSR and a 68-bit accumulator. In this work, we consider the NFSR as having an arbitrary feedback function (and indeed the specification does not provide one). When a MAC is computed, the NFSR is loaded with the most significant 68 bits of the key, i.e. $K_{131}\|\cdots\|K_{64}$ and the accumulator is set to zero. To produce $\mathsf{MAC}(K,X)$, the NFSR is clocked $|X|$ times, i.e. it is shifted left and the least significant bit is set to the feedback XORed with the input bit $X_i \oplus K_i$ where $i = 0,\ldots,|X|-1$. The accumulator is updated *if and only* $X_i = 1$, by XORing the current NFSR state to it (we assume this is done prior to clocking the NFSR). When $|X| > 64$, which is the case for the second MAC call, we assume that one re-uses $K_{63},\ldots,K_0$ for clocking, until all of X is processed, although this makes no difference to our attack.

The Attack

The attack on the RBS scheme we present in the following uses a *single* chosen plaintext and has *expected worst case* time complexity 2^{65} and negligible memory complexity. The attack is based on the following two crucial observations.

Observation 1. *When computing $R = \mathsf{MAC}(K,M)$, if $M = 1$, then it immediately follows from the definition of the MAC that $R = K_{131}\|\cdots\|K_{64}$, i.e. the 68 most significant bits of the key.*

Observation 2. *Assuming one knows $K_{a-1}\|\cdots\|K_0$ for some a with $1 \le a \le 132$, then one can determine the first $\ell = \mathsf{hw}(K_{a-1}\|\cdots\|K_0)$ bits of R, as the bits of R are directly mapped to C by the K_i where $K_i = 1$. These in turn correspond to the first ℓ bits of $K_{131}\|\cdots\|K_{64}$. These can in turn be used to determine more of R, and so on.*

Combined, these observations imply that for $M = 1$, i.e. a single 1-bit, we know that $R = K_{131}\|\cdots\|K_{64}$. When guessing any number of the least significant key bits, a number of bits of R and thus of $K_{131}\|\cdots\|K_{64}$, equal to the Hamming weight of the guess, can be directly obtained from C.

Definition 25 (Free bit iteration). *The i^{th}* free bit iteration*, with $i \ge 0$, refers to the number of bits obtained "for free" in an iteration as described in Observation 2.*

Thus, the 0th free bit iteration refers to the analysis of how many free bits are obtained from the initially guessed key bits; the 1st free bit iteration refers to how many free bits are obtained from the ones obtained from the 0th free bit iteration, and so on. For $i \ge 0$, in the ith free bit iteration, we let ℓ_i denote the *expected* number of free bits obtained and let δ_i denote the *expected* density of 1-bits in the remaining unknown bits, after obtaining the ℓ_i free bits. We then obtain the following result.

Lemma 2. *Let $K_{a-1}\|\cdots\|K_0$ be the initially guessed key bits and let $\ell_0 = \mathsf{hw}(K_{a-1}\|\cdots\|K_0)$. Then*

$$\begin{aligned}\delta_i &= \frac{m-\sum_{j=0}^{i}\ell_j}{n+m-a-\sum_{j=0}^{i-1}\ell_j}, \quad i \geq 0 \quad \text{and}\\ \ell_i &= \ell_{i-1}\delta_{i-1}, \qquad i \geq 1.\end{aligned} \tag{2.47}$$

Proof. In the i^{th} free bit iteration, $a+\sum_{j=0}^{i-1}\ell_j$ bits have already been guessed, so the denominator of δ_i is what remains unknown. The key bits guessed thus far have Hamming weight $\sum_{j=0}^{i}\ell_j$, so the 1-bits density among the last bits is δ_i.

The number of bits expected to obtained for free in iteration $i+1$ is determined by the expected Hamming weight of the free bit portion just obtained in iteration i, which in turn is $\ell_i\delta_i$. □

We now derive a closed formula for the quantity ℓ_i by observing that the ratios ℓ_{i+1}/ℓ_i between consecutive elements of the sequence are actually constant, i.e. independent of i. We formally prove this in the following lemma.

Lemma 3. *Let a and ℓ_0 be such that $m-\ell_0 \neq n+m-a$ and $n+m-a \neq 0$. With the notations of Lemma 2, we have for $i \geq 1$ that*

$$\ell_i = \left(\frac{m-\ell_0}{n+m-a}\right)^i \ell_0. \tag{2.48}$$

Proof. We prove the claim by induction. For $i=1$, Eq. (2.47) yields $\ell_1 = \left(\frac{m-\ell_0}{n+m-a}\right)\ell_0 = \left(\frac{m-\ell_0}{n+m-a}\right)^1\ell_0$. Assuming Eq. (2.48) holds for all $k \leq i$, we have

$$\begin{aligned}\ell_{i+1} &= \frac{m-\sum_{j=0}^{i}\ell_j}{n+m-a-\sum_{j=0}^{i-1}\ell_j}\cdot\ell_i\\ &= \frac{m-\ell_0\sum_{j=0}^{i}\left(\frac{m-\ell_0}{n+m-a}\right)^j}{n+m-a-\ell_0\sum_{j=0}^{i-1}\left(\frac{m-\ell_0}{n+m-a}\right)^j}\cdot\left(\frac{m-\ell_0}{n+m-a}\right)^i\ell_0.\end{aligned} \tag{2.49}$$

For $r \neq 1$, the geometric series $\sum_{i=0}^{N} r^i$ is equal to $(r^{N+1}-1)/(r-1)$. Instantiating this with $r=\frac{a}{b}$ yields

$$\sum_{i=0}^{N}\left(\frac{a}{b}\right)^i = \frac{\left(\frac{a}{b}\right)^N a-b}{a-b} \qquad \text{and} \qquad \sum_{i=0}^{N-1}\left(\frac{a}{b}\right)^i = \frac{\left(\frac{a}{b}\right)^N b-b}{a-b}. \tag{2.50}$$

Since $\left(\frac{m-\ell_0}{n+m-a}\right) \neq 1$, we can apply this to the two sums in Eq. (2.49), yielding

$$\ell_{i+1} = \frac{m-\ell_0\cdot\frac{\left(\frac{m-\ell_0}{n+m-a}\right)^i(m-l_0)-(n+m-a)}{-\ell_0-n+a}}{n+m-a-\ell_0\frac{\left(\frac{m-\ell_0}{n+m-a}\right)^i(n+m-a)-(n+m-a)}{-\ell_0-n+a}}\cdot\left(\frac{m-\ell_0}{n+m-a}\right)^i\ell_0, \tag{2.51}$$

which can be reformulated to

$$= \frac{\frac{m(-\ell_0-n+a)-\ell_0\left(\frac{m-\ell_0}{n+m-a}\right)^i(m-\ell_0)+\ell_0(n+m-a)}{-\ell_0-n+a}}{\frac{(n+m-a)(-\ell_0-n+a)-\ell_0\left(\frac{m-\ell_0}{n+m-a}\right)^i(n+m-a)+\ell_0(n+m-a)}{-\ell_0-n+a}} \cdot \left(\frac{m-\ell_0}{n+m-a}\right)^i \ell_0, \tag{2.52}$$

and collecting common terms gives

$$\begin{aligned}
&= \frac{(m-\ell_0)\left(\left(\frac{m-\ell_0}{n+m-a}\right)^i \ell_0 + n - a\right)}{\ell_0+n-a} \cdot \frac{\ell_0+n-a}{(n+m-a)\left(\left(\frac{m-\ell_0}{n+m-a}\right)^i \ell_0 + n - a\right)} \cdot \left(\frac{m-\ell_0}{n+m-a}\right)^i \ell_0 \\
&= \left(\frac{m-\ell_0}{n+m-a}\right) \cdot \left(\frac{m-\ell_0}{n+m-a}\right)^i \ell_0 \\
&= \left(\frac{m-\ell_0}{n+m-a}\right)^{i+1} \ell_0,
\end{aligned} \tag{2.53}$$

as claimed. □

Note, that the preconditions of the previous lemma do not impose a limitation for the evaluation of the ℓ_i for relevant values of a. For instance, with $a = n$, the closed formula holds for any $1 \le \ell_0 \le m$, and the remaining case $\ell_0 = 0$ is trivial since all remaining unknown bits must be equal to one.

Optimal Number of Initially Guessed Key Bits. The closed formula of Lemma 4 also yields an estimate for the optimal number of key bits a that should be guessed initially, so as to minimize the complexity of a key recovery attack. Specifically, we should choose $a < n + m$ such that

$$\ell_0 \sum_{i=0}^{\infty} \left(\frac{m-\ell_0}{n+m-a}\right)^i \tag{2.54}$$

reaches $n + m - a$, the number of still unknown bits. Since

$$\begin{aligned}
\ell_0 \sum_{i=0}^{\infty} \left(\frac{m-\ell_0}{n+m-a}\right)^i &= \frac{1}{1-\left(\frac{m-\ell_0}{n+m-a}\right)} \ell_0 \\
&= \left(\frac{n+m-a}{\ell_0+n-a}\right) \ell_0,
\end{aligned} \tag{2.55}$$

this means that the optimal choice of a should be such that

$$\begin{aligned}
&\left(\frac{n+m-a}{\ell_0+n-a}\right)\ell_0 = n+m-a \\
&\Leftrightarrow a = n \quad \text{or} \quad a = n+m.
\end{aligned} \tag{2.56}$$

Note, however that this only holds asymptotically, and it is expected that slightly more than n bits will need to be guessed to determine the remaining part of the key. For RBS, this suggests that an initial guess of around $n = 64$ key bits should be sufficient to determine all remaining 68 key bits. In order to determine how many more bits than n we should guess, a more careful analysis of the progression of the ℓ_i's is needed. In the following, we develop a conservative estimate.

Lemma 4. *Let a and ℓ_i be as in Lemma 2. Let $L(a, \ell_0) = (\ell_0, \dots, \ell_t)$ be the series of ℓ_i defined from a and ℓ_0 such that t is the largest integer s.t. $\ell_t \geq 1$. When guessing a initial key bits, the* expected number of extra free bits obtained *is determined as $\sum_{j=0}^{t-1} \ell_j$ and the* expected Hamming weight *of these bits is determined as $\sum_{j=0}^{t} \ell_j$.*

Proof. This follows directly from the definition of ℓ_i and $L(a, \ell_0)$. □

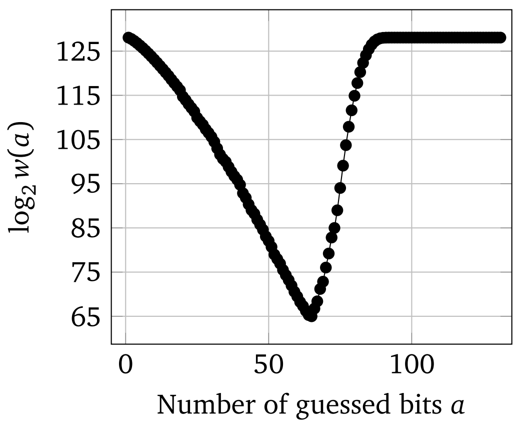
```
Algorithm 8: RBS-KEYRECOVERY
   Data: Number of initial key bits to guess, $a$
 1 $C \leftarrow \textsc{Rbs}(M = 1)$
 2 for $\ell_0 = \max\{0, 64 - a\}, \dots, \min\{68, a\}$ do
 3     forall the guesses of $K'_{a-1} \| \cdots \| K'_0$ of Hamming weight $\ell_0$ do
 4         Let $L = (\ell_0, \dots, \ell_t)$, where $t$ is the largest integer s.t. $\ell_t \geq 1$
 5         $\Xi \leftarrow \max\{0, 132 - a - \sum_{j=0}^{t-1} \ell_j\}$            // # of bits yet unknown
 6         $\Phi \leftarrow \max\{0, 68 - \sum_{j=0}^{t} \ell_j\}$                // # of 1-bits remaining
 7         forall the $\binom{\Xi}{\Phi}$ remaining candidates for $K'_{131} \| \cdots \| K'_{131-\Xi+1}$ do
 8             if $C = \textsc{Rbs}(M = 1)$ under the key $K'_{131} \| \cdots \| K'_0$ then
 9                 return $K'_{131} \| \cdots \| K'_0$ as the correct key $K$
10             end
11         end
12     end
13 end
```

Theorem 3. *Let a, ℓ_i and $L(a, \ell_0)$ be as in Lemma 4. Let $w(a)$ denote the* worst case expected *complexity of key recovery when a is the number of key bits initially guessed. Then*

$$w(a) = \sum_{\ell_0 = \max\{0, a-64\}}^{\min\{68, a\}} \binom{a}{\ell_0} \binom{\max\{0, \lfloor 132 - a - \sum_{j=0}^{t-1} \ell_j \rfloor\}}{\max\{0, \lfloor 68 - \sum_{j=0}^{t} \ell_j \rfloor\}}. \tag{2.57}$$

Proof. When initially guessing $K_{a-1} \| \cdots \| K_0$, the Hamming weight of this guess, ℓ_0, is bounded below by $\max\{0, a - 64\}$, because when $a > 64$, the Hamming weight must be positive by the pigeon-hole principle. The Hamming weight ℓ_0 is bounded above by either a or 68.

There are $\binom{a}{\ell_0}$ ways to distribute the ℓ_0 ones over $K_{a-1} \| \cdots \| K_0$. For each of these, the rightmost binomial coefficient of Eq. (2.57) gives the number of ways to place the remaining 1-bits among the unknown bits for this fixed combination of (a, ℓ_0). We take the sums of the ℓ_j as $\lfloor \sum_j \ell_j \rfloor$ for a conservative estimate of the complexity. Summing over all the possible ℓ_0 for a fixed a, the result follows. □



(a) Plot as a function of a

a	$\log_2 w(a)$
61	68.24
62	67.32
63	66.22
64	65.27
65	65.00
66	66.75
67	68.39
68	71.18
69	72.83
70	76.03

(b) Data points for the best values of a

Figure 2.14: Expected worst time complexity for key recovery in RBS as a function of the number of bits initially guessed, denoted a

Key Recovery. We summarize the resulting key recovery attack on RBS as Algorithm 8. It remains to determine the number of key bits a that should be guessed initially. Figure 2.14 shows the base-2 logarithm of the expected worst case complexity $w(a)$. While Figure 2.14a shows a plot of $w(a)$ with $a \in \{1, \ldots, 131\}$, Figure 2.14b gives a numerical illustration of the best values for a giving the lowest complexity. From the data, we find that guessing $a = 65$ bits gives the lowest key recovery complexity of 2^{65}.

2.3.3 Discussion and Conclusions

We have presented key recovery attacks on two recent authenticated ciphers: AVALANCHE and RBS. The former was a round-one candidate in the ongoing CAESAR competition for authenticated encryption schemes, while the latter is a proposal for use in lightweight applications.

While AVALANCHE makes use of the AES, which is a solid primitive, we stress that the attacks presented here are purely structural, i.e. the weaknesses are present due to *the way the primitives are combined* and not the primitives themselves. For AVALANCHE, the key recovery is possible due to the nonce being used as (part of) the key material, thus facilitating a key collision attack. For RBS, we used a guess-and-determine approach. In all cases, the key was recovered with a complexity of at most square root of the brute-force effort. Our attacks allows an adversary to perform forgeries in both cases, and also to the decrypt arbitrary ciphertexts.

We remark that in [79], we give an attack very similar to the one presented on AVALANCHE, on another round-one CAESAR proposal, CALICO. However, due to the similarity of the two attacks, we opted not to include the latter here.

3 Design of Symmetric Primitives

In the previous chapter, we saw examples of the application of various cryptanalytic attack vectors to block ciphers and authenticated encryption schemes, in a range of different applications and with varying adversarial models and goals. The two tasks of cryptanalyzing a particular symmetric primitive on one hand, and that of designing a new excellent primitive on the other hand, are inextricably linked: to be able to motivate the design choices made, one must understand the possible security violations that can occur, under various assumptions. One could argue that designing a new good symmetric primitive is harder than trying to break one. After all, to provide a cryptanalytic result, one needs to find a single property that can be lifted to an attack, while the designer needs to consider *all* possible attacks including, to some extent, the capabilities of existing attacks in the future.

In this chapter, we turn towards design aspects of symmetric primitives. We cover two results, which go in somewhat different directions. First, in Section 3.1, we consider generalizations of the widely acclaimed AES block cipher, and how a particular operation used in such ciphers can be optimized. Second, we introduce our round-one proposal for the ongoing CAESAR competition for authenticated encryption schemes in Section 3.2. Our proposal is called PRØST, and is based on a newly designed and highly secure permutation.

3.1 Permutations and Rotations in AES-like Ciphers

In this section, we consider ciphers which have a structure inspired by the AES, the widely used block cipher described in Section 1.2.1. We develop a structured approach to analyzing the permutation layer, i.e. the generalized ShiftRows-like operation, for AES-like ciphers with respect to diffusion and resistance towards differential- and linear cryptanalysis. For this, we start by defining a general framework for AES-like ciphers. Note that we do not restrict to the case where permutation is identical in all rounds; rather we allow for different choices of the permutation in different rounds. Moreover, we first consider arbitrary word-wise permutations and later restrict ourselves to word-wise rotations of the rows. The latter have the appeal of being efficiently implementable on many modern CPUs.

Publication

The results presented in this section are from:

[42] Christof Beierle, Philipp Jovanovic, Martin M. Lauridsen, Gregor Leander, and Christian Rechberger. Analyzing Permutations for AES-like Ciphers: Understanding ShiftRows. In Kaisa Nyberg, editor, *Topics in Cryptology - CT-RSA 2015, The Cryptographer's Track at the RSA Conference 2015, San Francisco, CA, USA, April 20-24, 2015. Proceedings*, volume 9048 of *Lecture Notes in Computer Science*, pages 37–58. Springer, 2015.

Author Contribution

The contributions of the author presented in this section are: definition of an AES-like cipher; contributing to all reduction results and their proofs, as presented in Section 3.1.4; the mixed-integer linear programming model to solve for bounds on active S-boxes; contributing to the search algorithm for optimal parameters, as described in Section 3.1.5; and obtaining parts of the experimental results presented in Section 3.1.5 and summarized in Appendix C.

3.1.1 Introduction and Motivation

Since 2000 with the standardization of Rijndael as the AES [252], an astonishing number of new primitives using components similar to the AES have seen the light of day. Examples of such include, but are not limited to: block ciphers 3D by Nakahara Jr. [178], Anubis by Barreto and Rijmen [38], LED by Guo, Peyrin, Poschmann, and Robshaw [152], mCrypton by Lim and Korkishko [211] and PRINCE by Borghoff, Canteaut, Güneysu, Kavun, Knezevic, Knudsen, Leander, Nikov, Paar, Rechberger, Rombouts, Thomsen, and Yalçın [83]; hash functions like Whirlpool by Barreto and Rijmen [39], ECHO by Benadjila, Billet, Gilbert, Macario-Rat, Peyrin, Robshaw, and Seurin [48], Grøstl by Gauravaram, Knudsen, Matusiewicz, Mendel, Rechberger, Schläffer, and Thomsen [139], LANE by Indesteege, Andreeva, Cannière, Dunkelman, Käsper, Nikova, Preneel, and Tischhauser [161], PHOTON by Guo, Peyrin, and Poschmann [151], Twister by Fleischmann, Forler, Gorski, and Lucks [126]; as well as components of CAESAR candidates PAEQ by Biryukov and Khovratovich [59], PRIMATEs by Andreeva, Bilgin, Bogdanov, Luykx, Mendel, Mennink, Mouha, Wang, and Yasuda [23], PRØST by Kavun, Lauridsen, Leander, Rechberger, Schwabe, and Yalçın [181], and STRIBOB by Saarinen [277]. This can largely be attributed to the seminal wide-trail design strategy [105] which was introduced along with Rijndael and its predecessor SQUARE by Daemen, Knudsen, and Rijmen [107] for the first time.

The wide-trail strategy is an elegant way of ensuring good diffusion properties and at the same time allow designers to easily give bounds on the resistance towards differential- and linear cryptanalysis. Additionally, another advantage is that it decouples the choice of the non-linear layer and the linear layer to a large extent: in a nutshell, any good S-box combined with any good linear layer will result in a cipher resistant against linear- and differential attacks.

For AES-like ciphers, including all the above mentioned designs, the linear layer itself is composed of two parts: one resembles the AES MixColumns operation and the other resembles the AES ShiftRows operation. The MixColumns-like operation is a matrix multiplication of the columns of the state and the ShiftRows-like operation is a permutation of the words of the state. For the former, the criteria are well understood. All that is required here is that this operation has a suitably high *branch number*. While we define the concept formally below, in short the branch number corresponds to the minimal sum of the number of active S-boxes in an input/output column, provided an active input column, and the number of active S-boxes is the essential tool for bounding the success probability of linear- and differential attacks. In stark contrast, for the operation resembling ShiftRows, the situation is significantly less clear. Basically, the ShiftRows-like operation highly influences the number of active S-boxes when considering more than two rounds only. Understanding the bounds for more than two rounds is crucial for many good designs. With a well-chosen ShiftRows-like operation it is usually possible to derive much stronger bounds for more rounds than the trivial bound one gets by multiplying the two-round bound by half the number of rounds.

In the case of the AES (and others including [48, 59, 83]) one uses a so-called *superbox* argument to prove strong bounds on four rounds of the cipher. For others, the problem is modeled as a mixed-integer linear programs like in [23, 181, 237] which allows the computation of bounds for an (in principle) arbitrary number of rounds *for a given choice of the* ShiftRows-like operation. However, no structured approach for analyzing the influence of the ShiftRows-like operation on the security of the cipher has been undertaken previously. The results so far remain ad hoc and specific to a given choice of parameters. Considering the large number of designs following this approach, this shortcoming is quite surprising and unsatisfactory from a scientific perspective. In particular, the choices made are often not optimal and not based on an adequate understanding of the implications.

First, and as a core contribution to a structured approach, we simplify the problem of determining good permutations for AES-like ciphers by introducing the notion of equivalent permutation parameters. It is intuitively clear that many choices of the permutation will lead to the same behavior of the cipher. One such example is changing the order of the rotation constants for the ShiftRows operation in the AES, i.e. rotate the first row by 3, the second by 2, and so on. We will make this intuition precise and, as will be shown below, discover more involved examples of the above. The notion of equivalence will imply the same lower bound on the number of guaranteed active S-boxes. This is interesting theoretically, as it allows to simplify the problem. For example, we prove that a general permutation can never yield better results than a permutation that operates on the rows individually. Furthermore, using this notion of equivalence, we derive a normalized representation of any word-wise rotation of the rows. This allows to significantly reduce the problem domain and thus the search space for a computational approach.

In the second part of our analysis, we use this normalized representation in a combination with solving mixed-integer linear programs using the IBM ILOG CPLEX library [100]. The source code for this part is available as [43]. This results in optimal parameter suggestions for a wide range of AES-like ciphers. In particular, it allows us to suggest improved parameters for Rijndael-192, Rijndael-256, PRIMATEs-80 and PRØST-128 on this front. Finally, given our extensive experimental results, we conjecture an optimal lower bound on the number of active

S-boxes possible for specific cases of the state geometry. Those parameters are such that they allow for an iterative version of the superbox argument mentioned above. We also provide a permutation which guarantees this conjectured optimal bound. In contrast to prior work, e.g. that of ECHO and PAEQ, this permutation layer is generic and, more importantly, realized with *cyclic row rotations only*, thus allowing for an easy and efficient implementation.

3.1.2 The AES and AES-like Ciphers

When introducing the concept of block ciphers in Section 1.2.1, we mentioned the AES and how it is a very frequently used block cipher today. In this section, we go into the details of the AES, and describe exactly how, and why, it works like it does. As mentioned previously, the AES is a family of three block ciphers, AES-128, AES-192 and AES-256. All have a block size of $n = 128$ bits, and the key sizes are $\kappa \in \{128, 192, 256\}$, according to the naming of the cipher. The AES ciphers are a subset of the Rijndael family, the entrant to the AES competition by Daemen and Rijmen [104]. In the following, we first specify the AES algorithm, discuss some of the choices made by the designers, and finally we introduce our concept of *AES-like ciphers*, a generalization of the AES, which we will be analyzing in the remainder of this section.

The AES

The AES operates on blocks of $n = 128$ bits. Let $X \in \mathbb{F}_2^n$ denote an AES state. The state is split into 16 *words* of $m = 8$ bits (i.e. one byte) each, denoted $X_0, \ldots, X_{15}$. The words are arranged in a 4×4 state, as shown in Figure 3.1. Table 3.1 gives an overview of the parameters for the three AES ciphers.

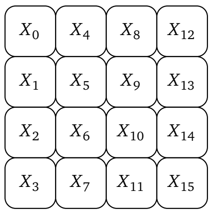


Figure 3.1: The AES state

Table 3.1: Overview of AES parameters

Cipher	Block size n	Key length κ	Rounds T
AES-128	128	128	10
AES-192	128	192	12
AES-256	128	256	14

The AES is a key-alternating block cipher, using round keys $K_0, \ldots, K_T$ derived using a key schedule algorithm from the κ-bit master key K. A round function $F_t : \mathbb{F}_2^n \times \mathbb{F}_2^n \to \mathbb{F}_2^n$ is applied a total of T times. The first $T-1$ rounds are defined by

$$F_t(K_{t+1},X) = \mathsf{AddRoundKey}(K_{t+1}) \circ \mathsf{MixColumns} \circ \mathsf{ShiftRows} \circ \mathsf{SubBytes}(X), \quad 0 \le t < T-1, \tag{3.1}$$

while the last round F_{T-1} is different, as it omits the MixColumns operation:

$$F_{T-1}(K_T,X) = \mathsf{AddRoundKey}(K_T) \circ \mathsf{ShiftRows} \circ \mathsf{SubBytes}(X). \tag{3.2}$$

The application of the round functions are preceded by adding the round key K_0 as a pre-whitening key. As such, the AES is specified by

$$\mathscr{E}_K(X) = F_{T-1}(K_T) \circ \cdots \circ F_0(K_1)(X \oplus K_0). \tag{3.3}$$

Each of the operations SubBytes, ShiftRows, MixColumns and AddRoundKey used in the round functions F_t, are described in the following. For more details on the design criteria described in the following, we refer to [106].

SubBytes. The SubBytes operation is the non-linear part of the cipher. The AES uses an 8-bit S-box to achieve this non-linearity. By the *Rijndael finite field*, we mean the finite field defined by $\mathbb{F}_{2^8} = \mathbb{F}_2[x]/(x^8 + x^4 + x^3 + x + 1)$. We consider words of the state as being elements in this field. Letting $X \in \mathbb{F}_{2^8}$, the S-box used in the AES is specified by an affine transformation of the inverse $Y = X^{-1}$ of X in the Rijndael finite field,

$$S(X) = \begin{pmatrix} 1&1&1&1&1&0&0&0 \\ 0&1&1&1&1&1&0&0 \\ 0&0&1&1&1&1&1&0 \\ 0&0&0&1&1&1&1&1 \\ 1&0&0&0&1&1&1&1 \\ 1&1&0&0&0&1&1&1 \\ 1&1&1&0&0&0&1&1 \\ 1&1&1&1&0&0&0&1 \end{pmatrix} \cdot \begin{pmatrix} Y_7 \\ Y_6 \\ Y_5 \\ Y_4 \\ Y_3 \\ Y_2 \\ Y_1 \\ Y_0 \end{pmatrix} + \begin{pmatrix} 0 \\ 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \end{pmatrix}, \tag{3.4}$$

where for $X = 0$ one defines $X^{-1} = 0$.

The S-box used in the AES was chosen according to two criteria. First, it should be highly non-linear, and in particular the linear correlation and difference propagation should be as low as possible. The S-box chosen has the property that the highest possible differential probability is 2^{-6}, while the highest possible linear correlation amplitude is 2^{-3}; both are optimal for 8-bit S-boxes (see e.g. Nyberg [246]). Second, the algebraic expression for the S-box should be complicated, when expressed in $\mathbb{F}_{2^8}$. This is achieved by employing the affine transformation on the inverse. Furthermore, the S-box has no fixed points nor any opposite fixed points, i.e. $\nexists X \in \mathbb{F}_{2^8} : X \oplus S(X) \in \{00, \mathrm{FF}\}$.

ShiftRows. In order to obtain diffusion in the cipher, the AES uses the ShiftRows operation in conjunction with the MixColumns operation (see below). The ShiftRows does what the name suggests: it shifts (cyclically) the rows of the 4×4 state. The shift direction is right to left[1], and the row shift amount are given by the vector $(0, 1, 2, 3)$. As such, the ShiftRows operation maps a state X as shown in Figure 3.2.

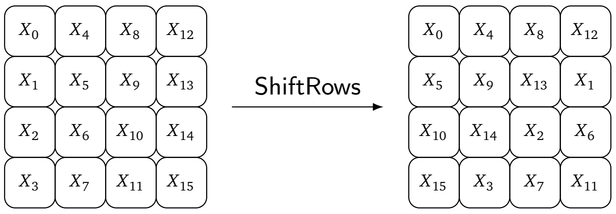

x_0 x_4 x_8 x_{12}
x_1 x_5 x_9 x_{13}
x_2 x_6 x_{10} x_{14}
x_3 x_7 x_{11} x_{15}
ShiftRows
x_0 x_4 x_8 x_{12}
x_5 x_9 x_{13} x_1
x_{10} x_{14} x_2 x_6
x_{15} x_3 x_7 x_{11}

Figure 3.2: The AES ShiftRows operation

The design criteria for the chosen shift amounts are that (i) they should all be different (this is to achieve optimal diffusion, as we shall also see later in this section), and (ii) they should give good resistance to truncated differential attacks and *saturation attacks*. We do not consider those two last points any further in this section. We remark that Rijndael with 7 or 8 state columns, which is not a part of the AES, employ other shift amounts, in particular $(0, 1, 2, 4)$ and $(0, 1, 3, 4)$, respectively.

MixColumns. As mentioned, diffusion in the AES is obtained via a combination of ShiftRows and MixColumns. The MixColumns operation is a linear mixing, working on each column independently. Thus, where the ShiftRows operation moves words to different columns, MixColumns mixes the words inside each column in a linear way. Specifically, the MixColumns operation works by multiplying each column of the state from the left by a 4×4 matrix defined over the Rijndael finite field. The MixColumns matrix M, with entries given in hexadecimal notation for bytes, is defined by

$$M = \begin{pmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{pmatrix}. \tag{3.5}$$

Thus, in the MixColumns operation for $j = 0, \ldots, 3$, we update column consisting of words X_j, X_{j+4}, X_{j+8} and X_{j+12}, as

$$\begin{pmatrix} X_j \\ X_{j+4} \\ X_{j+8} \\ X_{j+12} \end{pmatrix} \mapsto M \cdot \begin{pmatrix} X_j \\ X_{j+4} \\ X_{j+8} \\ X_{j+12} \end{pmatrix}. \tag{3.6}$$

The MixColumns operation fits with the choice of a state of 4 words per column, as to obtain good performance on 32-bit architectures. However, high performance on 8-bit architectures is also a design criteria for the operation. For the cryptographic properties, the transformation was chosen as to be linear over $\mathbb{F}_2$, and to have good diffusion properties. Furthermore, the *branch number* is five (we define this notion formally in Definition 30), which is the highest

[1]We remark that in our analysis below, we consider a left-to-right shift direction

obtainable for such a transformation. As we shall see later in this section, this fact can be used together with the wide-trail design strategy, to obtain good bounds on the resistance towards differential- and linear cryptanalysis. Note that the low coefficients in M means that MixColumns is very efficiently implementable: multiplication by 01 has no cost; multiplication by 02 is a single left shift followed by a conditional XOR; and multiplication by 03 can be implemented as multiplication by 02 plus another XOR by the operand. While the inverse of the MixColumns operation is efficient, it is not *as efficient*, because the coefficients of the corresponding matrix are not as low as those of M.

AddRoundKey. The AddRoundKey operation in AES simply XORs the provided n-bit round key to the n-bit state. As such, in the specification of the F_t of Eq. (3.1) and Eq. (3.2) above,

$$\mathsf{AddRoundKey}(K)(X) = X \oplus K. \tag{3.7}$$

AES-like Ciphers

As already stated, this section looks into a particular design criteria of what we call *AES-like ciphers*. These are ciphers which generalize the AES, and even Rijndael (which is a superset of the AES), to allow arbitrary state sizes and more freedom in the operations used. We define them formally next.

In the following, we refer to binary strings of m bits as *words*. As an example above, we saw that the AES uses words of $m = 8$ bits. We consider words as elements of the finite field $\mathbb{F}_{2^m}$. We refer to $M \times N$ matrices with word entries as *states*. For a state X we use X_i to denote the i^{th} row of X, and $X_{i,j}$ denotes word in the jth column of X_i. As such, and in contrast to the AES above, we use two indices for state words, rather than one. For an $M \times N$ difference X, we use the symbol with a tilde on top, e.g. $\tilde{X}$, to denote the *activity pattern* of X, an $M \times N$ matrix over $\mathbb{F}_2$ where $\tilde{X}_{i,j} = 1$ if $X_{i,j} \neq 0$ and $\tilde{X}_{i,j} = 0$ otherwise. We describe formally our notion of AES-like ciphers in Definition 26.

Definition 26. *An* AES-like cipher *is a block cipher $\mathscr{E}_K$ which is parametrized by a fixed key $K \in \mathbb{F}_2^{\kappa}$, the state dimension $M \times N$, the word size m, the number of rounds T, and a* permutation parameter *$\pi = (\pi_0, \ldots, \pi_{T-1})$, where each π_t is a permutation on $\mathbb{Z}_M \times \mathbb{Z}_N$ (i.e. it permutes the words of a state). The AES-like cipher is composed of round functions F_t, s.t. $\mathscr{E}_K = F_{T-1} \circ \cdots \circ F_0$. Each round function is given by*

$$F_t = \mathsf{AddRoundKey}_t \circ \mathsf{Permute}_{\pi_t} \circ \mathsf{MixColumns}_t \circ \mathsf{SubBytes}, \quad 0 \leq t < T. \tag{3.8}$$

The four bijective transformations on the state used in the round function are defined in the following.

1. SubBytes *substitutes each word of the state according to one or several S-boxes $S : \mathbb{F}_{2^m} \to \mathbb{F}_{2^m}$.*
2. MixColumns$_t$ *applies, in round t, for all columns $j \in \mathbb{Z}_N$ left-multiplication by an $M \times M$ matrix $\mathbf{M}_j^t$ over $\mathbb{F}_{2^m}$:*

$$\begin{aligned} \mathsf{MixColumns}_t &: (\mathbb{F}_{2^m})^{M \times N} \to (\mathbb{F}_{2^m})^{M \times N} \\ \forall j \in \mathbb{Z}_N &: \begin{pmatrix} X_{0,j} & \cdots & X_{M-1,j} \end{pmatrix}^T \mapsto \mathbf{M}_j^t \cdot \begin{pmatrix} X_{0,j} & \cdots & X_{M-1,j} \end{pmatrix}^T, \end{aligned} \tag{3.9}$$

where multiplication in $\mathbb{F}_{2^m}$ is defined by an arbitrary irreducible polynomial from $\mathbb{F}_2[x]$ of degree m.

3. $\mathsf{Permute}_{\pi_t}$ *permutes, in round t, the words within the state due to a given permutation π_t. We use the notation that for a position $(i,j) \in \mathbb{Z}_M \times \mathbb{Z}_N$ in the state, $\pi_t(i,j)$ gives the new position of that word under the permutation π_t:*

$$\begin{gathered}\mathsf{Permute}_{\pi_t} : (\mathbb{F}_{2^m})^{M\times N} \to (\mathbb{F}_{2^m})^{M\times N} \\ \forall i \in \mathbb{Z}_M, \forall j \in \mathbb{Z}_N : X_{i,j} \mapsto X_{\pi_t(i,j)}.\end{gathered} \tag{3.10}$$

4. $\mathsf{AddRoundKey}_t$ *performs word-wise XOR to the state using the t^{th} round key.*

Subsequently, we omit the $\mathsf{AddRoundKey}_t$ operation of Definition 26 from consideration, as it does not affect diffusion properties nor resistance towards differential- and linear cryptanalysis of the AES-like cipher. Note also, that for generality we consider in Definition 26 an arbitrary word permutation $\mathsf{Permute}_{\pi_t}$ while later we will, for efficiency reasons, restrict ourselves to row-wise rotations of the words, as in the ShiftRows operations of the AES.

3.1.3 Bounding Differential- and Linear Hull Probabilities

In the analysis we present, we are concerned with two security aspects of an AES-like cipher, namely diffusion on the one hand and resistance against differential- and linear attacks on the other hand. We formally define our notations for both criteria in the following.

Diffusion

As we described in Section 1.2, the concept of diffusion is about complicating the relationship between the ciphertext bits and plaintext bits. When designing a cipher, it is desirable to obtain what we call *full diffusion* after as few rounds as possible. Indeed, the number of rounds chosen for a cipher is often determined partly by this number.

Definition 27 (Diffusion degree)**.** *For a function $F : \mathbb{F}_2^k \to \mathbb{F}_2^n$, we define the* diffusion degree $d(F)$ *for F as the fraction of bits in the image under F that depend on* each bit *of the pre-image, i.e.*

$$d(F) = \frac{1}{n} \cdot \sharp\{j \in \mathbb{Z}_n \mid \forall i \in \mathbb{Z}_k : \exists X \in \mathbb{F}_2^k : F(X \oplus \mathbf{e}_i)_j \neq F(X)_j\}, \tag{3.11}$$

where $F(X)_j$ denotes the j^{th} bit of $F(X)$. We say that F obtains full diffusion *when $d(F) = 1$.*

Definition 28 (Diffusion-optimality)**.** *Fix the state dimensions $M \times N$. Consider a permutation sequence π for an AES-like cipher which obtains full diffusion after t rounds. We say that π is* diffusion-optimal *if there exists no $\pi' \neq \pi$ which obtains full diffusion after $t' < t$ rounds.*

Differential/Linear Cryptanalysis for AES-like Ciphers

As we saw in Section 1.3, there are some similarities between differential- and linear cryptanalysis. In particular, they both are based on finding characteristics or trails of good probability. In the following, we use always the word *trail* to cover both, and indeed our analysis is equally applicable to analyze both the resistance towards differential- and linear cryptanalysis. We focus, however, on the differential case.

The probability (respectively correlation) of the trails can be upper bounded by lower bounding the number of active S-boxes in any trail. Here, an S-box is active in a given trail if it has a non-zero input difference (respectively mask). In general, if p is the largest probability (respectively correlation) for the S-box to satisfy a differential- or linear property, and any trail has at least k active S-boxes, then the trail property holds with probability (respectively correlation) at most p^k. This approach to ensuring resistance against linear and differential attacks is the basis of the wide-trail design strategy as introduced by the Rijndael designers Daemen and Rijmen in [105]. The second important merit of the wide-trail strategy is, that it allows to treat the S-box and the linear layer of the cipher as black boxes, as long as they fulfill certain conditions. In our analysis of AES-like ciphers, we follow both aspects of this philosophy. The designer is interested in having the lightest trail being as heavy as possible, so as to set the upper bound on the trail probability (respectively correlation) as low as possible. Indeed, knowing this probability is essential when determining the number of rounds for the cipher in the design phase. We give definitions of trails and trail weights in the following.

Definition 29 (Trail and trail weight). *For an AES-like cipher $\mathscr{E}_K$ using m-bit words and state dimension $M \times N$, a T-round* trail *is a $(T+1)$-tuple $(X^0, \dots, X^T) \in \left((\mathbb{F}_{2^m})^{M\times N}\right)^{T+1}$ and the* weight *of the trail is defined as*

$$\sum_{t\in\mathbb{Z}_T} \sum_{i\in\mathbb{Z}_M} \sum_{j\in\mathbb{Z}_N} \tilde{X}^t_{i,j}. \tag{3.12}$$

A pair of inputs $X, X' \in (\mathbb{F}_{2^m})^{M\times N}$ are said to follow the (differential) trail $(X^0, \dots, X^T)$ *over T rounds* if and only if $X^0 = X \oplus X'$ *and for all $t = 1, \dots, T$ it holds that*

$$X^t = (F_{t-1} \circ \cdots \circ F_0)(X) \oplus (F_{t-1} \circ \cdots \circ F_0)(X'). \tag{3.13}$$

Note from Definition 29 that the weight of a trail corresponds exactly to the number of active S-boxes over those T rounds.

Definition 30 (Branch number). *For a linear automorphism $\theta : (\mathbb{F}_{2^m})^M \to (\mathbb{F}_{2^m})^M$, the* differential branch number B_θ *is defined as*

$$B_\theta = \min_{\substack{X,X'\in(\mathbb{F}_{2^m})^M \\ X\neq X'}} \left\{ \sum_{i\in\mathbb{Z}_M} \tilde{\alpha}_i + \tilde{\beta}_i \right\}, \qquad \alpha = X \oplus X', \beta = \theta(X) \oplus \theta(X'). \tag{3.14}$$

In the context of an AES-like cipher $\mathscr{E}_K$, we say $\mathscr{E}_K$ has branch number B if and only if

$$B = \min\left\{B_{\mathbf{M}^t_j\cdot} \mid j \in \mathbb{Z}_N, t \in \mathbb{Z}_T\right\}, \tag{3.15}$$

where $\mathbf{M}^t_j\cdot$ denotes the automorphism defined by left multiplication by $\mathbf{M}^t_j$.

In order to calculate a useful lower bound on the number of active S-boxes in an efficient way, our focus of this work is on the $\mathsf{Permute}_{\pi_t}$ part of the round function. The SubBytes operation will be considered as using an arbitrary S-box $S : \mathbb{F}_{2^m} \to \mathbb{F}_{2^m}$, and the analysis will be independent of the specific instance of S. Each of the $\mathbf{M}_j^t$ matrices used in the MixColumns operation will be considered as black-box linear operations, under the requirement that the AES-like cipher has branch number B. A formal definition of that idea is given in the following. For a T-round permutation parameter $\pi = (\pi_0, \dots, \pi_{T-1})$, let $\widetilde{\mathsf{AES}}_{M,N}(\pi, B)$ denote the set of all $M \times N$ AES-like ciphers over T rounds with branch number B using $\pi_0, \dots, \pi_{T-2}$ in the first $T-1$ rounds. The reason for not including π_{T-1} is that our proofs in the following use the fact that for different permutation sequences we can re-model one AES-like cipher into another, up to the last round, and up to changing MixColumns operations (but maintaining the branch number).

Definition 31. *We say that the sequence of permutations $\pi = (\pi_0, \dots, \pi_{T-1})$* tightly guarantees *$k$* active S-boxes for branch number *B* if and only if *there is a valid trail of weight k for some $\mathscr{E}_K \in \widetilde{\mathsf{AES}}_{M,N}(\pi, B)$ and there is no valid trail of weight $k' < k$, with $k' > 0$, for some $\mathscr{E}'_K \in \widetilde{\mathsf{AES}}_{M,N}(\pi, B)$. We denote this property by $\pi \xrightarrow{B} k$.*

Definition 32 (Trail-optimality)**.** *A sequence of permutations $\pi = (\pi_0, \dots, \pi_{T-1})$ with $\pi \xrightarrow{B} k$ is said to be* trail-optimal *if there exists no $\pi' = (\pi'_0, \dots, \pi'_{T-1})$ s.t. $\pi' \xrightarrow{B} k'$ where $k' > k$.*

Appendix C provides a proof that the number of tightly guaranteed active S-boxes is really independent of the specific S-box instantiations. From Definition 31, it follows that the number of guaranteed active S-boxes is always a lower bound for the actual minimum number of active S-boxes in any concrete instantiation of an AES-like cipher.

3.1.4 Equivalent Permutations

In the following, we present a range of results which simplifies the problem of identifying good permutation parameters π for AES-like ciphers by showing when different permutation parameters are equivalent with respect to resistance towards differential- and linear attacks. Obviously, for a fixed branch number, many different π will tightly guarantee the same number of active S-boxes. Thus, identifying conditions under which two different permutation sequences $\pi \neq \pi'$ tightly guarantee the same bound is significant: for a theoretical understanding, this approach simplifies the problem while for a computer-aided search for a good π parameter, this significantly reduces the search space. In Definition 33, we specify what it means for two permutation sequences to be equivalent.

Definition 33 (Equivalence of permutation sequences)**.** *Two permutation sequences π, π', for a T-round AES-like cipher are said to be* equivalent*, denoted $\pi \sim \pi'$,* if and only if *for all possible branch numbers B, the equality $\widetilde{\mathsf{AES}}_{M,N}(\pi, B) = \widetilde{\mathsf{AES}}_{M,N}(\pi', B)$ holds. Intuitively, this means that for all AES-like ciphers using π, there is an AES-like cipher using π' which it is functionally identical to, up until the last round.*

We remark that, using this notion of equivalence, one can transform each cipher $\mathscr{E}_K$ using π into a cipher $\mathscr{E}'_K$ using π' such that $\mathscr{E}_K = \tau \circ \mathscr{E}'_K$ for a permutation τ on the state words. Thus, equivalence will imply the same number of tightly guaranteed active S-boxes for all possible fixed branch numbers B.

Equivalences for Permutation Sequences

In order to prove the reduction to a normalized form on the round permutations, we show a range of observations in the following. First, Lemma 5 is a combinatorial result on permutations on Cartesian products.

Lemma 5 (Representation of permutations on Cartesian products). *Every permutation π_t on the words of an $M \times N$ state can be represented as $\pi_t = \gamma' \circ \phi \circ \gamma$ where γ, γ' are permuting the words within each column (separately from other columns) and ϕ is permuting the words within each row (separately from other rows).*

Proof. Let $T_A, T_B, T_C, T_D \in (\mathbb{Z}_M \times \mathbb{Z}_N)^{M\times N}$ s.t. $T_{A_{i,j}} = (i,j)$ and let T_B, T_C and T_D be defined s.t. $T_B = \gamma(T_A)$, $T_C = \phi(T_B)$, and $T_D = \gamma'(T_C)$, i.e.

$$T_A \xrightarrow{\gamma} T_B \xrightarrow{\phi} T_C \xrightarrow{\gamma'} T_D. \tag{3.16}$$

To show the result, we let $T_D = \pi_t(T_A)$ and show how to construct the permutations such that $T_D = (\gamma' \circ \phi \circ \gamma)(T_A)$. We first observe the following two properties which must hold:

1. T_B must be a matrix where, within each column $j \in \mathbb{Z}_N$, it holds that (i) the second coordinate of each point is equal to j, because γ only permutes within each column of T_A and (ii) the set of first coordinates cover all of $\mathbb{Z}_M$, because T_B is a permutation of $\mathbb{Z}_M \times \mathbb{Z}_N$.

2. T_C must be a matrix where, for each column $j \in \mathbb{Z}_N$, the points in column j of T_C are the same as those in column j of T_D. This is required because otherwise it is impossible to go between T_C and T_D using a permutation operating in each column.

If we can determine a matrix T_B with property (1) and a row permutation ϕ s.t. $T_C = \phi(T_B)$ has property (2), we are clearly done, because T_A and T_D can be obtained from T_B respectively T_C by applying a permutation moving the words inside each column.

For a matrix $A \in (\mathbb{Z}_M \times \mathbb{Z}_N)^{M\times N}$, let $Q(A)$ be an $N \times N$ matrix for which $Q(A)_{i,j}$ is the number of occurences of $j \in \mathbb{Z}_N$ in the second coordinate of the points in column $i \in \mathbb{Z}_N$ of A. As $Q(T_B)$ and $Q(T_C)$ are both magic squares of weight M, it follows from the Birkhoff-von Neumann Theorem (see e.g. [32, p. 164]), that one can decompose $Q(T_C)$ into a sum of M permutation matrices, and thus

$$Q(T_C) = P_0 + \cdots + P_{M-1}. \tag{3.17}$$

Let ϕ be a permutation within each row, defined by applying P_i to row $i \in \mathbb{Z}_M$. Then $Q(\phi(T_B)) = Q(T_C)$.

What is left to show is that there exists a T_B obtained by moving words inside the columns of T_A s.t. the first coordinates in each column j of T_C is correct, given the fixed permutation ϕ. To see this, consider the case where T_C requires a point (a,b) to be in column j. Clearly, (a,b) is in column b of both T_A and T_B. Now, let P_i be such that it moves *some* point in position (a',b) of

T_B from column b to column j of T_C. If $(a', b) = (a, b)$, then (a, b) does not need to be moved within column b from T_A to T_B by γ, but if $(a', b) \neq (a, b)$, one can use γ to move (a, b) to (a', b) so it ends up in column j of T_C. As each point (a, b) will only be present once in T_C, it can be moved once between T_A and T_B and never moved again. This procedure holds for all points (a, b), and as such the result follows[2]. □

Lemma 6 (Equivalence under permutations within columns). *Let $\pi = (\pi_0, \ldots, \pi_{T-1})$ be a permutation sequence for a T-round AES-like cipher $\mathcal{E}_K$ and let γ, γ' be arbitrary permutations on the words within the each column of a state. Then, for all $t = 0, \ldots, T-1$, it holds that*

$$\pi \sim (\pi_0, \ldots, \gamma' \circ \pi_t \circ \gamma, \ldots, \pi_{T-1}). \tag{3.18}$$

In particular, the number of tightly guaranteed active S-boxes is invariant under inserting permutations, before and after any π_t, which act on the columns of the state separately.

Proof. Fix the branch number B and let $\mathcal{E}_K \in \widetilde{\mathsf{AES}}_{M,N}(\pi, B)$. We consider any round $t \in \mathbb{Z}_T$. We first show that $\pi \sim \pi' = (\pi_0, \ldots, \pi_t \circ \gamma, \ldots, \pi_{T-1})$. Let $\mathcal{E}'_K$ be like $\mathcal{E}_K$ but using permutation sequence π', with rounds denoted $F'_t, t \in \mathbb{Z}_T$. Thus, $\mathcal{E}'_K \in \widetilde{\mathsf{AES}}_{M,N}(\pi', B)$. It holds that

$$F'_t = \mathsf{Permute}_{\pi_t} \circ \mathsf{Permute}_{\gamma} \circ \mathsf{MixColumns}_t \circ \mathsf{SubBytes}. \tag{3.19}$$

Since γ operates on the columns separately, one can define

$$\mathsf{MixColumns}'_t = \mathsf{Permute}_{\gamma} \circ \mathsf{MixColumns}_t, \tag{3.20}$$

which in turn is a linear layer for an AES-like cipher with the same branch number, and we have

$$F'_t = \mathsf{Permute}_{\pi_t} \circ \mathsf{MixColumns}'_t \circ \mathsf{SubBytes}. \tag{3.21}$$

Now, $\mathcal{E}'_K$ is a cipher which uses the permutation sequence π and thus $\mathcal{E}'_K \in \widetilde{\mathsf{AES}}_{M,N}(\pi, B)$. The other inclusion follows the same way by applying γ^{-1}.

For showing the case of $\pi' = (\pi_0, \ldots, \gamma' \circ \pi_t, \ldots, \pi_{T-1})$, the argument is parallel. By combining the two, the result follows. □

Definition 34 (ρ-alternating permutation sequence). *Let T and ρ be positive integers and consider a T-round AES-like cipher $\mathcal{E}_K$. The vector $\pi = (\pi_0, \ldots, \pi_{\rho-1})_T$ is called a* ρ-alternating permutation sequence over T rounds, *if $\mathcal{E}_K$ alternatingly repeats the π_i permutations over T rounds, such that the permutation π_i, $0 \leq i < \rho$, is used in rounds t where $t \equiv i \bmod \rho$, $0 \leq t < T$.*

As an easy result, one obtains Theorem 4, which we state without proof.

Theorem 4 (Reduction to permutations on the rows). *Let $\pi = (\pi_0, \ldots, \pi_{\rho-1})_T$ be a ρ-alternating permutation sequence. Then one can construct a $\pi' = (\pi'_0, \ldots, \pi'_{\rho-1})_T$ with $\pi \sim \pi'$, s.t. for each $t \in \mathbb{Z}_\rho$, it holds that π'_t permutes only the words within each row of the state.*

[2]We would like to thank John Steinberger for aiding with this proof

Equivalences for Rotation Matrices

Until this point we have focused on AES-like ciphers with arbitrary word-wise permutations $\mathsf{Permute}_{\pi_t}$ as part of the round function. However, due to their implementation characteristics, such general permutations are not suitable for designs of cryptographic primitives. To that end, we limit ourselves from this point on to AES-like ciphers where the permutation operation of the round function *cyclically rotates each row of the state from left-to-right* using a rotation matrix as specified in Definition 35.

Definition 35 (Rotation matrix). *Consider an AES-like cipher where the permutation operation in the round function consists of cyclic word-wise rotations of each state row. For such a cipher, we define a* rotation matrix *as a matrix $\sigma \in \mathbb{Z}_N^{\rho \times M}$, where ρ is a positive integer, such that*

1. *If $\rho = T$, then $\sigma_{t,i}$ denotes the rotation amount for row $i \in \mathbb{Z}_M$ in round t, and*
2. *If $\rho < T$, then we have the further requirement that the rotation constants alternate, such that $\sigma_{k,i}$ denotes the rotation amount for row $i \in \mathbb{Z}_M$ in rounds t where $t \equiv k \bmod \rho$,*

where, without loss of generality, we let the rotation direction be left-to-right.

As rotation matrices are a special case of arbitrary permutations, we remark that the notion of equivalence includes these as well. We simplify our notion of an AES-like cipher to only use row-wise rotations in the permutation part of each F_t. In particular, we substitute the $\mathsf{Permute}_{\pi_t}$ operation by

$$\begin{aligned} &\mathsf{ShiftRows}_{\sigma_t} : (\mathbb{F}_{2^m})^{M \times N} \to (\mathbb{F}_{2^m})^{M \times N} \\ &\forall i \in \mathbb{Z}_M, \forall j \in \mathbb{Z}_N : X_{i,j} \mapsto X_{i,j+\sigma_{t \bmod \rho, i} \bmod N}. \end{aligned} \tag{3.22}$$

Example 2. *For the AES, we have $M = N = 4$ and $\rho = 1$ with $\sigma = \begin{pmatrix} 0 & 3 & 2 & 1 \end{pmatrix}$.*

Next, we continue by giving results on the equivalence of rotation matrices.

Lemma 7 (Equivalence under re-ordering of row entries). *Let $\sigma \in \mathbb{Z}_N^{\rho \times M}$ be a rotation matrix and let $\vartheta_0, \ldots, \vartheta_{\rho-1}$ be arbitrary permutations such that ϑ_t permutes the elements in row t of σ. Define σ' s.t. $\forall t \in \mathbb{Z}_\rho : \sigma'_t = \vartheta_t(\sigma_t)$. Then $\sigma \sim \sigma'$.*

Proof. This directly follows from Lemma 6, as using σ'_t is equivalent to using $\gamma' \circ \sigma_t \circ \gamma$ for appropriate permutations γ' and γ, permuting the words in each column of the state separately. □

Lemma 8 (Equivalence under row-wise constant addition). *Let $\sigma \in \mathbb{Z}_N^{\rho \times M}$ be a rotation matrix and let $c_0, \ldots, c_{\rho-1} \in \mathbb{Z}_N$. Define a rotation matrix σ' where $\forall t \in \mathbb{Z}_\rho, \forall i \in \mathbb{Z}_M : \sigma'_{t,i} = \sigma_{t,i} + c_t \bmod N$. Then $\sigma \sim \sigma'$.*

Proof. We split the proof into two cases, depending on the number of rounds.

Case $T \le \rho$. If $T < \rho$, one can add constants to $\sigma_T, \dots, \sigma_{\rho-1}$, since these are never used anyway. Thus, let us consider $T = \rho$. We give a proof by induction that one can add independent constants $c_t, \dots, c_{T-1}$ to $\sigma_t, \dots, \sigma_{T-1}$ to obtain an equivalent rotation matrix σ', and proceed by induction on t. Clearly, one can add a constant to σ_{T-1} to obtain an equivalent σ', since the set $\widetilde{\mathsf{AES}}_{M,N}(\sigma, B)$ does not cover the use of σ_{T-1}. Assuming the statement holds for $t, \dots, T-1$, we now prove that it is possible to add a constant c_{t-1} to σ_{t-1} as well. Using the notation that $\mathsf{SR} = \mathsf{ShiftRows}$, $\mathsf{MC} = \mathsf{MixColumns}$, $\mathsf{SB} = \mathsf{SubBytes}$ and RS_k is a rotation of the whole state by k positions, we have

$$\begin{aligned} F_t \circ F_{t-1} &= (\mathsf{SR}_{\sigma_t} \circ \mathsf{MC}_t \circ \mathsf{SB}) \circ (\mathsf{RS}_{-c_{t-1}} \circ \mathsf{RS}_{c_{t-1}}) \circ (\mathsf{SR}_{\sigma_{t-1}} \circ \mathsf{MC}_{t-1} \circ \mathsf{SB}) \\ &= \mathsf{SR}_{\sigma_t} \circ \mathsf{RS}_{-c_{t-1}} \circ \mathsf{RS}_{c_{t-1}} \circ \mathsf{MC}_t \circ \mathsf{RS}_{-c_{t-1}} \circ \mathsf{SB} \circ (\mathsf{RS}_{c_{t-1}} \circ \mathsf{SR}_{\sigma_{t-1}} \circ \mathsf{MC}_{t-1} \circ \mathsf{SB}), \end{aligned} \tag{3.23}$$

since $\mathsf{RS}_{-c_{t-1}}$ commutes with SB. Now, since $\mathsf{MC}'_t = \mathsf{RS}_{c_{t-1}} \circ \mathsf{MC}_t \circ \mathsf{RS}_{-c_{t-1}}$ defines a (just rotated) linear column mixing, and since SR_{σ_t} commutes with $\mathsf{RS}_{-c_{t-1}}$, we have

$$F_t \circ F_{t-1} = (\mathsf{RS}_{-c_{t-1}} \circ \mathsf{SR}_{\sigma_t} \circ \mathsf{MC}'_t \circ \mathsf{SB}) \circ (\mathsf{RS}_{c_{t-1}} \circ \mathsf{SR}_{\sigma_{t-1}} \circ \mathsf{MC}_{t-1} \circ \mathsf{SB}), \tag{3.24}$$

and we see that by adding c_{t-1} to σ_{t-1} and $-c_{t-1}$ to σ_t we obtain an equivalent σ'. The result now follows by induction, since the addition of $-c_{t-1}$ to σ_t can be undone by the induction assumption.

Case $T > \rho$. For the case $T > \rho$, let H be a $T \times M$ matrix where $H_t = \sigma_k$ when $t \equiv k \bmod \rho$. For a T-round AES-like cipher $\mathscr{E}_K$, H and σ are clearly equivalent rotation matrices. From the above, it follows we can add c_t to row t of H, $t \in \mathbb{Z}_T$, and obtain an equivalent H'. In particular, adding the same c_k to all rows t where $t \equiv k \bmod \rho$, we obtain H' which is equivalent to σ, and has the property that $H'_i = H'_j$ if $i \equiv j \bmod \rho$, and in particular the first ρ rows of H' equals σ' and the result follows. □

Theorem 5 (Equivalence for rotation matrices). *Given a rotation matrix $\sigma \in \mathbb{Z}_N^{\rho \times M}$, one can obtain an equivalent matrix $\sigma' \in \mathbb{Z}_N^{\rho \times M}$ for which the following holds simultaneously*

1. *Each row $\sigma'_t, t \in \mathbb{Z}_\rho$ is ordered increasingly by value,*
2. *For all $t \in \mathbb{Z}_\rho$ it holds that $\sigma'_{t,0} = 0$ and*
3. *For all $t \in \mathbb{Z}_\rho$ it holds that $\sigma'_{t,1} \le \frac{N}{2}$.*

Proof. Points (1) and (2) follow directly from Lemma 7 and 8, respectively. For point (3), let us assume without loss of generality that (1) and (2) hold, and take the case where $M \ge 2$ and consider the element $\sigma_{t,1}$ from some row σ_t, $0 \le t < \rho$. If $\sigma_{t,1} > \frac{N}{2}$, we add $-\sigma_{t,1} \bmod N$ and the result follows from Lemmas 7 and 8. □

Besides Theorem 5, we heuristically suggest a search for optimal rotation matrices to restrict itself to matrices where all entries in a row are different, i.e. $\forall t \in \mathbb{Z}_\rho : \sigma_{t,j} = \sigma_{t,j'} \Leftrightarrow j = j'$, as equal entries in some σ_t are redundant with respect to the diffusion properties of the cipher.

Moreover, when N is even, we require that σ contains at least one odd entry, because otherwise even-numbered columns never mix with odd-numbered columns. In fact, extending this, we require that σ should contain at least one element which does not divide N. However, due to the very few cases, we deem that implementing this restriction is computationally more cumbersome than actually including the cases.

We refer to a rotation matrix which satisfies the properties mentioned, plus properties (1) through (3) of Theorem 5, as the *normal form* of its equivalence class of rotation matrices. In the next section, we turn more practical, as we discuss an approach of actually obtaining results for specific AES-like cipher instances.

3.1.5 Mixed-Integer Linear Programming and Experimental Results

As mentioned, in our consideration of the problem of bounding the number of active S-boxes in a trail, for a particular AES-like cipher, we consider the S-boxes and linear layers as black box operations. One advantage to modeling those operations as such is, that one can easily compute useful lower bounds on the number of guaranteed active S-boxes using a mixed-integer linear programming approach. We describe this approach next.

Formulating the MILP Program

In the following, we describe the mixed-integer linear program which models the problem of determining the tightly guaranteed trail weight under a given rotation matrix $\sigma \in \mathbb{Z}_N^{\rho\times M}$. We give the parameters, decision variables, the constraints and the target optimization in Model 1. We remark that the use of a MILP model to determine bounds on the number of active S-boxes over a cipher is a quite common approach. An example application to the AES is by Mouha, Wang, Gu, and Preneel [237]. Other works employing MILP approaches include that of Borghoff, Knudsen, and Stolpe [82], Bogdanov [71, 72] and Wu and Wang [300]. We note that Model 1 is specified for the case where each $\mathbf{M}_j^t$ used in the $\mathsf{MixColumns}_t$ operation is a *maximum distance separable* (MDS) matrix, i.e. a matrix such that its left multiplication leads to a linear automorphism θ of branch number $B_\theta = M + 1$, as this is usually what is applied in designs. If, on the other hand, non-MDS matrices are deployed, the model can be easily modified to cover these cases as well, at the cost of a slightly more complicated model. Theorem 6 formalizes how Model 1 provides us with the sought bound.

Theorem 6. *The solution of Model 1 is always a lower bound on the number of tightly guaranteed active S-boxes for an AES-like cipher with branch number B and rotation matrix σ. If the branch number is optimal for the given dimensions and a linear mixing layer with this branch number exists (and the word length $m > \log_2(M+2)$), this provides a tight bound.*

Proof. This follows from Corollary 3 in Appendix C. □

Experimental Results

A major part of our contribution in this section consists of a wide range of optimal choices of rotation matrices for various state geometries $M \times N$, ρ and number of rounds T. For all our

Model 1: MILP model for determining the guaranteed trail weight in an AES-like cipher using the parameters specified

Parameters

Name	Domain	Description
M	$\mathbb{N}$	Number of rows in state
N	$\mathbb{N}$	Number of columns in state
T	$\mathbb{N}$	Number of rounds
ρ	$\mathbb{N}$	Number of rows in rotation matrix σ
B	$\mathbb{N}$	Branch number of AES-like cipher
σ	$\mathbb{Z}_N^{\rho\times M}$	Rotation matrix

Decision variables

Name	Domain	Index domain	Description
$\tilde{X}^t_{i,j}$	$\mathbb{F}_2$	$i \in \mathbb{Z}_M, j \in \mathbb{Z}_N, t \in \mathbb{Z}_T \cup \{T\}$	$\tilde{X}^t_{i,j} = 1$ *if and only if* the word in position (i,j) is active in the state input to F_t
a^t_j	$\mathbb{F}_2$	$j \in \mathbb{Z}_N, t \in \mathbb{Z}_T$	Auxiliary variable; $a^t_j = 1$ *if and only if* column j has an active word in the state input to F_t

Minimize

$$\sum_{t\in\mathbb{Z}_T}\sum_{i\in\mathbb{Z}_M}\sum_{j\in\mathbb{Z}_N}\tilde{X}^t_{i,j}$$

subject to

$$\sum_{i\in\mathbb{Z}_M}\sum_{j\in\mathbb{Z}_N}\tilde{X}^0_{i,j} \geq 1 \qquad (1)$$

$$\forall t \in \mathbb{Z}_T, \forall j \in \mathbb{Z}_N \quad : \quad \sum_{i\in\mathbb{Z}_M}\tilde{X}^t_{i,j} + \tilde{X}^{t+1}_{i,(j+\sigma_{t \bmod \rho, i}) \bmod N} \geq B \cdot a^t_j \qquad (2)$$

$$\forall t \in \mathbb{Z}_T, \forall i \in \mathbb{Z}_M, \forall j \in \mathbb{Z}_N : \quad a^t_j \geq \tilde{X}^t_{i,j} \qquad (3)$$

experiments, we concentrated on the case of MDS $\mathsf{MixColumns}_t$ layers, i.e. AES-like ciphers with optimal branch number. Using the heuristic approach from Section 3.1.4, i.e. by brute-forcing the normal form of each equivalence class of rotation matrices, we provide optimal solutions for the analyzed cases as per Theorem 6. The full table of results is given in Appendix C. The results were obtained by an implementation iterating over all normal forms of rotation matrices, and for each of them making a call to an MILP solver to obtain the optimal trail weight assuming that particular rotation matrix. The solver used was IBM ILOG CPLEX [100]. We remark that our program can be made to work with other solvers such as Gurobi [287]. The source code for our program is available as [43].

We highlight in Table 3.2 results which suggest improvements for some existing AES-like primitives. We see that in some cases, direct replacement of σ yields better (i.e. increased) bounds, while in other cases one must increase ρ to obtain better bounds. Among our findings are tight bounds which are not a multiple of the branch number for an even number of rounds. This implies that there exists some MDS linear mixing layers such that the lightest valid trail contains a two-round subtrail of weight more than B. Thus, some optimal trails have non-optimal transitions locally.

Table 3.2: Improvements for existing AES-like primitives. An entry $(\rho_P, \mathscr{B}_P)/(\rho_M, \mathscr{B}_M)$ gives ρ and the number of tightly guaranteed S-boxes $\mathscr{B}$ in a T-round trail for the *primitive* (subscript P) and the *modified primitive* (subscript M), respectively. The † symbol indicates results where only diffusion-optimal σ were tested, which means actual obtainable bounds may be higher.

Rounds	Rijndael-192	Rijndael-256	PRIMATEs-80	PRØST-128
5	–	–	$(1,54)/(2,56)$	–
6	$(1,42)/(1,45)$	$(1,50)/(2,55)$	–	$(2,85)/(2,90)^{\dagger}$
7	$(1,46)/(1,48)$	–	–	$(2,96)/(2,111)^{\dagger}$
8	$(1,50)/(1,57)$	–	–	–
10	–	$(1,85)/(2,90)$	–	–
12	$(1,87)/(1,90)$	$(1,105)/(2,111)$	–	–

3.1.6 Optimal Solutions

In this section we describe, for special cases of the state geometry, optimal solutions with respect to both our main criteria, i.e. with respect to diffusion properties on one hand and resistance towards differential- and linear attacks on the other hand.

Diffusion-Optimal Rotation Matrices

Under the assumptions that each S-box $S : \mathbb{F}_{2^m} \to \mathbb{F}_{2^m}$ and each $\mathbf{M}_j^t$ matrix has the property that each output bit depends on each input bit, we describe in the following a way of tracking the diffusion properties for an AES-like cipher $\mathscr{E}_K$. Let z denote an arbitrary fixed bit of an input to $\mathscr{E}_K$. When, in the beginning of a round, a single bit in a column depends on z, then each bit in the column will depend on z after applying $\mathsf{MixColumns}_t \circ \mathsf{SubBytes}$. Thus, with fixed parameters M, N and σ, determining how many rounds t are required to obtain full diffusion reduces to answering how many rounds are required to have at least one bit depending on z in each column: if this is obtained after t' rounds then full diffusion is obtained after $t = t' + 1$ rounds. This is formalized in the following.

Definition 36 (Sumset). *Let G be an additive group and let $A, B \subset G$. We define the* sumset*, written $A + B$, as*

$$A + B = \{a + b \mid a \in A \wedge b \in B\}, \tag{3.25}$$

where the sum is over G. We write kA for the sumset $A + A + \cdots + A$ with k terms.

Theorem 7. *Consider an AES-like cipher with fixed parameters M, N, ρ and σ. Let without loss of generality z denote a bit in the word $X_{0,0}$ for an input X. Let $\alpha(t)_i$, $0 \leq i < \rho$, denote the number of times σ_i is used in a* ShiftRows *operation during t rounds of the cipher. Then, after T rounds, the indices of columns which contain bits depending on z are given by the sumset*

$$\alpha(T)_0\sigma_0 + \alpha(T)_1\sigma_1 + \cdots + \alpha(T)_{\rho-1}\sigma_{\rho-1}, \tag{3.26}$$

where addition is over $\mathbb{Z}_N$, and where we abuse notation and consider σ_t as a subset of $\mathbb{Z}_N$.

Proof. Let $S_{-1} = \{0\}$. We recursively define $S_t = \{v + s \mid s \in S_{t-1} \wedge v \in \sigma_{t \bmod \rho}\}$ for $t \geq 0$, where addition is in $\mathbb{Z}_N$. Note that the set S_t corresponds exactly to the sumset $\alpha(t)_0\sigma_0 + \cdots + \alpha(t)_{\rho-1}\sigma_{\rho-1}$. Clearly, $S_0 = \{v \mid v \in \sigma_0\}$ is the set of indices of columns that contain words depending on z after round F_0. Now, assume that S_t contains the column indices which has some word depending on z after F_t. Then, after applying $\mathsf{MixColumns}_{t+1}$, *all* words in columns $j \in S_t$ depend on z. Now, when we apply $\mathsf{ShiftRows}_{\sigma_{t+1 \bmod \rho}}$, the words depending on z are moved exactly to the indices given in S_{t+1}, and thus the result is obtained by induction. □

Corollary 2. *Consider an AES-like cipher with fixed parameters M, N, ρ and σ. If t' is the smallest positive integer s.t. the sumset $\alpha(t')_0\sigma_0 + \cdots + \alpha(t')_{\rho-1}\sigma_{\rho-1}$ over $\mathbb{Z}_N$ generates all of $\mathbb{Z}_N$, then the cipher obtains full diffusion after $t = t' + 1$ rounds.*

Proof. The proof follows from Theorem 7. Note that we chose the input bit z from the word $X_{0,0}$. If it would be chosen from an arbitrary word $X_{i,j}$, the corresponding sumset would be shifted by some constant c. However, these are the same sumsets for all possible c, since they generate all of $\mathbb{Z}_N$. □

Theorem 8. *When $N = M^\rho$, a diffusion-optimal rotation matrix is $\sigma \in \mathbb{Z}_N^{\rho \times M}$ s.t. $\sigma_{t,i} = i \cdot M^t$ for $(t, i) \in \mathbb{Z}_\rho \times \mathbb{Z}_M$, i.e.*

$$\sigma = \begin{pmatrix} 0 & 1 & 2 & \cdots & M-1 \\ 0 & M & 2M & \cdots & (M-1)M \\ 0 & M^2 & 2M^2 & \cdots & (M-1)M^2 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & M^{\rho-2} & 2M^{\rho-2} & \cdots & (M-1)M^{\rho-2} \\ 0 & M^{\rho-1} & 2M^{\rho-1} & \cdots & (M-1)M^{\rho-1} \end{pmatrix}, \tag{3.27}$$

or any σ' where the rows of σ are permuted. These obtain full diffusion after $\rho + 1$ rounds.

Proof. By Theorem 7, the set of indices of columns containing a word depending on z after ρ rounds is given by the sumset $\sigma_0 + \cdots + \sigma_{\rho-1}$ over $\mathbb{Z}_N$. This sumset has $M^\rho = N$ sums, and thus equals $\mathbb{Z}_N$ *if and only if* no two sums in the sumset are equal. To see why this is the case, consider constructing M-adic numbers using the sums in the sumset. We pick exactly one element from each row of σ and add them. As the elements in row t are $\sigma_t = \begin{pmatrix} 0M^t & 1M^t & \cdots & (M-1)M^t \end{pmatrix}$, the choice for the sum from σ_t is the tth least significant digit in the M-adic representation of that number. In other words, the rows of σ form a base for the M-adic number system, and we can form any number up to $\sum_{t=0}^{\rho-1}(M-1)M^t = N - 1$ with it. Since M^ρ elements cannot be generated using less than ρ parameters in the sumset, the diffusion-optimaltiy of σ follows. □

Example 3. *Theorem 8 implies that the AES has a diffusion-optimal choice of $\sigma = \begin{pmatrix} 0 & 3 & 2 & 1 \end{pmatrix}$ for its parameters $N = M = 4$ and $\rho = 1$.*

Trail-Optimal Solutions

In this section, we first state Theorem 9 which is of particular interest because of the large number of AES-like ciphers with square geometry, i.e. with $M = N$. Considering its statement, square states can be understood quite well. We also give a conjecture on the optimality of guaranteed trail weights for $M \times M^n$ AES-like ciphers over $T = 2^{n+1}$ rounds and give a construction which matches the conjectured bound.

Theorem 9 (Optimality for square geometries)**.** *Let σ be a rotation matrix in normal form operating on a square state of dimension $M \times M$. Then the number of tightly guaranteed active S-boxes is invariant under increasing ρ. In particular, any σ has $\sigma \sim \begin{pmatrix} 0 & 1 & \cdots & M-1 \end{pmatrix}$. Furthermore, assuming the existence of at least one MDS linear layer and the word length $m > \log_2(M+2)$, we have $\sigma \xrightarrow{M+1} k(M+1)^2$ over $4k$ rounds for all $k \in \mathbb{N}$.*

Proof. As for any $\rho > 1$, each row σ_t of a rotation matrix σ in normal form will equal $\begin{pmatrix} 0 & 1 & \cdots & M-1 \end{pmatrix}$, or any permutation hereof, this is equivalent to having $\rho = 1$ by Lemma 7. In order to prove the second statement, we first apply the *Four-Round Propagation Theorem* [105, Theorem 3] of the AES in a repeated manner, which provides the stated $k(M+1)^2$ as a lower bound. It is left to argue that there exists a valid $4k$-round trail of weight $k(M+1)^2$ for some $\mathcal{E}_K$ using the specific parameters. Therefore, we first define a four-round trail X of weight $(M+1)^2$ as

$$X = \left(\begin{pmatrix} 1 & 0 & \cdots & 0 \\ 0 & 0 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 0 \end{pmatrix}, \begin{pmatrix} 1 & 0 & \cdots & 0 \\ 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 1 \end{pmatrix}, \begin{pmatrix} 1 & 1 & \cdots & 1 \\ 1 & 1 & \cdots & 1 \\ \vdots & \vdots & \ddots & \vdots \\ 1 & 1 & \cdots & 1 \end{pmatrix}, \begin{pmatrix} 1 & 0 & \cdots & 0 \\ 0 & 0 & \cdots & 1 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 1 & \cdots & 0 \end{pmatrix}, \begin{pmatrix} 1 & 0 & \cdots & 0 \\ 0 & 0 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 0 \end{pmatrix} \right). \tag{3.28}$$

By repeating this structure k times, one can define a $4k$-round trail of weight $k(M+1)^2$. For the validity of this trail for some $\mathcal{E}_K$, one can see that it is obtainable by only using the identity as the S-box and existing mixing steps, applying Corollary 3 in Appendix C. □

Theorem 9 implies that a designer of an AES-like primitive who wants to improve upon the bound for a square dimension necessarily has to choose a rotation parameter σ consisting of at least one σ_t which breaks the normal form structure. Intuitively, this would not only provide a worse bound but also worse diffusion properties. However, giving an argument for the trail-optimality considering all possible rotation matrices (respectively permutations), seems quite difficult. For the special case of a hypercubed geometry, we give Conjecture 1.

Conjecture 1. *Let n be a positive integer. Given the state dimension $M \times M^n$ for an AES-like cipher, then a trail-optimal choice of the permutation sequence π over $T = 2^{n+1}$ rounds yields $\pi \xrightarrow{M+1} (M+1)^{n+1}$.*

The Superbox Argument. The superbox argument is a commonly used proof technique to lower bound the number of active S-boxes in an AES-like cipher over a certain number of rounds. It has been used for the AES but also for ECHO [48] and PAEQ [59]. One uses the fact that for a clever choice of the rotation matrix, the round operations can be commuted such that some part of the encryption first works locally, in parallel, on parts of the state which we call *superboxes*. Next, the superboxes are combined using state-wide operations which effectively mix the superboxes together, only to split the state into superboxes again, working with the localized operations. Such a large structure is referred to as a *megabox*, and covers four rounds of the cipher.

One can show that if a superbox has active input, there are at least B active S-boxes in the first two rounds inside this superbox. Now, with the right choice of rotation matrix, the operation that combines the superboxes again imply that for the next two rounds, the total number of active superboxes is at least B. From this, one obtains a four-round lower bound of B^2. This concept, which is the idea behind the Four-Round Propagation Theorem [105, Theorem 3], can be easily generalized by iteration for appropriate dimensions of state in the AES-like cipher, and with an appropriately chosen rotation matrix. We stress, however, that choosing the rotation matrix correctly for the given state dimension is of paramount importance to assuring the argument that one has e.g. B active superboxes in a megabox (or equivalently for higher dimensions).

As mentioned, in Theorem 10, we give a construction which achieves the bound given in Conjecture 1. Note that (especially for a cubed state dimension) this approach is not new in itself. Our main point here is that, in clear distinction to prior work such as that of Daemen, Knudsen, and Rijmen [108] and [59], we present an efficient way of implementing this idea by using *cyclic rotations only*. For a better visualization, Example 4 illustrates this construction for $M = 4$ and $n = 3$.

Theorem 10 (2^{n+1}-round Propagation Theorem). *There exists a rotation matrix $\sigma \in \mathbb{Z}_{M^n}^{2^n \times M}$, such that every (non-zero) valid 2^{n+1}-round trail over all $\mathcal{E}_K \in \widetilde{\mathsf{AES}}_{M,M^n}(\sigma, B)$ has a weight of at least B^{n+1}. The rotations are given by*

$$\begin{aligned} \forall j \in \mathbb{Z}_n : \sigma_{2^{n-j}-2} = \sigma_{2^{n-j}-1} = \begin{pmatrix} 0 & M^j & 2M^j & \cdots & (M-1)M^j \end{pmatrix} \quad \textit{and} \\ \forall j \in \mathbb{Z}_{n-1}, \forall i \in \mathbb{Z}_{2^{n-(j+1)}} : \sigma_i = \sigma_{2^{n-j}-3-i}. \end{aligned} \tag{3.29}$$

Proof. For $n = 1$, the statement is precisely the Four-Round Propagation Theorem for the AES. Therefore, we first prove the result for the eight-round case, i.e. for $n = 2$. We need to show that

$$\sigma = \begin{pmatrix} 0 & M & 2M & \cdots & (M-1)M \\ 0 & M & 2M & \cdots & (M-1)M \\ 0 & 1 & 2 & \cdots & M-1 \\ 0 & 1 & 2 & \cdots & M-1 \end{pmatrix} \xrightarrow{B} \mathscr{B} \tag{3.30}$$

over eight rounds for a $\mathscr{B} \geq B^3$. For the proof, we rely on a straightforward generalization of the Four-Round Propagation Theorem to the dimension one higher than the standard AES, as described previously. In particular, if one can partition the $M \times M^2$ state into M sub-states of M columns each (i.e. consider them as $M \times M$ sub-states), such that in four consecutive rounds, the ShiftRows operating in the first and second rounds shifts each such sub-state as if using the vector $\begin{pmatrix}0 & 1 & \cdots & M-1\end{pmatrix}$, with respect to considering that particular $M \times M$ sub-state, then the number of guaranteed active S-boxes in each such sub-state over four rounds it at least B^2 (assuming a non-zero input difference). Note that the rotations of the third and fourth round have no impact on the four-round trail weight.

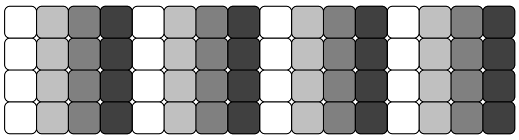


Figure 3.3: Positions of the 4 independent sets of columns in a 4×16 state

Using the σ specified, the first four rounds of $\mathscr{E}_K$ satisfies this property when the M sub-states of size $M \times M$ are taken to be every M^{th} column of the state, as indicated for a 4×16 state in Figure 3.3. The same thing holds when considering the last four rounds separately.

Now, due to the way the row shifting of the third round combines with the column mixing and row shifting of the fourth round, i.e. $\mathsf{ShiftRows}_{\sigma_3} \circ \mathsf{MixColumns} \circ \mathsf{SubBytes} \circ \mathsf{ShiftRows}_{\sigma_2}$, each $M \times M$ sub-state mixes completely with each of the $M \times M$ sub-states. As such, like in the Four-Round Propagation Theorem, the sum of active $M \times M$ sub-states from the third and fourth round is at least B. Combining this observation with the generalized Four-Round Propagation Theorem, the result of $B \cdot B^2$ follows.

The general case is now obtained by induction: in order to do the iteration to $2^{(n+1)+1}$ rounds, one has to apply the 2^{n+1}-round propagation. □

Example 4. *Let $M = 4$, $n = 3$ and $B = 5$. The state has geometry $M \times N = 4 \times 64$. The guaranteed trail weight of* 625 *active S-boxes over $T = 16$ rounds can be realized using the rotation matrix*

$$\sigma = \begin{pmatrix} 0 & 16 & 32 & 48 \\ 0 & 16 & 32 & 48 \\ 0 & 4 & 8 & 12 \\ 0 & 4 & 8 & 12 \\ 0 & 16 & 32 & 48 \\ 0 & 16 & 32 & 48 \\ 0 & 1 & 2 & 3 \\ 0 & 1 & 2 & 3 \end{pmatrix}. \tag{3.31}$$

We remark that especially for higher dimensions, a rotation matrix following this construction is not of much practical interest, as the diffusion properties are far from optimal. One open question is whether it is possible, for general M, to obtain these bounds without using a rotation matrix that allows a proof using a superbox-like argument. For the special case of $M = 2$ and $N = 4$, we found the example rotation matrix

$$\sigma = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 1 & 2 & 1 & 1 \end{pmatrix}^T, \tag{3.32}$$

which contains no superbox structure, and which yields $\sigma \xrightarrow{3} 27$ over $T = 8$ rounds.

3.1.7 Discussion and Conclusions

For AES-like ciphers, the linear mixing layer, often denoted MixColumns, is very well understood: one typically chooses mixing layers defined by MDS matrices to obtain optimal branch numbers. In sharp contrast to this, no systematic approach has been conducted to understand how the word-wise permutation layer in such ciphers affects the diffusion properties and resistance towards differential- and linear attacks. With the work described in this section, we have taken a large step to close that gap.

Specifically, we considered arbitrary word-wise permutations, with special focus on rotations due to their elegant implementation characteristics. We formalized the concept of AES-like ciphers, guaranteed trail weights and equivalence of permutation parameters and, using these formalizations, proved a range of results which reduces the consideration to a special normalized form. These results are employed in practice by connecting it with mixed-integer linear programming models for determining the guaranteed trail weights. To that end, we give a range of optimal word-wise rotations and improve on existing parameters for Rijndael-192, Rijndael-256, PRIMATEs-80 and PRØST-128.

Using superbox-like arguments we are able, as a separate result, to show for specific state geometries that a seemingly optimal bound on the trail weight can be obtained using cyclic rotations only for the permutation layer, i.e. in a very implementation-friendly way. Also coming out of our analysis is the observation that square state geometries are, in some sense, ideal when it comes to solving the problem of determining the best word-wise rotations, as there is just one solution which is optimal.

3.2 PRØST: Permutation-Based Authenticated Encryption

In this section, we describe the AEAD scheme PRØST which was a proposal for the CAESAR competition, an ongoing effort in the cryptographic community channeled by Dan Bernstein. A committee of members from both industry and academia are to select a final portfolio of new AEAD schemes, which should improve upon the current standards. The second-round candidates were announced on July 7, 2015, and PRØST was not among them.

Publications

The results presented in this section are from the PRØST CAESAR submission document [181] and a technical addendum proving the security of PRØST [206]:

[181] Elif Bilge Kavun, Martin M. Lauridsen, Gregor Leander, Christian Rechberger, Peter Schwabe, and Tolga Yalçın. Prøst. Submission to the CAESAR competition, 2014.

[206] Martin M. Lauridsen. Security Proofs for Prøst. http://proest.compute.dtu.dk/proofs.pdf, 2015.

Author Contribution

The author is one of the principal designers behind PRØST. Especially, the author contributed: the determination of rotation offsets used in the ShiftPlanes operation (see Section 3.2.2); analysis of bounds on trail weights; analysis of diffusion properties; descriptions of the use of the PRØST permutation in the existing modes of operation; proofs of security for the proposed schemes; and parts of the reference implementations.

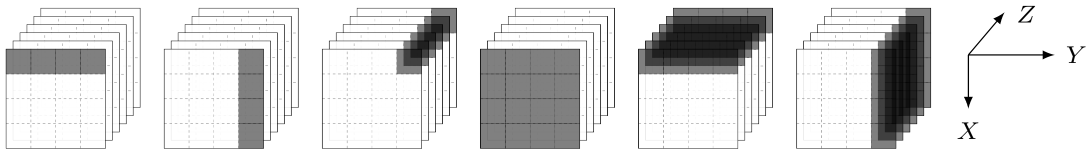
3.2.1 Introduction and Motivation

In the following, we specify PRØST, a new highly secure and efficient permutation with strong security bounds, suitable for a wide range of platforms and modes. We show how the PRØST permutation can be used in conjunction with existing AEAD modes of operation COPA and OTR, which are block cipher-based, and APE, which is permutation-based, all of which we described in Section 1.2.4. The combination of the PRØST permutation with those three modes of operation comprise the AEAD proposals for the CAESAR competition, as described above.

3.2.2 The PRØST Permutation

This section defines the PRØST permutation. In the following, we let $2n$ denote the size of the PRØST permutation. We remark, that this is in contrast to Section 1.2.4, where we used n to denote the size of the permutation, when introducing the duplexed sponge construction. For the PRØST submission document, we used PRØST-n to denote the permutation on $2n$ bits, because n gives the security level in bits. In the treatment here, we stick to this notation despite the confusion it may cause, as to adhere to the established notation. We expand on the theoretical security claims and proofs for PRØST in Section 3.2.6.

The PRØST state S is considered as a three-dimensional block of height h, width w and depth d, along the X, Y and Z axes, respectively, c.f. Figure 3.4g. When referring to the parts of the state, we use the same nomenclature as for the KECCAK permutation [51], as specified by Figure 3.4. We also refer to a lane of the state as a *register*.



(a) Row **(b)** Column **(c)** Lane **(d)** Slice **(e)** Plane **(f)** Sheet **(g)** Axes

Figure 3.4: Nomenclature for PRØST state parts

With the orientation of the (X, Y, Z) axes, we use $S_{X,Y}$ to refer to the d-bit register in row X and column Y of a state S, and we use $S_{X,Y,Z}$ to denote the Z^{th} bit of $S_{X,Y}$. For PRØST, we always have $h = w = 4$, so $2n = 16d$. As such, the embedding of the state S into the (X, Y)-plane results in a 4×4 matrix

$$S = \begin{pmatrix} S_{0,0} & S_{0,1} & S_{0,2} & S_{0,3} \\ S_{1,0} & S_{1,1} & S_{1,2} & S_{1,3} \\ S_{2,0} & S_{2,1} & S_{2,2} & S_{2,3} \\ S_{3,0} & S_{3,1} & S_{3,2} & S_{3,3} \end{pmatrix}, \tag{3.33}$$

where each $S_{X,Y}$ is a d-bit register.

Note 6. *We specify* PRØST *for $d \in \{16, 32\}$, so effectively we specify two permutations. To be sure to avoid confusion, we mention again that we denote the permutation according to its* security level*, which is half the state size, rather than the state size. As such, we denote the* PRØST *permutation on $2n$ bits by* PRØST-*n*.

The PRØST permutation consists of compositions of permutations which we refer to as *rounds*, borrowing from the design of iterated block cipher constructions. We denote the total number of rounds by T. We use $F_t : \mathbb{F}_2^{2n} \rightarrow \mathbb{F}_2^{2n}$ to refer to round t, $0 \leq t < T$, which is defined as

$$F_t = \mathsf{AddConstants}_t \circ \mathsf{ShiftPlanes}_t \circ \mathsf{MixSlices} \circ \mathsf{SubRows}. \tag{3.34}$$

As such, PRØST-n is defined as

$$\text{PRØST-}n = F_{T-1} \circ \cdots \circ F_0. \tag{3.35}$$

We describe mapping between state representations in $\mathbb{F}_2^{4\times4\times d}$ and $\mathbb{F}_2^{2n}$, after describing the round function components in the following. The parameters for the two permutations, PRØST-128 and PRØST-256, are summarized in Table 3.3. We give the definition of each of the round operations below.

Table 3.3: PRØST permutations and their parameters

Permutation	State size $2n$	Register length d	Rounds T
PRØST-128	256	16	16
PRØST-256	512	32	18

For PRØST, the main rationale was to design a permutation that is efficient in many platforms, and to use a design approach that allows strong and easily verifiable security arguments. In particular, PRØST follows the widely acclaimed wide-trail design strategy of the AES, as already described in Section 3.1, however with a modification, as we shall see, since we use two different ShiftRows-like operations instead of one. This leads to significantly better bounds on the best differential- and linear trail probabilities, than otherwise possible. In particular as we shall see, not considering the lack of a secret key, PRØST can be considered as an AES-like cipher as specified in Section 3.1, using dimensions $M = 4$, $N \in \{16, 32\}$, word size $m = 4$ and $\rho = 2$.

Efficiency of the permutation results from mainly two efforts. First, each component has been optimized with respect to implementation cost. Second, the strong security of the design allows to keep the number of rounds low. In the following, we introduce all the components making up the PRØST round, as specified by Eq. (3.34), and motivate the design choice for each of them.

SubRows

To obtain non-linearity in the round function, PRØST uses a single 4-bit S-box $S : \mathbb{F}_2^4 \rightarrow \mathbb{F}_2^4$ which is given in Table 3.4. The SubRows operation substitutes each row of the state according to this S-box. More precisely, a state S will be mapped to a state S' where

Table 3.4: The 4-bit PRØST S-box in hexadecimal notation

X	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
$S(X)$	0	4	8	f	1	5	e	9	2	7	a	c	b	d	6	3

$$\forall X \in \mathbb{Z}_4, \forall Z \in \mathbb{Z}_d \quad : \quad S'_{X,0,Z}||S'_{X,1,Z}||S'_{X,2,Z}||S'_{X,3,Z} = S\big(S_{X,0,Z}||S_{X,1,Z}||S_{X,2,Z}||S_{X,3,Z}\big). \tag{3.36}$$

The S-box used for SubRows is very simple in terms of hardware- and software efficiency. If we let $X = X_3||X_2||X_1||X_0$ denote the input to S, then the new value of $X_3||X_2||X_1||X_0 = S(X)$ can be computed as

$$\begin{aligned} t_0 &= X_3, & t_1 &= X_2 \\ X_3 &= X_1 \oplus (t_0 \odot t_1), & X_2 &= X_0 \oplus (t_1 \odot X_1) \\ X_1 &= t_0 \oplus (X_3 \odot X_2), & X_0 &= t_1 \oplus (X_2 \odot X_1). \end{aligned} \tag{3.37}$$

Thus, computing $S(X)$ requires 10 simple instructions. The concrete choice of S-box is the result of a hardware assisted search though a significant subset of all possible S-boxes. Besides being very efficient in terms of cycle count, this S-box is also optimal with respect to linear and differential attacks. In particular, S was chosen among all S-boxes fulfilling the following criteria:

1. S is an involution, which prevents overhead of implementing two S-boxes (one for F_t and one for F_t^{-1}),
2. The maximal probability of a differential is 2^{-2},
3. There are exactly 24 differentials of probability 2^{-2},
4. The maximal absolute bias of a linear approximation is 2^{-2},
5. There are exactly 36 linear approximations of absolute bias 2^{-2}, and
6. Output bits have algebraic degrees of $2, 2, 3$ and 3, respectively.

Having only one single S-box within one plane allows to implement the S-box application using bit-slicing. To do this, one can for example keep all four lanes of a plane in four registers of d bits each. By applying the operations of Eq. (3.37) on a register-basis, the S-box is applied to the whole plane at once, using just ten instructions. Furthermore, keeping the S-boxes identical for all planes and all rounds reduces the code space and avoids additional overhead. The increased danger of symmetries throughout the cipher is countered by relatively heavy round constants, as we shall see below.

In Section 1.3, we briefly mentioned a class of attacks called side-channel attacks. One attack vector falling under this category is the *differential power attack* (DPA) invented by Kocher, Jaffe, and Jun [193], which essentially allows an attacker to determine secret values *during* the encryption process, by using several measurements of the power consumption of the encrypting device. One solution which is commonly employed to thwart DPAs is *masking*. The idea is essentially that all sensitive variables during encryption are protected by operating on each of them in conjunction with one or more randomly chosen masks. For example, if one has a sensitive variable X_0, one can mask it by operating instead on the variable $X = X_0 \oplus X_1 \oplus \cdots \oplus X_d$. The problem now consists of *modifying* the description of the cipher, such as to account for the added masking, to make sure that one obtains a functionally equivalent cipher. It turns out that for linear operations, this can be obtained cheaply and without much effort, while for the non-linear operations the cost is high in comparison. Much current research is going in the direction of masking schemes for non-linear operations in cipher implementations that are secure against differential power attacks. Some notable works include that of Coron and Goubin [99], Akkar and Giraud [12], Golic and Tymen [145], and Schramm and Paar [282]. Under the term *threshold implementations*, Nikova, Rechberger, and Rijmen introduce in [245] an approach to masking which, unlike earlier techniques requires, random values only at the initialization, whereas previous approaches need fresh randomness for each non-linear component. Their approach is also effective in the presence of *glitches*, a type of short-lived fault occurring in hardware components due to e.g. race conditions. While we do not go into the details here, a general rule of thumb is that the larger the S-box is, the more costly it is to implement countermeasures for DPA attacks. To that end, comparatively cheap countermeasure implementation for our 4-bit S-box is one of the motivations for choosing such a non-linear operation in PRØST.

MixSlices

The MixSlices operation updates the state by multiplying each of the d slices arranged in a 16-bit column vector, from the left, by a 16×16 matrix M over $\mathbb{F}_2$. As such, we obtain $S' = \mathsf{MixSlices}(S)$ as

$$\forall Z \in \mathbb{Z}_d \quad : \quad \begin{pmatrix} S'_{0,0,Z} \\ S'_{0,1,Z} \\ S'_{0,2,Z} \\ S'_{0,3,Z} \\ S'_{1,0,Z} \\ \vdots \\ S'_{3,2,Z} \\ S'_{3,3,Z} \end{pmatrix} = M \cdot \begin{pmatrix} S_{0,0,Z} \\ S_{0,1,Z} \\ S_{0,2,Z} \\ S_{0,3,Z} \\ S_{1,0,Z} \\ \vdots \\ S_{3,2,Z} \\ S_{3,3,Z} \end{pmatrix}. \tag{3.38}$$

The matrix M used for MixSlices is given by

$$M = \begin{pmatrix}
1&0&0&0&1&0&0&1&0&0&1&0&1&0&1&1 \\
0&1&0&0&1&0&0&0&0&0&0&1&1&0&0&1 \\
0&0&1&0&0&1&0&0&1&1&0&0&1&0&0&0 \\
0&0&0&1&0&0&1&0&0&1&1&0&0&1&0&0 \\
1&0&0&1&1&0&0&0&1&0&1&1&0&0&1&0 \\
1&0&0&0&0&1&0&0&1&0&0&1&0&0&0&1 \\
0&1&0&0&0&0&1&0&1&0&0&0&1&1&0&0 \\
0&0&1&0&0&0&0&1&0&1&0&0&0&1&1&0 \\
0&0&1&0&1&0&1&1&1&0&0&0&1&0&0&1 \\
0&0&0&1&1&0&0&1&0&1&0&0&1&0&0&0 \\
1&1&0&0&1&0&0&0&0&0&1&0&0&1&0&0 \\
0&1&1&0&0&1&0&0&0&0&0&1&0&0&1&0 \\
1&0&1&1&0&0&1&0&1&0&0&1&1&0&0&0 \\
1&0&0&1&0&0&0&1&1&0&0&0&0&1&0&0 \\
1&0&0&0&1&1&0&0&0&1&0&0&0&0&1&0 \\
0&1&0&0&0&1&1&0&0&0&1&0&0&0&0&1
\end{pmatrix}. \tag{3.39}$$

It is involutive, i.e. self-inverse, with Hamming weight $\mathsf{hw}(M) = 88$. The matrix is MDS, so the branch number is $B = 5$. From the perspective of differential- and linear cryptanalysis, this means that if $k > 0$ is the number of active rows in a slice, then the corresponding slice of S' has at least $5 - k$ active rows. For the MixSlices operation, there are three major requirements:

1. The linear- and differential branch number should be $B = 5$,
2. The M matrix should have low density, and
3. As a heuristic to minimize implementation characteristics, for both encryption and decryption, the MixSlices operation should be an involution.

We elaborate on the first two requirements below.

High Branch Number. The main design goal of the MixSlices transformation is to follow the wide-trail design strategy. Hence, MixSlices is related to an $\mathbb{F}_2$-linear error-correcting code over $\mathbb{F}_2^4$ with minimum distance $B = 5$. Note that in our setting, the linear- and differential branch numbers are identical. In other words, a difference in $k > 0$ rows of a slice will result in a difference in at least $5 - k$ rows in the same slice after one application of MixSlices. While this is a good bound for 2 rounds, only the interaction with the ShiftPlanes$_t$ operation guarantees an overall secure design.

Low Density. The density, i.e. the Hamming weight of the M matrix, roughly corresponds to the number of XOR operations that have to be performed when implementing MixSlices. It is therefore a suitable metric when optimizing performance, both in software and hardware. Among all 16×16 binary matrices, we searched through the involutive ones with branch number $B = 5$ and with a particularly low Hamming weight. The best solution we were able to find with our hardware assisted search had a Hamming weight of $\mathsf{hw}(M) = 88$. Note that we cannot guarantee that our matrix is actually optimal, as the minimal number of ones in such a matrix is generally unknown.

ShiftPlanes$_t$

The ShiftPlanes$_t$ operation rotates the registers of each plane of the state towards right (positive Z direction) by a particular amount. In contrast to many existing designs, PRØST uses two different sets of rotation constants, depending on the parity of the round number t; one for even numbered rounds and one for odd numbered rounds. Again, this means in the notation of the analysis of rotation matrices in AES-like ciphers of Section 3.1, that we use $\rho = 2$.

For round t with $0 \le t < T$, the rotation constant for plane j with $0 \le j \le 3$ is given by the j^{th} column of row t mod 2, denoted $\sigma_{t \bmod 2, j}$, of a 2×4 rotation matrix σ over $\mathbb{Z}_d$. We use σ_0 and σ_1 to denote the first, respectively second row of σ. Thus, ShiftPlanes$_t$ applied to a state S in round t will result in a state S' where

$$\forall X \in \mathbb{Z}_4, \forall Y \in \mathbb{Z}_4 \quad : \quad S'_{X,Y} = \begin{cases} S_{X,Y} \ggg \sigma_{0,X} & , t \text{ even} \\ S_{X,Y} \ggg \sigma_{1,X} & , t \text{ odd.} \end{cases} \tag{3.40}$$

Table 3.5: Rotation matrices σ for the $\mathsf{ShiftPlanes}_t$ operation for $d = 16$ and $d = 32$

$d = 16$	$d = 32$
$\begin{pmatrix} 0 & 2 & 4 & 6 \\ 0 & 1 & 8 & 9 \end{pmatrix}$	$\begin{pmatrix} 0 & 4 & 12 & 26 \\ 1 & 24 & 26 & 31 \end{pmatrix}$

The particular rotation matrices are given in Table 3.5.

We posed two main design criteria for the $\mathsf{ShiftPlanes}_t$ transformation. Firstly, the rotation constants should contribute to full diffusion after as few rounds as possible. Secondly, the differential- and linear trails with fewest active S-boxes over a given number of rounds, should have as many active S-boxes as possible, just as it was the case for our analysis in Section 3.1. We remark that in fact the work on the choice of rotation constants for PRØST is what eventually resulted in the work on the more generalized AES-like ciphers of Section 3.1.

The σ matrices for PRØST were found to be optimal in the sense that they give the best diffusion, number of active S-boxes (see also Section 3.2.3) and implementation cost, for the specified register lengths d. For register length $d = 16$ we obtain full diffusion after 3 rounds and for $d = 32$ after 4 rounds. For lower bounds on the number of active S-boxes over various number of rounds, we refer to Section 3.2.3.

With respect to implementation cost, we have optimized to have as many multiples of 8 as possible among the rotation constants, as these can be implemented at no cost on 8-bit platforms and cheaply on larger platforms. For the constants that are not multiples of 8, we seek minimize their sum, as the implementation cost is proportional to the total sum of the constants. This sum, not counting multiples of 8, is 22 for the $d = 16$ case and 88 for the $d = 32$ case, which roughly translates to the implementation cost in cycles.

$\mathsf{AddConstants}_t$

In the $\mathsf{AddConstants}_t$ operation, different round- and lane-dependent constants are XORed to each lane. Let $c_0 = \mathsf{msb}_d(\texttt{75817b9d})$ and $c_1 = \mathsf{msb}_d(\texttt{b2c5fef0})$. In the following, let $j = 4X + Y$, with $X, Y \in \mathbb{Z}_4$, so j is the lane index with $0 \leq j \leq 15$. In round t, $0 \leq t < T$, we have $S' = \mathsf{AddConstants}_t(S)$ with

$$\forall X \in \mathbb{Z}_4, \forall Y \in \mathbb{Z}_4 \quad : \quad S'_{X,Y} = \begin{cases} S_{X,Y} \oplus (c_0 \lll (t + j)) & , j \text{ even} \\ S_{X,Y} \oplus (c_1 \lll (t + j)) & , j \text{ odd.} \end{cases} \tag{3.41}$$

The purpose of adding round constants is to make each round different. If the rounds are all the same, denoted F, then fixed points $X \in \mathbb{F}_2^{2n}$ such that $F(X) = X$ would extend to the entire permutation. For example, if $P = F^{10}$, then fixed points for F^2 and F^5 would also extend to P. Therefore, one can expect several fixed points for P, whereas for an ideal (i.e. random) permutation only a single fixed point is expected. By choosing round-dependent constants for $\mathsf{AddConstants}_t$, we expect the number of fixed points to be close to 1. The two constants `75817b9d` and `b2c5fef0` are generated from the first 64 digits after the decimal points of π, as illustrated by the code in Listing 3.1.

Listing 3.1: Code for generating constants c_0 and c_1

```
#include <stdint.h>
#include <stdio.h>
const uint32_t pi[64] = {1,4,1,5,9,2,6,5,3,5,8,9,7,9,3,2,
                         3,8,4,6,2,6,4,3,3,8,3,2,7,9,5,0,
                         2,8,8,4,1,9,7,1,6,9,3,9,9,3,7,5,
                         1,0,5,8,2,0,9,7,4,9,4,4,5,9,2,3};
int main(void) {
        uint32_t c0 = 0;
        uint32_t c1 = 0;
        int i;
        for(i = 0; i < 32; ++i) {
                c0 |= (pi[i] & 1) << i;
                c1 |= (pi[i+32] & 1) << i;
        }
        printf("c0 = 0x%08x\n", c1);
        printf("c1 = 0x%08x\n", c2);
        return 0;
}
```

Mapping Byte Array to PRØST State

The PRØST state is a 4×4 matrix of d-bit registers. Inputs to an authenticated cipher, as specified by the requirements of the CAESAR competition, are byte arrays. We now describe the mapping of a byte array of $2n$ bits to a PRØST state, and back.

On a high level, we map words of length $\ell = d/8$ bytes to matrix coefficients. This is done row first, column second. As such, the first ℓ bytes go into $S_{0,0}$, the next ℓ bytes go to $S_{0,1}$, and so on. On a lower level, we map arrays of ℓ bytes to ℓ-byte words. For this mapping we use little-endian representation, since this is natively supported by the large majority of modern computer architectures. For example, when we have $d = 32$, a register is $\ell = 4$ bytes long and the first four bytes of the byte array bs are mapped to register $S_{0,0}$ as

$$S_{0,0} = \texttt{bs}[3] \,\|\, \texttt{bs}[2] \,\|\, \texttt{bs}[1] \,\|\, \texttt{bs}[0], \tag{3.42}$$

or, considering $S_{0,0}$ as an integer, as

$$S_{0,0} = \sum_{i=0}^{3} \texttt{bs}[i] \cdot 2^{8i}. \tag{3.43}$$

When putting this together we obtain that register $S_{X,Y}$ of the PRØST state S with $0 \leq X, Y \leq 3$ is loaded from a byte array bs as

$$S_{X,Y} = \texttt{bs}[4\ell X + \ell Y + \ell - 1] \,\|\, \texttt{bs}[4\ell X + \ell Y + \ell - 2] \,\|\, \cdots \,\|\, \texttt{bs}[4\ell X + \ell Y], \tag{3.44}$$

or, again considering $S_{X,Y}$ as an integer, as

$$S_{X,Y} = \sum_{i=0}^{\ell-1} \texttt{bs}[4\ell X + \ell Y + i] \cdot 2^{8i}. \tag{3.45}$$

The mapping of a state to a byte stream is done by the obvious inversion of this mapping.

Let $g : \mathbb{F}_2^{4\times 4\times d} \to \mathbb{F}_2^{2n}$ be defined for a state S as

$$g(S) = S_{0,0} \| S_{0,1} \| \cdots \| S_{3,2} \| S_{3,3}. \tag{3.46}$$

When a PRØST state is XORed with a $2n$-bit value X, we implicitly mean that X is XORed to the state mapped to $\mathbb{F}_2^{2n}$ by g, i.e. $X \oplus g(S)$. Note that for all $S_{X,Y}$, the bits $S_{X,Y,0}$ and $S_{X,Y,d-1}$ are the most- respectively least significant bits of $S_{X,Y}$.

3.2.3 Cryptanalysis of Permutation

In the following, we present our analysis of the PRØST permutation itself, with respect to some of the most important aspects, seen from both a designers and attackers point of view. In particular, we discuss the following aspects of the PRØST permutation: diffusion, strict avalanche criterion, the avalanche effect, and its resistance towards differential- and linear cryptanalysis.

Diffusion

For our analysis, we assume that for the 4-bit S-box used, each of the 4 output bits depend on each of the 4 input bits. Also, we assume that the MixSlices operation mixes the bits in a slice, such that each output *row* depends on at least one bit from *each* input row. We remark that the assumption on the S-box is not true for the PRØST S-box: letting $X' = S(X)$, then X'_3 does not depend on X_0 and X'_2 does not depend on X_3. While this will, to some extent, mean that our analysis in the following is based on a too strong assumption, we deem that full diffusion will be obtained after at most one more round than found.

After applying SubRows of the first round, all bits within a row are interdependent by assumption. Applying MixSlices of the first round implies that we get bit interdependency between all bits in the same slice. The $\mathsf{ShiftPlanes}_t$ operation cyclically shifts the planes by a round-dependent amount, effectively mixing the slices. As SubRows and MixSlices take care of mixing bits in slices in each round, the question becomes how many rounds are required to make the plane shifting of $\mathsf{ShiftPlanes}_t$ create bit-interdependency between all planes.

When determining the choice of rotation matrix σ, we conducted an experiment trying out all combinations of $\sigma \in \mathbb{Z}_d^{2\times 4}$, investigating the required number of rounds to obtain full diffusion. For $d = 16$ we found that the best σ requires a minimum of 3 rounds for full diffusion, and there are 4096 such matrices. For $d = 32$ a minimum of 4 rounds are required for full diffusion, and there are 729088 rotation matrices σ obtaining this minimum. Unfortunately, there is no matrix which obtains the best diffusion for both $d = 16$ and $d = 32$, hence the two parameter matrices in Table 3.5, one for each case of d.

We remark that the method used here to determine the number of rounds required to obtain full diffusion is essentially the same approach as the one using sumsets, as described in Section 3.1.6.

Strict Avalanche Criterion and Avalanche Effect

For the purpose of investigating the *strict avalanche criterion* (SAC) and the *avalanche effect* in the PRØST permutation, we conduct randomized experiments to measure the degree to which these two properties are obtained. For PRØST-n, the statement of the avalanche effect is given by

$$\forall i \in \mathbb{Z}_{2n} \quad : \quad 2^{-2n} \sum_{X \in \mathbb{F}_2^{2n}} \mathrm{hw}\Big(\text{PRØST-}n(X) \oplus \text{PRØST-}n\big(X \oplus \mathbf{e}_i\big)\Big) = n, \tag{3.47}$$

while the statement of the SAC is given by

$$\forall i \in \mathbb{Z}_{2n} \quad : \quad \Pr_j\Big[\text{PRØST-}n(X)_j \neq \text{PRØST-}n\big(X \oplus \mathbf{e}_i\big)_j\Big] = \frac{1}{2}, \tag{3.48}$$

where the probability is taken over the index $j \in \mathbb{Z}_{2n}$. To experimentally measure the extent to which PRØST meets these two criteria, we sample a random subset $\mathscr{X} \subset \mathbb{F}_2^{2n}$. For $0 \leq i, j < 2n$, define

$$a_{i,j} = \sharp\Big\{X \in \mathscr{X} \mid \text{PRØST-}n\big(X \oplus \mathbf{e}_i\big)_j \neq \text{PRØST-}n(X)_j\Big\} \quad \text{and} \tag{3.49}$$

$$b_{i,j} = \sharp\Big\{X \in \mathscr{X} \mid \mathrm{hw}\big(\text{PRØST-}n\big(X \oplus \mathbf{e}_i\big) \oplus \text{PRØST-}n(X)\big) = j\Big\}. \tag{3.50}$$

We now define the degree of avalanche effect $\deg_{ava}$ and degree of SAC $\deg_{SAC}$ as defined by Serf in his investigation of the AES finalists [283],

$$\deg_{ava} = 1 - \frac{\sum_{i=0}^{2n-1} \left|\sharp\mathscr{X}^{-1} \cdot \sum_{j=0}^{2n-1} 2jb_{i,j} - 2n\right|}{4n^2} \quad \text{and} \tag{3.51}$$

$$\deg_{SAC} = 1 - \frac{\sum_{i=0}^{2n-1} \sum_{j=0}^{2n-1} \left|2a_{i,j} \cdot \sharp\mathscr{X}^{-1} - 1\right|}{4n^2}. \tag{3.52}$$

Our results show that we obtain SAC and avalanche effect degrees of $\deg_{ava} = \deg_{SAC} = 1.0$ (which is ideal) after 2 to 3 rounds for $d = 16$ and 3 to 4 rounds for $d = 32$.

Bounds on Active S-boxes

As we saw in Section 3.1, the minimum number of active S-boxes for any trail over T rounds of a primitive, in combination with the differential and/or linear properties of the S-box used, can be used to derive upper bounds on the differential- and linear trail probabilities. As the work on PRØST preceded the work of Section 3.1, we present in the following our analysis independently of the more general analysis on the AES-like ciphers above. We also remark that the rotation matrices providing the improvements for PRØST described in Table 3.2 of Section 3.1, have not been adopted into the PRØST specification.

To lower bound the number of active S-boxes for $d = 16$ and $d = 32$, we model the propagation of active S-boxes over a particular number of rounds as an integer programming problem, just as it was done with Model 1 for the generalized AES-like ciphers. The particular choices of $\sigma \in \mathbb{Z}_d^{2\times 4}$ used in PRØST arise from solving this problem for randomly chosen subsets of the rotation matrices σ giving optimal diffusion for $d = 16$ and $d = 32$ (see above), and choosing those giving the best bounds. The findings for the minimal number of active S-boxes using the σ matrices from Table 3.5, for various number of rounds, are given in Table 3.6.

Table 3.6: Lower bounds on the number of active S-boxes for $d \in \{16, 32\}$ for various number of rounds. The number in parenthesis was the best obtained when the solver stopped due to memory limitations.

	Rounds T				
d	4	5	6	7	8
16	25	41	85	96	105
32	25	41	105	169	(210)

3.2.4 PRØST-based CAESAR Proposals

In Section 1.2.4 we mentioned that symmetric primitives based on permutations are becoming increasingly popular, most notably in sponge constructions used for AEAD schemes. We saw APE as an example of such, and other examples include the pre-CAESAR SpongeWrap by Bertoni, Daemen, Peeters, and Assche [52] and PPAE by Khovratovich [183]. In the CAESAR competition, an abundance of other permutation-based designs have emerged. In particular, an exhaustive list (besides PRØST) is: Artemia [17], Ascon [116], CBEAM [276], ICEPOLE [235], Ketje [53], Keyak [54], Minalpher [280], NORX [34], π-Cipher [144], PRIMATEs [23], and STRIBOB [277].

In this section we describe our first-round proposals for the CAESAR competition, which all are specific instantiations of AEAD schemes using the PRØST permutation as the underlying primitive. Each of the instantiations are based on existing AE schemes. The proposals are summarized in Table 3.7, which lists the specific instantiations, while the specifications in the following use the general permutation size $2n$ for their descriptions.

Table 3.7: The PRØST CAESAR proposals, underlying PRØST permutation and their rankings. The columns indicate the properties of the proposals: whether the scheme is online in encryption and decryption ($\mathscr{E}$ and $\mathscr{D}$); parallelizability (P); nonce misuse resistance (NMR); easy constant-time implementation (CT); and cheap countermeasures for against power analysis (CM).

			Online					
Rank	Proposal	Permutation	$\mathscr{E}$	$\mathscr{D}$	P	NMR	CT	CM
1	PRØST-COPA-128	PRØST-128	✓	✓	✓	OAE1	✓	✓
2	PRØST-COPA-256	PRØST-256	✓	✓	✓	OAE1	✓	✓
3	PRØST-OTR-128	PRØST-128	✓	✓	✓	NAE	✓	✓
4	PRØST-OTR-256	PRØST-256	✓	✓	✓	NAE	✓	✓
5	PRØST-APE-256[256, 256]	PRØST-256	✓			OAE1	✓	✓
6	PRØST-APE-128[128, 128]	PRØST-128	✓			OAE1	✓	✓

Throughout our description of the proposals, we use P_n as a compact notation for the PRØST-n permutation. Multiplications are to be understood as working in the finite field $\mathbb{F}_{2^{2n}} = \mathbb{F}_2[X]/f_{2n}(X)$ defined modulo an irreducible polynomial $f_{2n}(X)$ over $\mathbb{F}_2$ of degree $2n$. The irreducible polynomials used for our proposals are given in Table 3.8. In the finite fields used, multiplication by the constant 2 (shown in Algorithm 9) is particularly simple, as it involves only a left shift by one position and a conditional XOR of the reduction polynomial, as we saw was the case when describing the multiplication by 02 in the MixColumns operation of the AES.

Table 3.8: Defining polynomials for finite fields used in PRØST

Field	Defining polynomial $f_{2n}(X)$	Hex
$\mathbb{F}_{2^{256}}$	$X^{256}+X^{10}+X^5+X^2+1$	425
$\mathbb{F}_{2^{512}}$	$X^{512}+X^8+X^5+X^2+1$	125

```
Algorithm 9: XTIME (multiplication by 2 in $\mathbb{F}_{2^{2n}}$)
    Data: $X \in \mathbb{F}_{2^{2n}}$
    Result: $2X$
1   $f_{256} \leftarrow 425$
2   $f_{512} \leftarrow 125$
3   if $\mathsf{msb}_1(X) = 1$ then
4       return $(X \ll 1) \oplus f_{2n}$
5   else
6       return $X \ll 1$
7   end
```

We remind from Section 1.2.4 that κ denotes the key length; η denotes the nonce length; and τ denotes the tag length. None of our proposals use secret message numbers. Table 3.9 gives the parameters for each of the proposals. We use k to denote the number of blocks of associated data and ℓ to denote the number of message blocks (after padding) throughout. We remark that the block size of each M_i, C_i and A_i are not always equal: for PRØST-COPA and PRØST-OTR, the block size is $2n$ while for PRØST-APE the block size equals the rate r (see below). For a message M, we write M_i, $1 \leq i \leq \ell$, to denote the i^{th} block of M, starting from the most significant end. We use an equivalent indexing for a ciphertext C and associated data A.

Table 3.9: PRØST proposal parameters

Rank	Proposal	Block size	κ	η	τ
1	PRØST-COPA-128	256	256	256	256
2	PRØST-COPA-256	512	512	512	256
3	PRØST-OTR-128	256	256	128	128
4	PRØST-OTR-256	512	512	256	256
5	PRØST-APE-256[256, 256]	256	256	256	256
6	PRØST-APE-128[128, 128]	128	128	128	128

Ciphertext Expansion

With many other AEAD schemes, including many CAESAR proposals, we see that the schemes look very simple at a first glance. However, when it comes to handling *fractional data*, i.e. when either associated data A or message M have a length not fitting with the underlying primitive, many proposals turn to overly complicated solutions. An example is COPA which uses the XLS construction by Ristenpart and Rogaway [265]. However, as pointed out by Nandi in [241, 242] the use of XLS in COPA allows for a distinguisher.

Motivated by avoiding pitfalls associated with special cases, and ease of implementation, we choose for *all* the PRØST proposals to *always* do ciphertext expansion by applying padding to the message (and associated data, if necessary), as the first thing before processing. In particular, letting $X \in \mathbb{F}_2^*$ be a binary string, we use the padding function

$$\begin{aligned} \mathsf{pad}_b : \mathbb{F}_2^* &\to \mathbb{F}_2^{mb}, \quad m, b \in \mathbb{N} \\ X &\mapsto X \| 10^{b-(|X| \bmod b)-1}. \end{aligned} \tag{3.53}$$

Thus, the padding appends a 1-bit followed by as many zeroes required to make $|\mathsf{pad}_b(X)|$ the smallest multiple of b which is strictly greater than $|X|$. We remark that this padding function is not identical to that introduced in Section 1.2.4, which mapped inputs of length at most $r-1$ bits, to outputs of length exactly r bits. This change is made, because the general duplex construction allows blocks of varying lengths, while for our proposals, blocks will always have identical lengths, so our padding works on the full string at once, rather than on a block basis. The reasoning behind our choice to always employ ciphertext expansion is that:

- This approach is less prone to implementation errors, which are a frequent source of real-world security break-downs. By effectively eliminating the special cases an implementation has to be able to handle, chances of introducing implementation errors are diminished,
- Proposal implementations become easier to optimize in both software and hardware, which in turn aids comparisons, and
- The impact on size of implementation can be significant. Be it code size in software or control logic in hardware, the lower complexity of padding rules, compared to handling special cases, leads to smaller implementations.

By allowing a minor ciphertext expansion, considering relevant packet sizes (we return to this in Chapter 4), we achieve the advantages mentioned above, except in some corner use-cases. We believe this is the right trade-off for most practical applications, and indeed will facilitate ease of use. We remark that with ciphertext expansion, ciphertexts always have a length equal to a multiple of the block size.

Block Cipher-based AEAD to Permutation-based AEAD

It is a well-known result that any permutation can be turned into a block cipher by plugging it into what is called an *Even-Mansour construction* [122] named after the authors. Letting $K = (K_0, K_1)$, the construction uses a permutation P to define a block cipher as

$$\mathcal{E}_K(X) = P(X \oplus K_0) \oplus K_1. \tag{3.54}$$

When $K_0 = K_1$, the construction is called a *Single-Key Even-Mansour* construction (SEM) and is due to Dunkelman, Keller, and Shamir [119]. As such, any block cipher-based AE scheme can be turned into a permutation-based scheme by using this construction. For the purpose of using PRØST-n in block cipher-based AEAD schemes, we use it in an SEM construction with a single, fixed key K. As such, we define

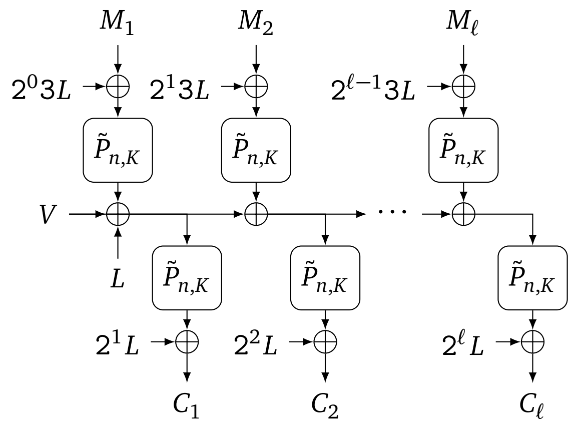
$$\begin{aligned}\tilde{P}_{n,K} : \mathbb{F}_2^{2n} \times \mathbb{F}_2^{2n} &\to \mathbb{F}_2^{2n} \\ (K, X) &\mapsto P_n(X \oplus K) \oplus K.\end{aligned} \tag{3.55}$$

With this, we next define specific instantiations of authenticated encryption schemes based on PRØST, both when the underlying scheme is block cipher- or permutation-based, either by using it directly, or through $\tilde{P}_{n,K}$.

PRØST-COPA-n

In this part, we give our specification of PRØST-COPA-n, the instantiation of the COPA AEAD scheme using $\tilde{P}_{n,K}$ as the underlying block cipher. For PRØST-COPA-n the block size is $2n$ bits. The resulting proposal PRØST-COPA-n offers OAE1 security, leaking block-aligned prefixes of messages under nonce misuse. PRØST-COPA-n is a fully parallelizable, online AEAD scheme, and uses the very efficient doubling of Algorithm 9, as opposed to general multiplication in $\mathbb{F}_{2^{2n}}$, for the tweak values. The scheme uses two calls to PRØST-n and two doublings (i.e. multiplications by 2) in $\mathbb{F}_{2^{2n}}$, per message block.

Figure 3.5: Encryption of ℓ message blocks with PRØST-COPA-n

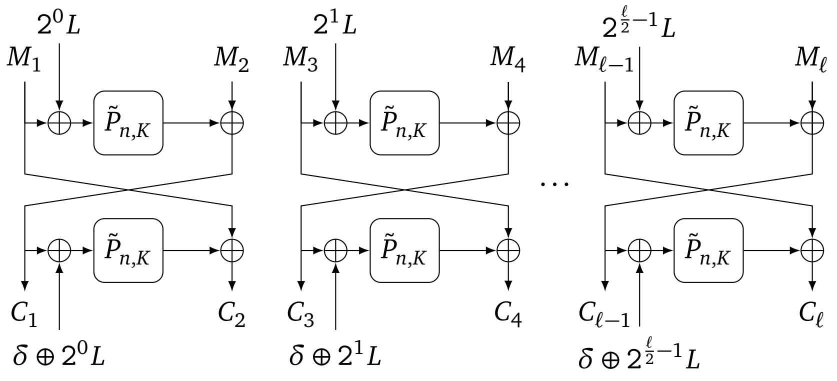


The encryption of ℓ message blocks is depicted in Figure 3.5 (processing associated data is not shown). In the figure, the values V and L are output from the processing of associated data and nonce. The complete description of encryption with authentication, decryption with verification and processing of associated data for PRØST-COPA-n is given by Algorithms 14 through 16 in Appendix D.

PRØST-OTR-*n*

Next, we specify PRØST-OTR-n, which is the instantiation of OTR, as introduced in Section 1.2.4, using $\tilde{P}_{n,K}$ as the underlying block cipher. For PRØST-OTR-n the block size is $2n$ bits. As we have already seen, OTR uses 2-round Feistel constructions to encrypt two consecutive message blocks. It uses one call to PRØST-n and half a doubling in $\mathbb{F}_{2^{2n}}$ per message block.

Figure 3.6: Encryption of ℓ message blocks with PRØST-OTR-n, when ℓ is even



PRØST-OTR-n inherits the features of OTR. In particular, it offers NAE security, so all security claims are lost under nonce misuse. Comparing to schemes which offer e.g. OAE1 or even MRAE security, the gain is performance. The proposal is online and completely parallelizable. It does not require the inverse of the underlying primitive, due to the employment of the Feistel structure.

The encryption of ℓ message blocks, for even ℓ, with PRØST-OTR-n is depicted in Figure 3.6. In the figure, the value δ is the encryption of the padded nonce, i.e. $\delta = \tilde{P}_{n,K}(\mathsf{pad}_{2n}(N))$. The complete description of encryption with authentication, decryption with verification and processing of associated data for PRØST-OTR-n is given by Algorithms 17 through 21 in Appendix D.

PRØST-APE-*n*$[r,c]$

Besides the two block cipher-based AEAD schemes used in conjunction with the PRØST permutation, via the SEM construciton, we also propose to use PRØST in the APE mode of operation. As we saw in Section 1.2.4, APE is a permutation-based AEAD scheme using the duplexed sponge construction. As such, we have two parameters: the rate r and the capacity c. These are what are indicated by the $[r,c]$ part of the name PRØST-APE-$n[r,c]$. Thus, we have $r + c = 2n$. For PRØST-APE-$n[r,c]$ the block size is r bits.

PRØST-APE-$n[r,c]$ is online in encryption, i.e. one does not need to know all message blocks, and in particular the number of message blocks, before one can start encrypting. For decryption, however, one starts decrypting the last ciphertext block first, and hence decryption is *not* online. The fact that one decrypts in reverse implies the need for the inverse of the underlying permutation as well.

Figure 3.7 illustrates the encryption and tag generation for ℓ message blocks of r bits each. In the figure, the value $IV = IV_r \| IV_c$ is the result of processing associated data and nonce, which is not shown. The nonce N is effectively considered part of the associated data, and is prepended

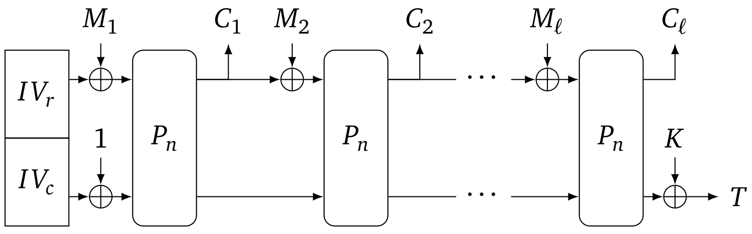


Figure 3.7: Encryption and authentication of ℓ message blocks of r bits each, using PRØST-APE-$n[r,c]$

to this. The complete description of encryption with authentication, decryption with verification and processing of associated data for PRØST-APE-$n[r,c]$ is given by Algorithms 22 and 23 in Appendix D.

As we shall see, APE achieves privacy and integrity up to the bound $2^{c/2}$ in the ideal permutation model. As such, the choice of rate and capacity influence performance and security, in the sense that increasing capacity and decreasing rate will increase security and decrease performance, and vice versa. We refer to Table 3.7 for the specific parameters of our proposals. Under nonce misuse, PRØST-APE offers OAE1 security, just like PRØST-COPA.

3.2.5 Features

As already mentioned, the PRØST permutation is designed to be highly secure and with good implementation characteristics. These properties are inherited by the proposals, as they directly apply the permutation in a mode of operation. All the proposals suggested have the following features in common:

- They are based on a large cryptographically secure permutation. This is arguably one of the simplest, if not the simplest way, to build primitives for symmetric cryptography, including authenticated encryption primitives. In particular, this avoids complicated and often very inelegant considerations related to key schedules for traditional block cipher based constructions, and

- They are based on a *single* permutation. Having a single permutation further avoids considerations related to the independence of permutations.

3.2.6 Security Goals and Proofs

Next, we introduce some crucial concepts of security notions, and we use these to give proofs of security for the proposed schemes. The proofs largely rely on the existing proofs for the underlying modes of operation used. When considering the provable security of an AEAD scheme, we consider two requirements:

1. The ciphertexts C produced by the AEAD scheme should be indistinguishable from the output provided by an ideal primitive with the same functional signature. This security notion is denoted IND-CPA (for indistinguishability under a CPA attack). The IND-CPA

notion is what guarantees that the primitive provides confidentiality. In our notions and proofs below, we refer to this property as either IND-CPA or *privacy* (in particular, either priv or perm-priv for short).

2. It should not be possible for an adversary to be able to provide a valid ciphertext/tag pair (C, T) without knowledge of the secret key K. In other words, the adversary must not be able to forge. This security notion is denoted INT-CTXT (for integrity of ciphertext). The notion of INT-CTXT is what guarantees that the primitive provides authenticity and integrity. In our notions and proofs below, we refer to this property as either INT-CTXT or *authenticity* (in particular, either auth or perm-auth for short).

We consider an AEAD scheme to be secure *if and only if* it satisfies both the IND-CPA and INT-CTXT properties. Below in Definitions 40 through 43, we define the two notions formally, both for block cipher- and permutation-based AEAD, and we put them into the context of our PRØST proposals. Table 3.10 summarizes our security claims for our six proposals with respect to privacy (the IND-CPA notion) and authenticity (the INT-CTXT notion). Note that for block cipher-based proposals, the claimed security level is n bits for a proposal using PRØST-n, while for the permutation-based PRØST-APE, the security level is $c/2$. We prove these bounds below.

Table 3.10: Proposal ranks and security claims (in bits) for our proposals for privacy and authenticity

Rank	Proposal	Privacy	Authenticity
1	PRØST-COPA-128	128	128
2	PRØST-COPA-256	256	256
3	PRØST-OTR-128	128	128
4	PRØST-OTR-256	256	256
5	PRØST-APE-256[256, 256]	128	128
6	PRØST-APE-128[128, 128]	64	64

Standard- and Ideal Model

When proving the security for a mode of operation, one proceeds under a set of assumptions, usually captured by one of two possible models: the *standard model* or the *ideal model*.

With a proof in the standard model, the power of the adversary is only limited by the time and computational power available. Nothing is assumed about the underlying primitive; rather, the proof evolves around a security reduction, relating the security of the mode to the security of the underlying primitive. In the standard model, the adversary is given access to the mode itself, but can also compute as many primitive outputs as she wants, given the resource restrictions. The outcome of a proof in this model is usually a security bound which is directly related to that of the primitive, plus some additional gain in advantage for the adversary in distinguishing the mode, due to the way the primitive is employed.

Proofs in the ideal model allow the adversary to interact with both an instantiation of the mode itself on one hand, and of the underlying primitive on the other hand, so as to imitate the fact that in the standard model she can compute primitive outputs on her own. The primitive itself is modeled as an ideal (i.e. random) permutation. In particular, one typically models the primitive using *lazy sampling*, where we imagine the adversary asking queries to a black box. When the box is queried with an input not previously seen, it randomly picks an output that *was not chosen before*, while on previously seen inputs it deterministically returns the corresponding output. An important difference between proofs in the standard- and ideal models is, that in the latter there is no security reduction. As such, one obtains a tangible bound, but which in turn relies on the assumption of ideality of the primitive. In reality of course, the primitive used is not ideal, and as such one generally prefers a proof in the standard model. However, there is no known construction proven secure in the ideal model, which has been shown insecure when instantiated (unless specifically constructed to prove such a point). We remark that in both cases, concepts such as side-channel attacks and the possibilities of errors are completely abstracted away from the model.

The sketch of our approach to the security proofs will be as follows. For all our proofs, we assume PRØST to be an ideal permutation, so we work in the ideal model. Two of the suggested modes of operation for PRØST are block cipher-based (COPA and OTR), while the last (APE) is permutation-based. To that end, the security proofs for PRØST-APE are obtained via the security proofs for the APE mode of operation, as they assume *any* ideal permutation. For COPA and OTR the reference proofs are in the standard model, and as such involves a term $\mathbf{Adv}_{\mathscr{E}}^{\mathsf{sprp}}$ which denotes the security of the underlying primitive (we define this notion below in Definition 39). We prove in the ideal model a bound for $\mathbf{Adv}_{\mathscr{E}}^{\mathsf{sprp}}$ in the case where $\mathscr{E} = \tilde{P}_{n,K}$, and plug this into the term for both the PRØST-COPA and PRØST-OTR security proofs, to obtain proofs in the ideal model. The author would like to thank Bart Mennink for his tireless help in understanding the following notions and their relation to the security proofs for the PRØST proposals.

Security Definitions

In the following, we give the canonical definitions of security that allow us to show the security of our proposals with respect to privacy and authenticity.

Definition 37 (Distinguisher)**.** *Let k be a positive integer. We define a* distinguisher $\mathbb{D}$ *to be an algorithm which is given access to a list of k oracles $\mathcal{O} = \mathcal{O}_1, \ldots, \mathcal{O}_k$ which together represent one of two systems. By $\mathbb{D}^{\mathcal{O}_1,\ldots,\mathcal{O}_k}$ we denote that $\mathbb{D}$ has access to the particular list of oracles. First, a fair coin is flipped to obtain a value $b \in \{0,1\}$. If $b = 1$, the oracles represent on one hand the behavior of an instantiation of a concrete cryptographic scheme (called the* real world*), while if $b = 0$, they represent an idealized version of the scheme on the other hand (called the* ideal world*). The goal of the distinguisher $\mathbb{D}$ is to guess which is the case. We write (without loss of generality) that $\mathbb{D}^{\mathcal{O}_1,\ldots,\mathcal{O}_k} = 1$ if the distinguisher guesses that $b = 1$, and $\mathbb{D}^{\mathcal{O}_1,\ldots,\mathcal{O}_k} = 0$ otherwise.*

For extra notation, when used as oracles, we let $\bowtie$ denote an oracle which is an ideal version of the AEAD encryption function, i.e. it returns a randomly sampled ciphertext/tag pair on each call. Also, when the symbol $\doteq$ is used as an oracle, we mean an oracle that *always* returns the decryption failure symbol $\perp$. For the security of PRØST-COPA, we will need the notion of an *online cipher*. The notation is re-used from the security proof of COPA [21].

Definition 38 (Online cipher). *A cipher $\mathcal{E} : \mathbb{F}_2^\kappa \times \left(\mathbb{F}_2^{2n}\right)^+ \to \left(\mathbb{F}_2^{2n}\right)^+$ is said to be* online *if it has the properties that*

1. *It is a permutation on every block of* $2n$ *bits, and*
2. *The output blocks are identical for two different inputs with a common prefix, i.e. $\mathcal{E}_K(X\|Y)$ and $\mathcal{E}_K(X\|Y')$ are identical on the first $|X|$ bits for any $X, Y, Y' \in \left(\mathbb{F}_2^{2n}\right)^+$.*

As such, an online cipher with a fixed key is a permutation on the blocks starting from block i and onwards, and is determined by the first $i-1$ blocks. We let $\mathsf{OPerm}(2n)$ *denote the set of all such online permutations $\pi : \left(\mathbb{F}_2^{2n}\right)^+ \to \left(\mathbb{F}_2^{2n}\right)^+$. For APE, we use a slightly different online permutation, see [24].*

Definition 39 (Advantage compared to strong pseudo-random permutation). *Let $\mathcal{E} : \mathbb{F}_2^\kappa \times \mathbb{F}_2^m \to \mathbb{F}_2^m$ be a block cipher. The* sprp advantage *(short for* strong pseudo-random permutation*) over $\mathcal{E}$ of a distinguisher $\mathbb{D}$ is defined as*

$$\mathbf{Adv}_{\mathcal{E}}^{\mathsf{sprp}}(\mathbb{D}) = \left|\Pr_K\left[\mathbb{D}^{\mathcal{E}_K,\mathcal{E}_K^{-1}} = 1\right] - \Pr_\pi\left[\mathbb{D}^{\pi,\pi^{-1}} = 1\right]\right|. \tag{3.56}$$

The probabilities are taken over $K \xleftarrow{\$} \mathbb{F}_2^\kappa$, $\pi \xleftarrow{\$}$ $\mathsf{Perm}(m)$ and any random choices made by $\mathbb{D}$. We denote by $\mathbf{Adv}_{\mathcal{E}}^{\mathsf{sprp}}(t,q)$ the maximum advantage taken over all distinguishers $\mathbb{D}$ that run in time t and make q queries. Note: In the case where $\mathcal{E}_K$ is the Single-Key Even-Mansour construction using an ideal permutation P and key K, the distinguisher has also access to the underlying permutation (in both directions) in both worlds (and thus access to four oracles). As such, in the real world, $(\mathcal{O}_1, \mathcal{O}_2, \mathcal{O}_3, \mathcal{O}_4) = (\mathcal{E}_K, \mathcal{E}_K^{-1}, P, P^{-1})$, while in the ideal world $(\mathcal{O}_1, \mathcal{O}_2, \mathcal{O}_3, \mathcal{O}_4) = (\pi, \pi^{-1}, P, P^{-1})$. In the ideal world, P and π are independent.

Definition 40 (IND-CPA advantage). *Let Π be a block cipher-based AE scheme. The* IND-CPA advantage *over Π of a distinguisher $\mathbb{D}$ is defined as*

$$\mathbf{Adv}_{\Pi}^{\mathsf{priv}}(\mathbb{D}) = \left|\Pr_K\left[\mathbb{D}^{\mathcal{E}_K} = 1\right] - \Pr_{\bowtie}\left[\mathbb{D}^{\bowtie} = 1\right]\right|. \tag{3.57}$$

The probabilities are taken over $K \xleftarrow{\$} \mathcal{K}$, $\bowtie \xleftarrow{\$}$ $\mathsf{OPerm}(2n)$ and any random choices made by $\mathbb{D}$. By $\mathbf{Adv}_{\Pi}^{\mathsf{priv}}(t,q,\sigma,\ell)$ we denote the maximum advantage taken over all distinguishers $\mathbb{D}$ that run in time t and make q queries, of length at most ℓ blocks, and of total length at most σ blocks.

Definition 41 (INT-CTXT advantage). *Let Π be a block cipher-based AE scheme. The* INT-CTXT advantage *over Π of a distinguisher $\mathbb{D}$ is defined as*

$$\mathbf{Adv}_{\Pi}^{\mathsf{auth}}(\mathbb{D}) = \left|\Pr_K\left[\mathbb{D}^{\mathcal{E}_K,\mathcal{E}_K^{-1}} = 1\right] - \Pr_K\left[\mathbb{D}^{\mathcal{E}_K,\circeq} = 1\right]\right|. \tag{3.58}$$

The probabilities are taken over $K \xleftarrow{\$} \mathcal{K}$ and any random choices made by $\mathbb{D}$. We denote by $\mathbf{Adv}_{\Pi}^{\mathsf{auth}}(t,q,\sigma,\ell)$ the maximum advantage taken over all distinguishers $\mathbb{D}$ that run in time t and make q queries, of length at most ℓ blocks, and of total length at most σ blocks. We assume that $\mathbb{D}$ does not make a decryption query (A,C,T) if it has made a query (A,M) to the encryption oracle and obtained (C,T).

Definition 42 (IND-CPA advantage for permutation-based AE). *Let Π be a permutation-based AE scheme. The* IND-CPA advantage *over Π of a distinguisher $\mathbb{D}$ is defined as*

$$\mathbf{Adv}_{\Pi}^{\text{perm-priv}}(\mathbb{D}) = \left| \Pr_{K,\pi}\left[\mathbb{D}^{\mathscr{E}_K,\pi,\pi^{-1}} = 1\right] - \Pr_{\bowtie,\pi}\left[\mathbb{D}^{\bowtie,\pi,\pi^{-1}} = 1\right] \right|. \tag{3.59}$$

The probabilities are taken over $K \xleftarrow{\$} \mathscr{K}$, $\pi \xleftarrow{\$} \mathsf{Perm}(2n)$, $\bowtie \xleftarrow{\$} \mathsf{OPerm}(r)$, and any random choices made by $\mathbb{D}$. By $\mathbf{Adv}_{\Pi}^{\text{perm-priv}}(q,m)$ we denote the maximum advantage taken over all distinguishers $\mathbb{D}$ making q queries totaling m blocks.

Definition 43 (INT-CTXT advantage for permutation-based AE). *Let Π be a permutation-based AE scheme. The* INT-CTXT advantage *over Π of a distinguisher $\mathbb{D}$ is defined as*

$$\mathbf{Adv}_{\Pi}^{\text{perm-auth}}(\mathbb{D}) = \left| \Pr_{K,\pi}\left[\mathbb{D}^{\mathscr{E}_K,\mathscr{E}_K^{-1},\pi,\pi^{-1}} = 1\right] - \Pr_{\pi}\left[\mathbb{D}^{\mathscr{E}_K,\doteq,\pi,\pi^{-1}} = 1\right] \right|. \tag{3.60}$$

The probabilities are taken over $K \xleftarrow{\$} \mathscr{K}$, $\pi \xleftarrow{\$} \mathsf{Perm}(2n)$, and any random choices made by $\mathbb{D}$. By $\mathbf{Adv}_{\Pi}^{\text{perm-auth}}(q,m)$ we denote the maximum advantage taken over all distinguishers $\mathbb{D}$ making q queries totaling m blocks. We assume that $\mathbb{D}$ does not make a decryption query (A,C,T) if it has made a query (A,M) to the encryption oracle and obtained (C,T).

Patarin's H-coefficient Technique

For our proof of Lemma 9 in the following, we will rely on a proof technique due to Patarin [258] called the *H-coefficient technique*. We state here the result as we need it and refer to [258] and e.g. the work of Chen and Steinberger [93] for further details.

Let $\mathbb{D}$ be a distinguisher trying to distinguish between two worlds X and Y. The interaction of $\mathbb{D}$ is captured by a transcript which is denoted ω. We let D_X and D_Y denote the probability distribution over transcripts when interacting with worlds X and Y, respectively. Let $\mathscr{T}$ be the set of all feasible transcripts which is partitioned into a set of *good* and *bad* transcripts s.t. $\mathscr{T} = \mathscr{T}_{\text{good}} \cup \mathscr{T}_{\text{bad}}$. Now, consider a fixed distinguisher $\mathbb{D}$ and let ϵ be s.t. for all $\omega \in \mathscr{T}_{\text{good}}$ it holds that

$$\frac{\Pr[D_X = \omega]}{\Pr[D_Y = \omega]} \geq 1 - \epsilon. \tag{3.61}$$

Then, the H-coefficient technique says that $\mathbf{Adv}(\mathbb{D}) \leq \epsilon + \Pr[D_Y \in \mathscr{T}_{\text{bad}}]$.

Lemma 9 (Security bound on SEM). *Let $P : \mathbb{F}_2^{2n} \to \mathbb{F}_2^{2n}$ be an ideal permutation s.t. an adversary can make at most ρ evaluations of P in time t and let $\tilde{P}_K$ denote the SEM construction using P and key K. Then*

$$\mathbf{Adv}_{\tilde{P}_K}^{\text{sprp}}(t,q) \leq \frac{2\rho q}{2^{2n}}. \tag{3.62}$$

Proof. Let $\mathbb{D}$ be any distinguisher which can evaluate P at most ρ times in time t. For the proof we use Patarin's H-coefficient technique. Let X denote the real world in which $\mathbb{D}$ interacts with the oracles $(\mathscr{O}_1, \mathscr{O}_2, \mathscr{O}_3, \mathscr{O}_4) = (\tilde{P}_K, \tilde{P}_K^{-1}, P, P^{-1})$ and let Y denote the ideal world where $\mathbb{D}$ interacts with $(\mathscr{O}_1, \mathscr{O}_2, \mathscr{O}_3, \mathscr{O}_4) = (\pi, \pi^{-1}, P, P^{-1})$ for $K \xleftarrow{\$} \mathscr{K}$ and $\pi \xleftarrow{\$} \mathsf{Perm}(2n)$.

Denote by $\omega_{\mathscr{E}} = \{(s_i, t_i)\}_{i=1}^{q}$ the result of the interaction of $\mathbb{D}$ using q construction queries. Similarly, the result of the interaction using ρ queries to P we denote by $\omega_P = \{(x_i, y_i)\}_{i=1}^{\rho}$. To ease the analysis, the key K is disclosed at the end of the experiment (in the ideal world, a dummy key K is disclosed). We define a transcript as a tuple $(\omega_{\mathscr{E}}, \omega_P, K)$ and a bad transcript is such a tuple where it holds that

$$K \in \{s \oplus x, y \oplus t \mid (s,t) \in \omega_{\mathscr{E}} \wedge (x,y) \in \omega_P\}. \tag{3.63}$$

Bounding $\Pr[D_Y \in \mathscr{T}_{\text{bad}}]$. There are ρ pairs $(x,y) \in \omega_P$ and for each of them we consider each pair $(s,t) \in \omega_{\mathscr{E}}$. This means there are at most $q \cdot \rho$ values for $s \oplus x$, any of which equals K with probability 2^{-2n}. A similar argument applies to the probability of there being a pair (x,y) and (s,t) s.t. $t \oplus y = K$. As such, we find $\Pr[D_Y \in \mathscr{T}_{\text{bad}}] \leq \frac{2q\rho}{2^{2n}}$.

Bounding $\Pr[D_X = \omega]/\Pr[D_Y = \omega]$ **for** $\omega \in \mathscr{T}_{\text{good}}$. Consider some $\omega \in \mathscr{T}_{\text{good}}$. Let Ω_X and Ω_Y denote all possible oracles in the real world and ideal world, respectively. Correspondingly, let compat_X and compat_Y denote oracles in Ω_X, respectively Ω_Y, which are compatible with transcript ω.

Since the key space has size 2^{2n}, and there are $2^{2n}!$ permutations on $2n$ bits, we have that $\sharp\Omega_X = 2^{2n} \cdot 2^{2n}!$ and $\sharp\Omega_Y = 2^{2n} \cdot \left(2^{2n}!\right)^2$. Now, $\omega \in \mathscr{T}_{\text{good}}$ implies that any tuple in ω defines a unique input/output pair to P. As $\omega_{\mathscr{E}} \cup \omega_P$ consists of $q + \rho$ tuples, the number of compatible $2n$-bit permutations in the real world is $\sharp\mathsf{compat}_X = (2^{2n} - q - \rho)!$. Correspondingly, in the ideal world, the number of permutations compliant with P is $(2^{2n} - \rho)!$ while the number of permutations compliant with the construction queries is $(2^{2n} - q)!$. As such, $\sharp\mathsf{compat}_Y = \left(2^{2n} - q\right)!\left(2^{2n} - \rho\right)! \leq \left(2^{2n} - q - \rho\right)! 2^{2n}!$. By definition, we find that

$$\begin{aligned}
\Pr[D_X = \omega] &= \frac{(2^{2n} - q - \rho)!}{2^{2n} \cdot 2^{2n}!} \\
&= \frac{(2^{2n} - q - \rho)! 2^{2n}!}{2^{2n} \cdot (2^{2n}!)^2} \\
&\geq \frac{\sharp\mathsf{compat}_Y}{\sharp\Omega_Y} = \Pr[D_Y = \omega].
\end{aligned} \tag{3.64}$$

As such, we see $\Pr[D_X = \omega] \geq \Pr[D_Y = \omega]$, so $\epsilon \leq 0$ and by applying Patarin's H-coefficient technique, we find that

$$\mathbf{Adv}^{\mathsf{sprp}}_{\tilde{P}_K}(t,q) \leq \frac{2q\rho}{2^{2n}}. \tag{3.65}$$

□

With the statement of Lemma 9 in hand, we move on to show the security of our proposals next. In all three cases, we rely on the existing proofs for the modes of operation, when taking the final step to apply them to using the PRØST permutation as the idealized underlying primitive.

PRØST-COPA-*n*

Theorem 11 (IND-CPA for PRØST-COPA-n)**.** *Assume that* PRØST-n *is an ideal permutation and that an adversary can make at most* ρ *evaluations of the* PRØST-n *permutation in time* t'*, where* $t' \approx t$*. Then*

$$\mathbf{Adv}^{\mathsf{priv}}_{\text{PRØST-COPA-}n}(t,q,\sigma,\ell) \leq \frac{39(\sigma+q)^2}{2^{2n}} + \frac{8\rho(\sigma+q)}{2^{2n}} + \frac{(\ell+2)(q-1)^2}{2^{2n}}. \tag{3.66}$$

Proof. The proof follows from combining the proof for privacy of COPA [21, Theorem 2] with Lemma 9. □

Theorem 12 (INT-CTXT for PRØST-COPA-n)**.** *Assume that* PRØST-n *is an ideal permutation and that an adversary can make at most* ρ *evaluations of the* PRØST-n *permutation in time* t'*, where* $t' \approx t$*. Then*

$$\mathbf{Adv}^{\mathsf{auth}}_{\text{PRØST-COPA-}n}(t,q,\sigma,\ell) \leq \frac{39(\sigma+q)^2}{2^{2n}} + \frac{8\rho(\sigma+q)}{2^{2n}} + \frac{(\ell+2)(q-1)^2}{2^{2n}} + \frac{2q}{2^{\tau}}. \tag{3.67}$$

Proof. The proof follows from combining the proof for authenticity of COPA [21, Theorem 3] with Lemma 9. □

PRØST-OTR

The security proof for OTR by Minematsu [232] is technically in the ideal model, but it can be considered as being in the standard model by admitting the term $\mathbf{Adv}^{\mathsf{sprp}}_{\mathscr{E}}(t',2\sigma)$. In particular, this occurs by modeling the encrypting of a block in OTR, with its masking, as an XE construction (see the work by Rogaway [270]).

Theorem 13 (IND-CPA for PRØST-OTR-n)**.** *Assume that* PRØST-n *is an ideal permutation and that an adversary can make at most* ρ *evaluations of the* PRØST-n *permutation in time* t'*, where* $t' \approx t$*. Then*

$$\mathbf{Adv}^{\mathsf{priv}}_{\text{PRØST-OTR-}n}(t,q,\sigma,\ell) \leq \frac{6(\sigma+q)^2}{2^{2n}} + \frac{4\rho(\sigma+q)}{2^{2n}}. \tag{3.68}$$

Proof. The proof follows from combining three parts: the proof for privacy of OTR [232, Theorem 1] in the ideal model; the fact that the modeling of OTR using XE-blocks admits the term $\mathbf{Adv}^{\mathsf{sprp}}_{\mathscr{E}}(t',2\sigma)$ in the standard model (where $t' \approx t$); and Lemma 9 which gives the term $\mathbf{Adv}^{\mathsf{sprp}}_{\mathscr{E}}(t',2\sigma)$. □

Theorem 14 (INT-CTXT for PRØST-OTR-n)**.** *Assume that* PRØST-n *is an ideal permutation and that an adversary can make at most* ρ *evaluations of the* PRØST-n *permutation in time* t'*, where* $t' \approx t$*. Then*

$$\mathbf{Adv}^{\mathsf{auth}}_{\text{PRØST-OTR-}n}(t,q,\sigma,\ell) \leq \frac{6(\sigma+q)^2}{2^{2n}} + \frac{4\rho(\sigma+q)}{2^{2n}} + \frac{q}{2^{\tau}}. \tag{3.69}$$

Proof. The proof follows from combining three parts: the proof for authenticity of OTR [232, Theorem 2] in the ideal model; the fact that the modeling of OTR using XE-blocks admits the term $\mathbf{Adv}^{\mathsf{sprp}}_{\mathscr{E}}(t',2\sigma)$ in the standard model (where $t' \approx t$); and Lemma 9 which gives the term $\mathbf{Adv}^{\mathsf{sprp}}_{\mathscr{E}}(t',2\sigma)$. □

PRØST-APE-$n[r,c]$

In this section we present security bounds for PRØST-APE-$n[r,c]$. The proofs of security for the APE mode of operation from [24] assume an ideal permutation, thus under this assumption, the security bounds for the APE construction carry directly over to PRØST-APE-$n[r,c]$. Note that the security bounds do not depend on the time t used by the distinguisher. Indeed, the bounds hold for the strongest type of distinguishers whose time complexity is unbounded; only the number of queries q and their total length m matter.

Theorem 15 (IND-CPA for PRØST-APE-$n[r,c]$). *Assume that* PRØST-n *is an ideal permutation. Then*

$$\mathsf{Adv}^{\text{perm-priv}}_{\text{PRØST-APE-}n[r,c]}(q,m) \leq \frac{m^2}{2^{2n}} + \frac{m(m+1)}{2^c}. \tag{3.70}$$

Proof. The proof is given in [24, Theorem 1]. □

Theorem 16 (INT-CTXT for PRØST-APE-$n[r,c]$). *Assume that* PRØST-n *is an ideal permutation. Then*

$$\mathsf{Adv}^{\text{perm-auth}}_{\text{PRØST-APE-}n[r,c]}(q,m) \leq \frac{m^2}{2^{2n}} + \frac{2m(m+1)}{2^c}. \tag{3.71}$$

Proof. The proof is given in [24, Theorem 2]. □

3.2.7 External Analysis

Since its submission in early 2014, with the CAESAR competition being a high-profile topic, PRØST has received a lot of attention from the cryptographic community. In the following, we discuss the analyses published up until the time of writing.

In [90], Canteaut and Roué study improved bounds on the *maximum expected differential probability* (MEDP) and the *maximum expected square correlation* (MELP) for substitution-permutation networks. Essentially, their work presents a framework for analyzing the upper bound on the resistance of an SPN to differential- and linear cryptanalysis over 2 rounds. With respect to PRØST, the authors consider an SPN with the same building blocks as the PRØST permutation, and show bounds on the MEDP and MELP for two rounds of such an SPN.

An integral attack employing the *Fast Fourier Transform* (FFT) is presented on 8 rounds of PRØST-128 and on 9 rounds of PRØST-256 by Todo and Aoki in [290]. Using the integral distinguishers, the authors are able to perform key-recovery attacks, for the number of rounds mentioned, on $\tilde{P}_{n,K}$. The attack on $\tilde{P}_{128,K}$ has a complexity of 2^{64} data and 2^{80} time, while the complexity for the attack on $\tilde{P}_{256,K}$ is 2^{65} data and $2^{80.9}$ time.

In [117], Dobraunig, Eichlseder, and Mendel show that forgery attacks are possible against PRØST-OTR-n in the related-key model. That is, an attacker is assumed to be able to obtain inputs (N,A,M) and corresponding outputs (N,A,C,T) for some fixed key K, and the attack then shows that it is possible to forge valid outputs for an instantiation of PRØST-OTR-n with a key K' which is related to K in a particular way. Referring to the illustration of PRØST-OTR-n of Figure 3.6, the related-key forgery essentially works as follows. By choosing a related key $K' = K \oplus \Delta$, and related (padded) nonce $\mathsf{pad}_{2n}(N') = \mathsf{pad}_{2n}(N) \oplus \Delta$, where Δ has zeroes on the least $2n-\eta$ bits, the values δ and L in the figure become affected by Δ in a controllable way. In particular, one can choose different message blocks, modified by Δ in a particular way, such that one obtains the same ciphertexts. The details of the attack are slightly more complicated than that, but we do not go into the details here. We remark, however, that the attacks presented in [117] do not contradict the proofs of Section 3.2.6, as they do not cover the related-key setting.

In [180], PRØST-OTR-n is under further scrutiny as Karpman presents a way to turn a related-key distinguisher on the Even-Mansour construction into a related-key key-recovery attack. With this, the analysis is applicable to PRØST-OTR-n. It uses relations of two kinds for the key: XOR and modular addition. We remark that neither of analyses presented in [117] nor [180] are applicable to PRØST-COPA-n as it uses the encryption of zeroes, $L = \mathscr{E}_K(0^{2n})$, for encryption and tag generation, and thus can not be modified via a change in the nonce, as is the case for PRØST-OTR-n.

We already saw how the work of [42], as presented in Section 3.1, generalizes the analysis of bounds on the resistance towards differential- and linear cryptanalysis of AES-like primitives, including the PRØST permutation. With the analysis, we found that the choice for rotation matrix in PRØST-128 is not optimal with respect to the considered metrics. However, we remark that the optimized rotation matrix has not been adopted into a new version of the PRØST permutation.

Most recently, Mennink defines in [229] a tweakable block cipher XPX which generalizes a range of existing constructions: Even-Mansour; the XEX construction of Rogaway; as well as the tweakable Even-Mansour used in the CAESAR proposal Minalpher [280]. The XPX construction is shown to be strong tweakable pseudo-random permutation under certain conditions on the tweak space. Furthermore, and more interestingly, XPX is proven secure against related-key attacks with various key relation functions, depending on the tweak space used. Of particular interest to our proposal, it is shown that PRØST-COPA-n is secure against related-key attacks, when either the key K' is related to K, PRØST-$n(K')$ is related to PRØST-$n(K)$, or both, by the XOR with some value Δ.

3.2.8 Discussion and Conclusions

In this section we have described PRØST, our proposal for a new permutation-based scheme for authenticated encryption with associated data. PRØST was submitted in March 2014 as a first-round candidate to the ongoing CAESAR competition. The PRØST submission contains six ranked proposals, employing three existing third-party modes of operations COPA, OTR, and APE, in a combination with the PRØST permutation using two different permutation sizes.

A total of 57 submissions were made to the competition. During the first year after submission, the cryptographic community were occupied analyzing the proposals. Often, total breaks were made due to small mistakes such as lack of domain separation between processing of associated data and message. In many cases, such observations led to submitters publishing updated versions with minor tweaks on the CAESAR competition mailing list. In this, PRØST was no exception, as we gave in January 2015 an updated version PRØST v1.1. For our case, the minor tweaks consisted of: the choice to always do ciphertext expansion by padding, as explained in Section 3.2.4; corrections regarding the number of rounds required to obtain full diffusion; and other minor changes. We refer to http://proest.compute.dtu.dk for the PRØST v1.1 specification containing the full list of changes. While some attacks could be subverted using only minor tweaks, other times the attacks were more fatal, such as the key-recovery attacks described on

AVALANCHE in Section 2.3.1 and on CALICO in [79]. As mentioned, PRØST did not move to the second round, despite the absence of any serious attacks.

While there certainly is room for improvements in PRØST, e.g. with respect to the enhanced parameters for the $\mathsf{ShiftPlanes}_t$ operation of Section 3.1, we believe that the design and choice of modes of operation is sound. In particular, we believe that the direction taken with respect to using several round-dependent permutations is interesting, as it leads to better diffusion in elongated state representations such as that of PRØST. As PRØST follows the wide-trail design strategy, allowing to prove better bounds on the resistance towards attacks such as differential- and linear cryptanalysis, this creates word alignment on the rows of the state. This can be seen as contrary to KECCAK, which avoids word alignment, but where proving bounds is difficult. We highlight as an open problem for future work a trade-off between the ability to prove good bounds on one hand, and the absence of word alignment on the other hand.

4 Implementation Aspects

When discussing the implementation of a cryptographic algorithm, we necessarily need to do so with one or several particular platforms in mind. Some applications such as RFID tags have extremely limited resources available for cryptographic implementations, in terms of e.g. chip area and power, and thus require specially tailored algorithms. Meanwhile, a popular web content provider needs to service many connections simultaneously using fairly standard hardware, and thus would like to use an encryption method which delivers high performance in software. In this chapter, we do not consider hardware implementations; rather, our scope is high-speed implementation of block cipher modes of operation.

With cryptography in general, but especially with symmetric cryptography, there is a strong dependency between the development of cryptographic algorithms and that of computer technology. Quite naturally, with off-the-shelf CPUs having an increasing computing power, the time it takes to execute a cryptographic attack decreases. This is an example of how development in technology affects the design of a symmetric cryptography. However, advances made in cryptographic research also influence the development of general-purpose computers. The prime example of this is the decision, by leading chip manufacturer Intel, to include special hardware support for the AES block cipher since the introduction of their *Westmere* microarchitecture in 2010. The work presented in this chapter evolves around the use of exactly these special instructions.

The performance of symmetric primitives in software is extremely important. In the past decades, the Internet has grown tremendously. We have seen a massive shift towards streamed content, where e.g. a piece of music or a video is being played while the download progresses, rather than being played locally from the disk. Examples of extremely popular services include YouTube, Netflix, HBO, Hulu, Spotify and Pandora, to name a few. At any time, such providers service an enormous user base over SSL/TLS connections, so it is of utmost importance that the cryptographic algorithms used in those protocols have a high performance. In fact, historically there has been reluctance to offer secure connections via SSL/TLS due to performance concerns.

However with the recent advances in this area, this is no longer true. For example, Adam Langley of Google said [205]:

> On our production frontend machines, SSL/TLS accounts for less than 1% of the CPU load, less than 10 KB of memory per connection and less than 2% of network overhead. Many people believe that SSL/TLS takes a lot of CPU time and we hope the preceding numbers will help to dispel that.

This example highlights the point: optimizations in the performance of cryptographic algorithms and implementations *matter*. Such efforts help minimize the cost of using e.g. SSL or TLS, which could otherwise be significant, thereby effectively facilitating the use of encryption at a mass scale.

The efforts to implement high-speed symmetric cryptography are focused on two levels: the high-level choices and the low-level optimization, and their interaction. The former has to do with particular choices of parameters in, for example, a cryptographic protocol for communication over the Internet. In such a case, choices include mode of operation to use, the underlying primitive, and block and key sizes. Those choices influence both the performance as well as security. For the low-level optimizations, the implementer must make a choice about the use of registers, instructions and their scheduling, so as to make the implementation perform optimally. In this chapter, we will consider optimizations on both levels: we investigate the performance obtained with several modes of operation, when implemented on a recent Intel microarchitecture, employing instruction tailored scheduling of the aforementioned special AES instructions available.

4.1 Motivation

In this chapter, we focus on high-performance software implementations of AES-based block cipher modes of operation. In particular, we consider several NIST-recommended modes for both encryption and AEAD, as well a single MAC. We also analyze the performance of all AES-based schemes from the first round of the ongoing CAESAR competition. To that end, we note that only modes using the *full* AES-128 are considered. We make this clarification, since some CAESAR proposals use round-reduced variants of AES-128.

Given the very recent Intel microarchitecture codenamed *Haswell*, which provides interesting optimizations for special instructions for the AES (we discuss these in detail in Section 4.3.2 below), we use Haswell as our target platform in this chapter. Traditionally, when benchmarking the performance of block ciphers and modes, a *single* message of a particular length, i.e. a particular number of blocks, is used. As we describe in greater detail later, this means that parallelizable modes of operation such as CTR mode are able to achieve very good performance figures, because they can operate on the many message blocks independently of each other. However, this is not true for sequential modes of operation, and comparatively, they have a quite bad performance. One of the major contributions of this chapter is, that we consider the performance of AES-based modes of operation on Haswell, using *multiple messages*. In other words, instead of adhering to the traditional thinking of the performance using a single message, we now consider the performance under the assumption that several messages are available for processing at once. In particular, we give a very simple and efficient algorithm, the *comb scheduler*, for scheduling the processing of the message blocks. With this approach, we show that sequential modes are able to achieve very good performance, even comparable to parallelizable modes.

The structure of this chapter is as follows. First, we give an overview of previous work on fast AES implementations, not considering modes of operation. We then continue to present the modes we consider in this work. In Section 4.3, we describe, with a focus on Haswell, the aforementioned special AES instructions that are available on modern Intel CPUs, and other concepts such as instruction pipelining, which are crucial to understanding our proposed comb scheduler. In Section 4.4, we describe the comb scheduler along with various considerations relating to it. In Sections 4.5 and 4.6 we apply the comb scheduler to a range of NIST-recommended modes and CAESAR proposals, respectively. Finally, we give our discussion and concluding remarks in Section 4.7. In this chapter, all mentions of the AES is to be understood as AES-128.

4.1.1 Timeline of AES Implementations

Naturally, Rijndael won the AES competition not only because it is a very secure block cipher, but also because it has excellent implementation characteristics on many platforms. However, the story of optimized AES implementations only begins with its standardization in 2001.

The first commonly referred AES implementations in the literature are the table-based implementations by Gladman [143]. His implementation in 2001 achieved a performance of approximately 25 cycles per byte (cpb). In 2007, Matsui and Nakajima [220] provided bit-sliced implementations performing at 9.2 cpb, however not counting the overhead associated with converting between standard- and bit-sliced representations of the AES state. A year later, in 2008, Bernstein and Schwabe obtained a table-based implementation of the AES, focusing on micro-optimizations, performing at 10.5 cpb [50]. This is to be compared to Gladman's implementations now performing at about 16 cpb. Käsper and Schwabe provided optimized bit-sliced implementations in 2009 [198], performing at 7 cpb, this including the overhead associated with the aforementioned state conversion. As already stated, Intel's first microarchitecture Westmere implementing dedicated AES instructions was launched in 2010, achieving a very high performance of 1.25 cpb. Since 2010, three new microarchitectures have been released by Intel under the names *Sandy Bridge*, *Ivy Bridge*, and Haswell. On Sandy Bridge, AES implementations using new instructions are performing at around 0.64 cpb. On the most recent architecture Haswell, the performance of the AES is all the way down to 0.625 cpb. Since Sandy Bridge, implementations of the AES can enjoy the extra instructions and larger register sizes provided by the *Advanced Vector Extensions* (AVX) and AVX2. We describe these in more detail in Section 4.3.6.

Publications

The results presented in this section are from:

[80] Andrey Bogdanov, Martin M. Lauridsen, and Elmar Tischhauser. Comb to Pipeline: Fast Software Encryption Revisited. In Leander [210], pages 150–171.

Author Contribution

The author is responsible for a large part of the twelve Haswell-optimized implementations of block cipher modes of operation considered in this work. Furthermore, the author is responsible for the principle behind the comb scheduler algorithm proposed in Section 4.4, which forms the basis of the consideration of performance of AES-based modes in this chapter.

4.2 Schemes Considered

In the following, we outline the block cipher modes of operation that we analyze in this chapter. We partition the modes into two groups: the NIST-recommended modes on one hand, and the CAESAR proposals on the other hand.

4.2.1 NIST-recommended Modes

In its special publications SP-800-38A-D [121], NIST recommends the following modes of operation: ECB, CBC, CFB, OFB and CTR as basic encryption modes; CMAC as authentication mode; and CCM and GCM as authenticated encryption modes. In Section 1.2.1 we described the encryption modes while GCM and CCM were covered in Section 1.2.4.

Cipher-based MAC (CMAC) is a block cipher-based MAC. At its core is XCBC, a MAC algorithm designed by Black and Rogaway [64], which was submitted to NIST as an improvement to CBC-MAC. Later refinements by Iwata and Kurosawa turned XCBC into *One-Key CBC-MAC* (OMAC) [165]. Later, OMAC was refined into OMAC1 [166], which is equivalent to CMAC. CMAC works by first computing two keys (K_1, K_2) from the master key K as

$$K_1 = \begin{cases} K_0 \lll 1 & , \mathsf{msb}_1(K_0) = 0 \\ (K_0 \lll 1) \oplus c & , \mathsf{msb}_1(K_0) = 1 \end{cases}, \quad \text{and} \tag{4.1}$$

$$K_2 = \begin{cases} K_1 \lll 1 & , \mathsf{msb}_1(K_1) = 0 \\ (K_1 \lll 1) \oplus c & , \mathsf{msb}_1(K_1) = 1, \end{cases} \tag{4.2}$$

where $K_0 = \mathcal{E}_K(0^n)$ and c is a particular constant. Consider message blocks $M_1, \dots, M_\ell$. If $|M_\ell| = n$ then we modify it by setting $M_\ell = M_\ell \oplus K_1$ and otherwise we set $M_\ell = \mathsf{pad}_n(M_\ell) \oplus K_2$. Now, the MAC over the message is computed as $T = \mathsf{msb}_\tau(C_\ell)$, where for $i = 1, \dots, \ell$,

$$C_i = \mathcal{E}_K(C_{i-1} \oplus M_i), \tag{4.3}$$

and where we define $C_0 = 0^n$.

4.2.2 Authenticated Encryption Modes and CAESAR

Besides the widely employed and standardized modes CCM and GCM, a great number of modes for authenticated encryption have been proposed, many of them being contestants in the currently ongoing CAESAR competition. We give a brief overview of the AE modes in this study, with our consideration split into two classes. In the first class we have the modes offering no security under nonce misuse (the NAE notion); we refer to these as *nonce-based* in the following. The second class consists of modes offering nonce misuse above the NAE notion; we refer to them as *nonce misuse resistant* (somewhat abusing the terminology). The modes considered in the former camp are CCM, GCM, OCB3, OTR, CLOC, COBRA, JAMBU and SILC, while the nonce misuse resistant modes considered are McOE-G, COPA, POET and Julius.

Table 4.1 gives a comparison of the modes considered in this work. The price to pay for a mode to offer a security notion stronger than NAE includes extra computation, a higher serialization degree, or both. One of the fundamental questions we answer in this work is how much one has to pay, in terms of performance, to maintain some level of security under nonce misuse.

Table 4.1: Overview of the AE modes considered in this paper. The $\|$ column indicates parallelizability; the *IF* (inverse-free) column indicates whether a mode needs the inverse of the underlying block cipher in decryption/verification; the $\mathcal{E}$ and *M* columns give the number of calls, per message block, to the underlying block cipher and multiplications in $\mathbb{F}_{2^n}$, respectively.

Mode	Reference	$\|$	IF	$\mathcal{E}$	M	Description
Nonce-based						
CCM	[296]		✓	2	–	CTR encryption, CBC-MAC authentication
GCM	[224]	✓	✓	1	1	CTR mode with chain of multiplications
OCB3	[196]	✓		1	–	Gray code-based xor-encrypt-xor (XEX)
OTR	[232]	✓	✓	1	–	Two-block Feistel structure
CLOC	[168]		✓	1	–	CFB mode with low overhead
COBRA	[26]	✓	✓	1	1	Combining OTR with chain of multiplications
JAMBU	[299]		✓	1	–	AES in stream mode, lightweight
SILC	[169]		✓	1	–	CLOC with smaller hardware footprint
Nonce misuse resistant						
McOE-G	[128]			1	1	Serial multiplication-encryption chain
COPA	[21]	✓		2	–	Two-round XEX
POET	[6]	✓		3	–	XEX with two AXU (full AES-128 call) chains
Julius	[36]			1	2	SIV with polynomial hashing

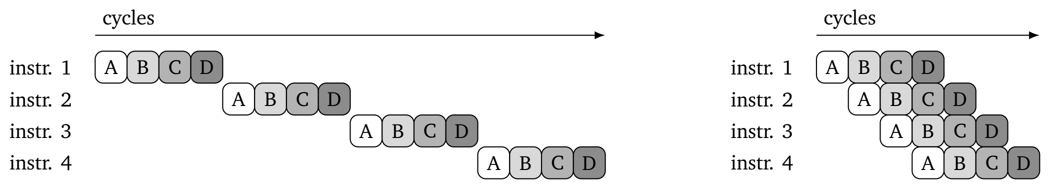
In Section 1.2.4, we already introduced the CCM, GCM, OCB3, OTR, and COPA block cipher modes of operation for AEAD. The description of the remaining modes is beyond the thesis at hand, and we refer to the relevant references listed in Table 4.1. We clarify that for COBRA we refer to the FSE 2014 version with its reduced security claims (compared to the withdrawn CAESAR candidate); with POET we refer to the version where the universal hashing is implemented as full AES-128 (since it would not otherwise comprise a mode of operation); and with Julius, we mean the CAESAR candidate regular Julius-ECB.

4.3 The Intel Instruction Set and Haswell

In this section, we describe theory pertaining to highly optimized implementations of the AES on high-end CPUs. Our treatment of the topics of special AES instructions, instruction pipelining and AVX, is made particularly in relation to the recent Haswell microarchitecture.

4.3.1 Instruction Pipelining

When an instruction is being executed on a CPU, the execution happens in different *stages*. For example, a classical *Reduced Instruction Set Computing* (RISC) pipeline works in five stages: instruction fetch, decoding, execution, memory access and writeback [156, Appendix C]. Naturally, the stages of a particular instruction need to happen in succession, as they depend on each other since the initiation of a stage depends on the completion of the former stage. However, if one considers two or more *independent* instructions, i.e. instructions working on completely independent data, then it is theoretically possible to execute the instructions at the same time *but in different stages*. This is what is utilized with instruction pipelining. The concepts of serial- and pipelined execution of instructions are illustrated in Figure 4.1.



(a) Serial execution **(b)** Pipelined execution

Figure 4.1: Serial execution (left) and pipelined execution (right) of four instructions (1 through 4), each executing in four stages (A through D)

In our fictional example, we see from Figure 4.1a, that with serial execution, we require 16 cycles to execute four instructions, each in four stages. However, if the instructions are independent, we see from Figure 4.1b that several instructions can be executed simultaneously, so long as they are in different stages. The result is, that we reduce the total number of cycles to 7 to complete the execution of all four instructions. The performance of a pipelined instruction is characterized by its latency L, the number of cycles to complete one instruction, and throughput T, the number of instructions that can be issued every clock cycle. For instance, with the pipelined instructions of Figure 4.1b, we have a latency of $L = 4$ cycles and a throughput of $T = 1$ instruction/cycle.

4.3.2 AES New Instructions

During the 00s, with the increasing number of applications, protocols and standards relying on the AES, it became clear that it had come to stay as the dominant block cipher of the next few decades. To that end, Intel proposed in 2008 a set of special instructions for fast AES encryption and decryption for their CPUs [146]. The instructions are called the *AES New Instruction Set* (AES-NI), and have been implemented since their 2010 Westmere microarchitecture. AES-NI provides instructions for computing one AES round with the `aesenc` instruction (`aesenclast` for the last, special round of the AES), its inverse round function instruction `aesdec` (again, with `aesdeclast` for the last round), and auxiliary instructions for key scheduling. The instructions do not only offer better performance than other implementations of the AES not using the special instructions, but security as well, since they are leaking no timing information. In other words, each instruction is constant-time, so the time to execute the operation does not depend on the data on which it executes, as opposed to e.g. using a table lookup for a straightforward implementation of the AES S-box. The AES-NI instructions are supported on a subset of Westmere, Sandy Bridge, Ivy Bridge and Haswell microarchitectures. A range of AMD processors also support the instructions under the name *AES Instructions*, including processors in the Bulldozer, Piledriver and Jaguar series [159].

In Haswell, the AES-NI encryption and decryption instructions had their latency improved from $L = 8$ cycles on Sandy and Ivy Bridge[1], down to $L = 7$ cycles [150]. This is especially beneficial for sequential modes such as CBC, CCM, McOE-G, CLOC, SILC and JAMBU. Furthermore, the throughput has been slightly optimized, allowing for better performance in parallel. Table 4.2 gives an overview of the latencies and inverse throughputs measured on our test machine (Core i5-4300U). The data was obtained using the test suite of Fog [129].

Table 4.2: Experimental latency L (cycles) and inverse throughput T^{-1} (cycles/instruction) of AES-NI and `pclmulqdq` instructions on Intel's Haswell microarchitecture

Instruction	L	T^{-1}
`aesenc`	7	1
`aesdec`	7	1
`aesenclast`	7	1
`aesdeclast`	7	1
`aesimc`	14	2
`aeskeygenassist`	10	8
`pclmulqdq`	7	2

4.3.3 Improvements to Finite Field Multiplications

The `pclmulqdq` instruction was introduced by Intel along with the AES-NI instructions [148], but is not part of AES-NI itself. The instruction takes two 128-bit inputs and a byte input `imm8`, and performs carry-less multiplication of a combination of one 64-bit half of each operand. The choice of halves of the two operands to be multiplied is determined by the value of bits 4 and 0 of `imm8`.

Most practically used AE modes employing multiplication in a finite field use block lengths of $n = 128$ bits. As a consequence, multiplications are in the field $\mathbb{F}_{2^{128}}$. As the particular choice of finite field does not influence the security proofs, modes use the tried-and-true GCM finite field. For our performance study, we have used two different implementation approaches for finite field

[1]We remark that Fog reports a latency of $L = 4$ cycles for `aesenc` on Ivy Bridge [129]

multiplication, which we in general denote `gfmul`. The first implementation, which we refer to as the *classical method*, was introduced in Intel's white paper [148]. It applies `pclmulqdq` three times in a carry-less Karatsuba multiplication followed by modular reduction. The second implementation variant, which we refer to as the *Haswell-optimized method*, was proposed by Gueron [147] with the goal of leveraging the much improved `pclmulqdq` performance on Haswell (see Table 4.2) to trade many shifts and XORs for one more multiplication. This is motivated by the improvements in both latency (7 vs. 14 cycles) and inverse throughput (2 vs. 8 cycles) on Haswell [150].

In modes where the output of a multiplication over $\mathbb{F}_{2^{128}}$ is not directly used, other than as a part of a chain combined using addition, the *aggregated reduction* method by Jankowski and Laurent [171] can be used to gain speed-ups. This method uses the inductive definitions of chaining values combined with the distributivity law for the finite field to postpone modular reduction, at the cost of storing powers of an operand. Among the modes we benchmark in this work, the aggregated reduction method is applicable only to GCM and Julius. We therefore use this approach for those two modes, but apply the general `gfmul` implementations to the other modes.

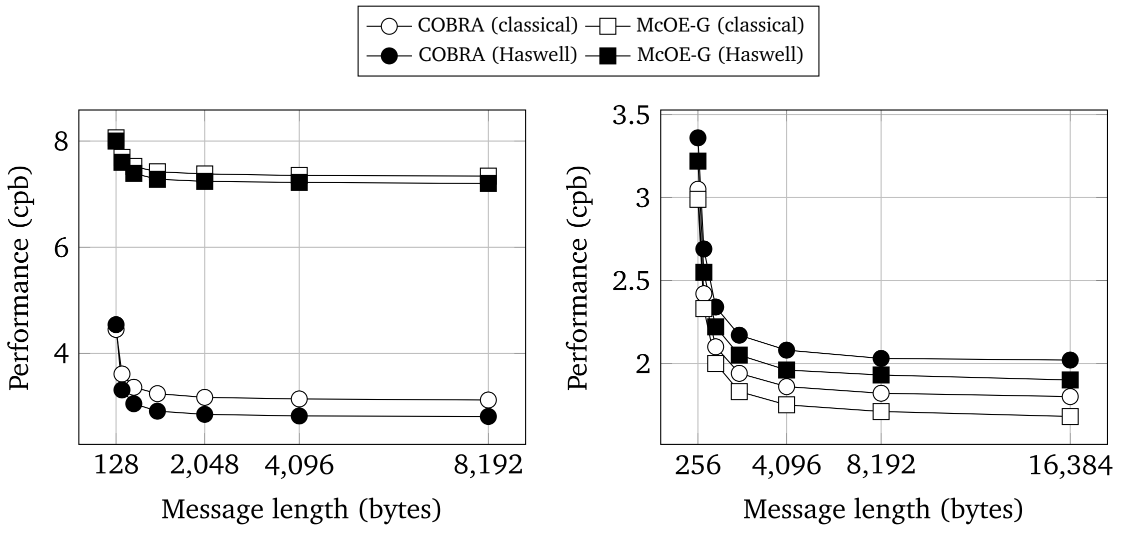
4.3.4 Classical vs. Haswell Multiplication

Here we compare the classical and Haswell-optimized methods of multiplication in $\mathbb{F}_{2^{128}}$. We compare the performance of the AE modes considered that use full $\mathbb{F}_{2^{128}}$ multiplications (as opposed to aggregated reduction), McOE-G and COBRA, when instantiated using the two different multiplication methods. Figure 4.2 shows that when processing a single message, the Haswell-optimized method performs better than the classical implementation of `gfmul`, while the situation is the other way around, when processing multiple messages in parallel.

Considering the optimizations made for the `pclmulqdq` instruction on Haswell, these observations make perfect sense. When processing only a single message, there is no independent data available on which to draw parallelism. As such, and since the finite field multiplication in COBRA and McOE-G is sequential, this becomes a bottleneck for single message processing, and the optimizations made to the instruction come to their right. On the other hand, when processing multiple messages, there is enough independent data to draw on to keep the pipeline filled, so the latency improvement of the instruction vanishes, and in turn the four instruction calls for the Haswell multiplication method aggravate the overall latency.

4.3.5 Haswell-Optimized Finite Field Doubling

The doubling operation in $\mathbb{F}_{2^{128}}$ is commonly used in AE schemes [28], and indeed among the schemes we benchmark, it is used by OCB3, OTR, COBRA, COPA and POET. Doubling in this field consists of left shifting the input by one bit and doing a conditional XOR of a reduction polynomial *if and only if* the most significant bit of the input equals one. Neither SSE+ nor AVX provide an instruction to shift a full `xmm` register bitwise, nor to directly test only its most significant bit. As such, these functions have to be emulated with other operations, opening up a number of implementation choices.



(a) Processing a single message **(b)** Processing multiple messages

Figure 4.2: Performance of COBRA and McOE-G using the classical- and Haswell multiplication methods for a single message (left) and 8 multiple messages of equal length (right)

Listing 4.1: Doubling in $\mathbb{F}_{2^{128}}$

```
 1  __m128i xtime(__m128i v) {
 2    __m128i v1, v2;
 3    v1 = _mm_slli_epi64(v, 1);
 4    v2 = _mm_slli_si128(v, 8);
 5    v2 = _mm_srli_epi64(v2, 63);
 6    if (msb of v == 1)
 7      return _mm_xor_si128
 8             (_mm_or_si128(v1, v2), RP);
 9    else
10      return _mm_or_si128(v1, v2);
11  }
```

Table 4.3: Performance of doubling with different approaches to MSB testing

	Approach	Cycles
(i)	Extraction	15.4
(ii)	Test	15.4
(iii)	MSB mask	16.7
(iv)	Compare + extract	5.6

We emulate a left shift by one bit by the following procedure, which is optimal with regard to the number of instructions and cycles: given an input $v \in \mathbb{F}_{2^{128}}$, the value $2v \in \mathbb{F}_{2^{128}}$ is computed as in Listing 4.1. Consider $v = v_L \| v_R$ where v_L and v_R are 64-bit values. In line 3 we set $v_1 = (v_L \ll 1)\|(v_R \ll 1)$ and lines 4 and 5 set first $v_2 = v_R \| 0^{64}$ and then $v_2 = (v_R \gg 63)\|0^{64}$. As such, we have $v \ll 1 = v_1 \mid v_2$. This leaves us with a number of possibilities when implementing the branching of line 6, which can be categorized as (i) extracting parts from v and testing, (ii) AVX variants of the `test` instruction, (iii) extracting a mask with the most significant bit of each part of v and (iv) comparing against a mask `MSB_MASK` $=$ `80`$\cdots$`00` and then extracting from the comparison result. Some of these approaches again leave several possibilities regarding the number of bits extracted, etc.

Interestingly, the approach taken to check the most significant bit of v has a substantial impact on the doubling performance. This is illustrated by Table 4.3 where we give performance of the doubling operation using various combinations of approaches. The numbers are obtained by averaging over 10^8 experiments. Surprisingly, we see that there is a significant speed-up, about a factor $\times 3$, when using comparison with MSB_MASK combined with extraction, over the other methods. Thus, we suggest to use this approach, where line 6 of Listing 4.1 can be implemented as

if (_mm_extract_epi8(_mm_cmpgt_epi8(MSB_MASK, v), 15) == 0).

4.3.6 General Considerations: AVX and AVX2 Instructions

In our Haswell-optimized AE scheme implementations we make heavy use of AVX which has been present in Intel processors since the Sandy Bridge microarchitecture. AVX can be considered as an extension of the *streaming SIMD extensions* (SSE) instructions[2], and later versions such as SSE2 through SSE4, operating on 128-bit registers xmm0 through xmm15. While AVX and AVX2, the latter which appears first on Haswell, brings mainly support for 256-bit wide registers to the table, this is not immediately useful in implementing an AES-based modes, as the special AES instructions as well as the pclmulqdq instruction support only the use of 128-bit xmm registers. However, a feature of AVX that we use extensively is the three-operand enhancement, due to the VEX coding scheme, of legacy two-operand SSE2 instructions. This means that, in a single instruction, one can non-destructively perform binary vector operations on two operands and store the result in a third operand, rather than overwriting one of the inputs. For example, one can perform the operation $Z = X \oplus Y$ rather than $X = X \oplus Y$. This eliminates overhead associated with mov operations required when overwriting an operand is not acceptable.

A further Haswell feature worth taking into account is the increased throughput for logical instructions such as vpxor/vpand/vpor on AVX registers: while the latency remains at one cycle, now up to 3 such instructions can be scheduled simultaneously. Notable exceptions are algorithms heavily relying on mixed 64/128 bit logical operations such as JAMBU, for which the inclusion of a fourth 64-bit *arithmetic logic unit* (ALU) implies that such algorithms will actually benefit from frequent conversion to 64-bit arithmetic via the vpextrq/vpinsrq instructions, rather than artificial extension of 64-bit operands to 128 bits for operation on the AVX registers.

On Haswell, the improved memory controller allows two simultaneous 16-byte aligned moves via vmovdqa from registers to memory, with a latency of one cycle. This implies that on Haswell, the comparatively large latency of cryptographic instructions such as aesenc or pclmulqdq allows the implementer to “hide” more memory accesses to the stack when larger internal state of the algorithm leads to register shortage. This also aids the generally larger working sets induced by the multiple message strategy described in Section 4.4.1.

4.4 Comb Scheduler: An Efficient Look-Ahead Strategy

A substantial number of block cipher modes of operation for (authenticated) encryption are inherently sequential in nature. Among the NIST-recommended modes, this includes the classic

[2]We denote by SSE+ also later versions, including SSE2 through SSE4

CBC, OFB, CFB and CCM modes as well as CBC derivatives such as CMAC. Also, more recent designs essentially owe their sequential nature to design goals, e.g allowing lightweight implementations or achieving stricter notions of security, for instance not requiring a nonce for security (or allowing its reuse). Examples of such include ALE, a design by Bogdanov, Mendel, Regazzoni, Rijmen, and Tischhauser [78], APE [24], the CLOC mode by Iwata, Minematsu, Guo, and Morioka [168], the McOE family of algorithms by Fleischmann, Forler, Lucks, and Wenzel [127, 128], and some variants of the POET mode [6] by Abed, Fluhrer, Forler, List, Lucks, McGrew, and Wenzel. While being able to perform well in other environments, such algorithms cannot benefit from the available pipelining opportunities on contemporary general-purpose CPUs. For instance, as detailed in Section 4.3.2, the AES-NI encryption instructions on Haswell feature a high throughput of $T = 1$ instruction/cycle, but a relatively high latency of $L = 7$ cycles. Modes of operation that need to process data sequentially will invariably be penalized in such environments.

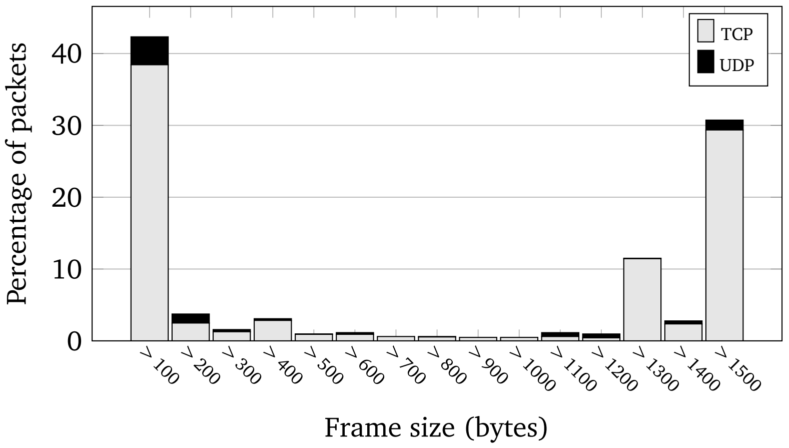


Figure 4.3: Distribution of frame sizes for TCP and UDP

Furthermore, even if designed with parallelizability in mind, (authenticated) modes of operation for block ciphers typically achieve their best performance when operating on somewhat longer messages, often due to the simple fact that these diminish the impact of potentially costly initialization phases and tag generation. Equally importantly, only longer messages allow high-performance software implementations to make full use of the available pipelining opportunities [11, 147, 196, 223]. In practice, however, one rarely encounters messages which allow to achieve the maximum performance of an algorithm. Recent studies on packet sizes on the Internet demonstrate that they basically follow a bimodal distribution [174, 239, 259]: 44% of packets are between 40 and 100 bytes long; 37% are between 1400 and 1500 bytes in size; the remaining 19% are somewhere in between. Throughout this chapter, we refer to this as the *realistic* distribution of message lengths. A distribution of frame sizes in TCP and UDP from [239] is shown in Figure 4.3. This emphasizes the importance of good performance for messages up to around 2 KB, as opposed to longer messages. Second, when looking at the weighted distribution, this implies that the vast majority of data is actually transmitted in packets of medium size between 1 and 2 KB. Considering the first mode of the distribution, we remark that many of the very small packets of Internet traffic comprise TCP ACKs (which are typically not encrypted), and that the use of authentication and encryption layers such as TLS or IPsec incurs overhead significant enough to blow up a payload of 1 byte to a 124 byte packet [164]. It is therefore this range of message sizes (128 to 2048 bytes) that authenticated modes of encryption should excel at processing, when employed for encryption of Internet traffic.

4.4.1 Filling the Pipeline: Multiple Messages

It follows from the discussion above that the standard approach of considering one message at a time, while arguably optimizing message processing latency, can not always generate optimal throughput in high-performance software implementations in most practically relevant scenarios. This is not surprising for the inherently sequential modes, but even when employing a parallelizable design, the prevailing distribution of message lengths makes it hard to achieve the best performance. In order to remedy this, we propose to consider the scheduling of multiple messages in parallel *already in the implementation of the algorithm itself,* as opposed to considering it as a (single-message) black box to the message scheduler. This opens up possibilities of increasing the performance in the cases of both sequential modes and the availability of multiple shorter or medium-sized messages. In the first case, the performance penalty of sequential execution can potentially be hidden by filling the pipeline with a sufficient number of operations on independent data. In the second case, there is a potential to increase performance by keeping the pipeline filled also for the overhead operations such as block cipher- or multiplication calls during initialization or tag generation.

Note that while we consider the processing of multiple messages on a single core, the multiple message approach naturally extends to multi-core settings. Conceptually, the transition of a sequential- to a multiple message implementation can be viewed as similar to the transition from a straightforward to a bit-sliced implementation approach. We note also, that an idealistic view of multiple-message processing was given in [78] for the dedicated authenticated encryption algorithm ALE. This consideration was rather rudimentary, did not involve real-world packet size distributions, and did not treat any modes of operation. It is also important to note, that while multiple message processing has the potential to increase the throughput of an implementation, it can also increase its latency (see also Section 4.4.3). The degree of parallelism therefore has to be chosen carefully and with the required application profile in mind.

4.4.2 Message Scheduling with a Comb

Consider the scenario where a number of messages of varying lengths need to be processed by a sequential encryption algorithm. As outlined before, blocks from multiple messages have to be processed in an interleaved fashion in order to make use of the available inter-message parallelism. Having messages of different lengths implies that generally the pipeline cannot always be filled completely. At the same time, the goal to schedule the message blocks such that pipeline usage is maximized has to be weighed against the computational cost of making such scheduling decisions: in particular, every conditional statement during the processing of the bulk data results in a pipeline stall.

Algorithm 10: COMBSCHEDULER

```
Data: $k$ messages $M_1, \dots, M_k$ of lengths $\ell_1, \dots, \ell_k$ blocks, parallelism degree $P$
$L \leftarrow$ list of tuples $(M_i, \ell_i)$, $1 \leq i \leq k$, sorted by decreasing $\ell_i$
Denote by $L[i] = (M_i, \ell_i)$ the $i^{\text{th}}$ tuple in $L$
while $|L| > 0$ do                              // Loop while messages are still to be processed
    $r \leftarrow \min\{P, |L|\}$
    Perform initialization for messages $M_1, \dots, M_r$
    $\mathscr{P}, \mathscr{B} \leftarrow$ PRE-COMPUTEWINDOWS$(\ell_1, \dots, \ell_r)$
    $completedBlocks \leftarrow 0$
    for $w = 1, \dots, |\mathscr{P}|$ do                                    // Loop over windows
        for $i = 1, \dots, \mathscr{B}[w]$ do                        // Loop over blocks in window
            for $j = 1, \dots, \mathscr{P}[w]$ do                  // Loop over messages in window
                Process block $(completedBlocks + i)$ of message $M_j$
            end
        end
        $completedBlocks \leftarrow completedBlocks + \mathscr{B}[w]$
    end
    Perform finalization for messages $M_1, \dots, M_r$
    Remove the $r$ first elements from $L$
end
```

In order to reconcile the goal of exploiting multi-message parallelism for sequential algorithms with the need for low-overhead scheduling, we propose *comb scheduling*. Comb scheduling is based on the observation that ideally, messages processed in parallel have the same length, so given a desired (maximum) parallelism degree P and a list of message lengths $\ell_1, \dots, \ell_k$, we can subdivide the computation in a number of *groups*, in each of which we process as many consecutive message blocks as we can in so-called *windows*, for as many independent messages as possible, according to the restrictions based on the given message lengths. Since our scheduling problem exhibits optimal substructure, this greedy approach yields an optimal solution. Furthermore, the scheduling decisions of how many blocks are to be processed at which parallelism level can be pre-computed once the ℓ_i, $1 \leq i \leq k$, are known. This implies that instead of making each processing step conditional, we only have conditional statements whenever we proceed from one group to the next. Our proposed comb scheduling method is outlined in Algorithms 10 and 11.

In order to simplify the combing, the messages are pre-sorted by decreasing length. This sorting step can be implemented via an optimal sorting network for the constant value of P chosen by the implementation, and can employ pointer swapping only, without copying of data blocks. Alternatively, a low-overhead algorithm like *insertion sort* can be used. The sorted messages are then processed in groups of P. A pre-computation is performed to determine the windows inside the group, i.e. how many windows are required to process the group, and for each window, how many messages still have blocks left to be processed (and how many blocks need processing in the windows). This information is returned in the lists $\mathscr{P}$ and $\mathscr{B}$ by Algorithm 11. Inside each group, the processing is window by window according to the pre-computed parallelism levels $\mathscr{P}$

Algorithm 11: PRE-COMPUTEWINDOWS($\ell_1,\dots,\ell_r$)

```
Data: $r$ message lengths $\ell_1,\dots,\ell_r$ in blocks, s.t. $\ell_i \geq \ell_{i+1}$ with $1 \leq i < r$
Result: List $\mathscr{P}$ with $\mathscr{P}[w]$ the number of messages to process in parallel in window $w$ and
        list $\mathscr{B}$ with $\mathscr{B}[w]$ the number of blocks to process in window $w$
$\mathscr{P} \leftarrow [],\quad \mathscr{B} \leftarrow []$                          // Initialize to empty lists
$w \leftarrow 1,\quad q_{last} \leftarrow 0,\quad i \leftarrow r$
while $i > 1$ do                                        // Scan windows right to left
    $q \leftarrow \ell_i,\quad j \leftarrow i-1$
    while $j \geq 1$ and $\ell_j = \ell_i$ do $j \leftarrow j-1$      // Left-extend while lengths are equal
    $\mathscr{P}[w] \leftarrow i$
    $\mathscr{B}[w] \leftarrow q - q_{last}$
    $q_{last} \leftarrow q,\quad i \leftarrow j,\quad w \leftarrow w+1$
end
if $i = 1$ then                                              // Leftover message
    $\mathscr{P}[w] \leftarrow 1$
    $\mathscr{B}[w] \leftarrow \ell_1 - q_{last}$
end
return $\mathscr{P}, \mathscr{B}$
```

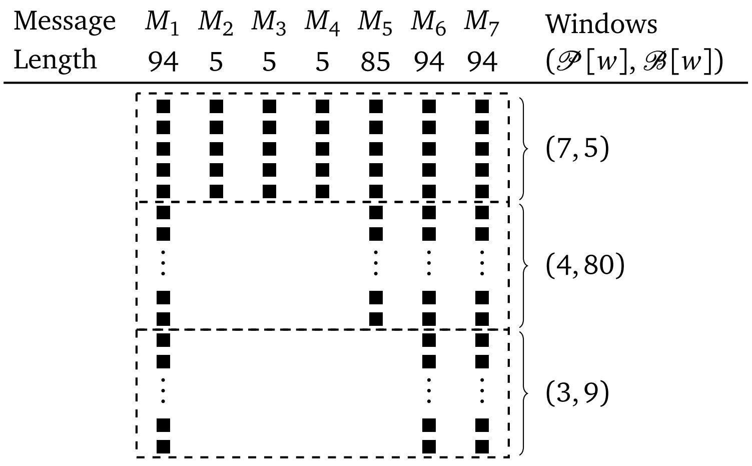

Message M_1 M_2 M_3 M_4 M_5 M_6 M_7
Length 94 5 5 5 85 94 94
Windows
$(\mathscr{P}[w], \mathscr{B}[w])$
(7,5)
(4,80)
(3,9)

Figure 4.4: Comb scheduling example for 7 messages of lengths $(\ell_1,\dots,\ell_7) = (94,5,5,5,85,94,94)$ blocks

and window lengths $\mathscr{B}$: in window w, the same $\mathscr{P}[w]$ messages of the current message group are processed $\mathscr{B}[w]$ blocks further. In the next window, at least one message will be exhausted, and the parallelism level decreases by at least one. As comb scheduling is processing the blocks by common (sub-)length from left to right, our method can be considered a symmetric-key variant of the well-known comb method for (multi-)exponentiation [212].

Example 5. *We illustrate comb scheduling in Figure 4.4 with an example where $P = k = 7$: The* *pre-computation determines that all* $\mathcal{P}[1] = 7$ *messages can be processed in a pipelined fashion for the first* $\mathcal{B}[1] = 5$ *blocks;* $\mathcal{P}[2] = 4$ *of the* 7 *messages can be processed further for the next* $\mathcal{B}[2] = 80$ *blocks; and finally* $\mathcal{P}[3] = 3$ *remaining messages are processed for another* $\mathcal{B}[3] = 9$ *blocks.*

Choice of Parallelism Degree

In order to make optimal use of the pipeline, the parallelism degree P should be chosen according to

$$P = L \cdot T, \tag{4.4}$$

with L denoting the latency (in cycles) and T the throughput (in instructions/cycle) of the pipelined instruction. For AES-NI, the latency and throughput of the `aesenc` instruction vary from platform to platform. From the summary for the Haswell microarchitecture of Table 4.2 in Section 4.3.2, this suggests $P = 7$ for this platform.

4.4.3 Latency vs. Throughput

A point worth discussing is the latency increase one has to pay when using multiple message processing. Since the speed-up is limited by the parallelization level, one can at most hope for the same latency as in the sequential processing case. We illustrate this by the example of CBC mode when implemented in the multiple message setting with comb scheduling. We consider two distributions for message lenghts: one where all messages are 2048 bytes long, and one realistic distribution of Internet traffic. The performance data is given in Table 4.4.

Table 4.4: Performance (in cpb) of CBC encryption and relative speed-up for comb scheduling with different parallelization levels for fixed message lengths of 2048 bytes (top) and realistic message lengths (bottom)

		Parallelization level P						
	Sequential	2	3	4	5	6	7	8
2048-byte messages	4.38	2.19	1.47	1.11	0.91	0.76	0.66	0.65
Relative speed-up	×1.00	×2.00	×2.98	×3.95	×4.81	×5.76	×6.64	×6.74
Realistic distribution	4.38	2.42	1.73	1.37	1.08	0.98	0.87	0.85
Relative speed-up	×1.00	×1.81	×2.53	×3.20	×4.06	×4.47	×5.03	×5.15

What we can see from Table 4.4 is, that for messages of an identical length of 2 KB, the ideal linear speed-up of a factor P is actually achieved for $P \in \{2, 3, 4\}$ parallel messages: setting $|M| = 2048$, instead of waiting $4.38 \cdot |M|$ cycles in the sequential case, one has a latency of either $2.19 \cdot 2 = 4.38 \cdot |M|$ cycles when $P = 2$; when $P = 3$ the latency is $1.47 \cdot 3 = 4.41 \cdot |M|$ cycles; and when $P = 4$ the latency is $1.11 \cdot 4 = 4.44 \cdot |M|$ cycles. Starting from $P = 5$ parallel messages, the latency slightly increases with the throughput, however remaining at a manageable level even for $P = 7$ parallel messages, where it is only around 5% higher than in the sequential case, while achieving a 6.64 times increase in throughput. For realistic message lengths, using $P = 7$

multiple messages, we see an average increase in latency of 39% which has to be contrasted to (and, depending on the application, weighed against) the significant 5.03 times increase in throughput.

4.4.4 Patenting

With the potentially substantial performance benefit due to comb scheduling, as we detail in Sections 4.5 and 4.6, the inventors of the algorithm have, through the Technical University of Denmark, filed a patent application with the European Patent Office under application number 15157994.3-1870. Possible uses include e.g. employment of comb scheduling in heavily loaded servers, where a sequential block cipher mode of operation is used to provide encrypted content to clients.

4.5 Pipelined NIST-recommended Modes

In this section, we present the results of our performance study of the NIST-recommended encryption- and MAC modes when instantiated with AES as the block cipher, and implemented with AES-NI and AVX vector instructions. Reminding that some modes covered, such as CBC and CFB, are sequential in encryption but parallel in decryption, we remark that we only benchmark encryption in this work.

Experimental Setting

All measurements were taken on a single core of an Intel Core i5-4300U CPU (Haswell) at 1900 MHz. For each combination of parameters, the performance was determined as the median of 91 averaged timings of 200 measurements each. This method has also been used by Krovetz and Rogaway in their benchmarking of authenticated encryption modes in [196]. The measurements are taken over samples from the realistic distribution on message lengths.

Out of the basic NIST-recommended modes, ECB and CTR are inherently parallelizable and already achieve good performance with trivial sequential message scheduling. Three other modes, CBC, OFB and CFB, however, are inherently sequential and therefore need to make use of inter-message parallelism to benefit from the available pipelining. The same holds for the NIST-recommended CMAC message authentication code. We therefore measure the performance of all modes with sequential processing, and additionally the performance of the sequential modes with comb message scheduling.

Discussion

Our performance results for pipelined implementations of NIST-recommended modes are presented in Table 4.5. It is apparent that the parallel processing of multiple messages using comb scheduling speeds up encryption performance by a factor of around 5, bringing the sequential modes within about 10% of the performance of CTR mode. The results also indicate that the overhead induced by the comb scheduling algorithm itself can be considered negligible compared to the AES calls.

Table 4.5: Performance comparison (in cpb) of NIST-recommended encryption- and MAC modes, with trivial sequential processing and with comb scheduling. Message lengths are sampled from the realistic Internet traffic distribution.

Mode	Sequential processing	**Comb scheduling**	Speed-up
AES-ECB	0.65	–	–
AES-CTR	0.78	–	–
AES-CBC	4.47	**0.87**	×5.14
AES-OFB	4.48	**0.88**	×5.09
AES-CFB	4.45	**0.89**	×5.00
CMAC-AES	4.29	**0.84**	×5.10

Due to their simple structure with almost no overhead, it comes as no surprise that CBC, OFB and CFB performance are virtually identical. That CMAC performs slightly better despite additional initialization overhead can be explained by the fact that there are no ciphertext blocks to be stored to memory.

4.6 Pipelined Authenticated Encryption

We now turn our attention to the AES-NI software performance of authenticated encryption modes. We consider the well-established modes CCM, GCM and OCB3 as well as a number of more recent proposals, many of them being contestants in the ongoing CAESAR competition.

Experimental Setting

The same experimental setup as for the NIST-recommended modes above applies. For our performance measurements, we are interested in the performance of the various AE modes of operation during their *bulk processing* of message blocks, i.e. during the encryption phase. To that end, we *do not* measure cycles spent on processing associated data. As some schemes can have a significant overhead when computing authentication tags for short messages, we *do* include this phase in the measurements as well.

4.6.1 Performance in the Real World

Out of the AE modes in consideration, GCM, OCB3, OTR, COBRA, COPA and Julius are parallelizable designs. We therefore only measure their performance with sequential message processing. On the other hand, CCM, CLOC, SILC, JAMBU, McOE-G and POET are sequential designs and as such will also be measured in combination with comb scheduling. In all cases, we again measure the performance using message lengths sampled from the realistic bimodal distribution of typical Internet traffic.

Table 4.6 lists the results of the performance measurements. For the parallelizable modes where comb scheduling was implemented, the relative speed-up compared to normal sequential

Table 4.6: Performance comparison (in cpb) of AES-based AE modes with trivial sequential processing and comb scheduling. Message lengths are sampled from the realistic Internet traffic distribution. Proposals from the CAESAR competition are marked by a †.

(a) Nonce-based AE modes

Mode	Sequential	**Comb**	Speed-up
CCM	5.22	**1.64**	×3.18
GCM	1.63	–	–
OCB3†	1.51	–	–
OTR†	1.91	–	–
COBRA	3.56	–	–
CLOC†	4.47	**1.45**	×3.08
JAMBU†	9.12	**2.05**	×4.45
SILC†	4.53	**1.49**	×3.04

(b) Nonce misuse resistant modes

Mode	Sequential	**Comb**	Speed-up
McOE-G	7.41	**1.79**	×4.14
COPA†	2.68	–	–
POET†	5.85	**2.14**	×2.73
Julius†	3.73	–	–

processing is indicated in the last column. In this table, the nonce-based AE modes are listed separately from those offering some level of nonce misuse resistance, in order to provide a better estimation of the performance penalty one has to pay for achieving a stricter notion of security.

Discussion

The performance data demonstrates that comb scheduling of multiple messages consistently provides a speed-up of factors between 3 and 4, compared to normal sequential processing. For typical Internet packet sizes, comb scheduling enables sequential AE modes to run with performance comparable to the parallelizable designs, in some cases even outperforming them. This can be attributed to the fact that AE modes typically have heavier initialization and finalization than normal encryption modes, consisting of setting up variables and generating the authentication tag, both implying a penalty in performance for short messages. By using comb scheduling, however, also the initial and final AES calls can be (at least partially) parallelized between different messages. The relative speed-up for this will typically reduce with the message length. The surprisingly good performance of McOE-G is due to the fact that it basically benefits doubly from multiple message processing: not only the AES calls, but also its sequential finite field multiplications, can now be pipelined. For the comb scheduling implementation of CCM, which is two-pass, it is worth noting that all scheduling pre-computations only need to be done once, since exactly the same processing windows can be used for both passes.

Best Performance Characteristics

From Table 4.6, it is apparent that for encryption of typical Internet packets, the difference between sequential and parallelizable modes, with respect to performance, somewhat blurs when comb scheduling is employed. This is especially true for the nonce-based setting, where CLOC, SILC, CCM, GCM and OCB3 all perform on a very comparable level. For the nonce misuse resistant modes, our results surprisingly show better performance of the two sequential modes for this application scenario. This can be attributed to the fact that the additional processing needed for achieving nonce misuse resistance hampers performance on short messages, which can be mitigated to some extent by comb scheduling.

4.6.2 Traditional Approach: Sequential Messages of Fixed Lengths

While the previous section analyzed the performance of the various AE modes using a model for a realistic message lengths, we provide some more detail on the exact performance exhibited by these modes for a range of *fixed* message lengths in this section. To that end, we provide performance measurements for specific message lengths between 128 and 2048 bytes. The results are summarized in Table 4.7.

Table 4.7: Performance comparison (in cpb) of AE modes for processing a single message of various fixed message lengths

(a) Nonce-based modes

	Message length (bytes)				
Mode	128	256	512	1024	2048
CCM	5.35	5.19	5.14	5.11	5.10
GCM	2.09	1.61	1.34	1.20	1.14
OCB3	2.19	1.43	1.06	0.87	0.81
OTR	2.97	1.34	1.13	1.02	0.96
CLOC	4.50	4.46	4.44	4.46	4.44
COBRA	4.41	3.21	2.96	2.83	2.77
JAMBU	9.33	9.09	8.97	8.94	8.88
SILC	4.57	4.54	4.52	4.51	4.50

(b) Nonce misuse resistant modes

	Message length (bytes)				
Mode	128	256	512	1024	2048
McOE-G	7.77	7.36	7.17	7.07	7.02
COPA	3.37	2.64	2.27	2.08	1.88
POET	6.89	5.74	5.17	4.88	4.74
Julius	4.18	4.69	3.24	3.08	3.03

Discussion

The performance data of Table 4.7 clearly shows the expected difference between sequential and parallelizable modes when no use of multiple parallel messages can be made. Among the sequential modes, only initialization-heavy modes such as McOE-G and POET show significant performance differences between shorter and longer messages, while this effect usually is very pronounced for the parallelizable modes. It can be seen from Table 4.7, that for the nonce-based modes, the best performance is generally offered by OCB3, although OTR and GCM provide quite similar performance on Haswell. Among the nonce misuse resistant modes, COPA has the best performance for all message sizes.

4.6.3 Exploring the Limits: Upper Bounding the Combing Advantage

Having seen the performance data with comb scheduling for realistic message lengths, it is natural to consider the question what the performance of the various modes would be for the ideal scenario where the scheduler is given only messages of a fixed length. In this case, the comb pre-computation would result in only one processing window, so essentially no scheduler-induced branches are needed during the processing of the messages. In a sense, this constitutes an *upper bound* for the multi-message performance with comb scheduling for the various encryption algorithms. Table 4.8 summarizes the performance of the previously considered sequential AE modes when comb scheduling is combined with fixed message lengths.

Table 4.8: Performance comparison (in cpb) of sequential AE modes when comb scheduling is used for various fixed message lengths

(a) Nonce-based modes

	Message length (bytes)				
Mode	128	256	512	1024	2048
CCM	1.51	1.44	1.40	1.38	1.37
CLOC	1.40	1.31	1.26	1.24	1.23
JAMBU	2.14	1.98	1.89	1.85	1.82
SILC	1.43	1.33	1.28	1.25	1.24

(b) Nonce misuse resistant modes

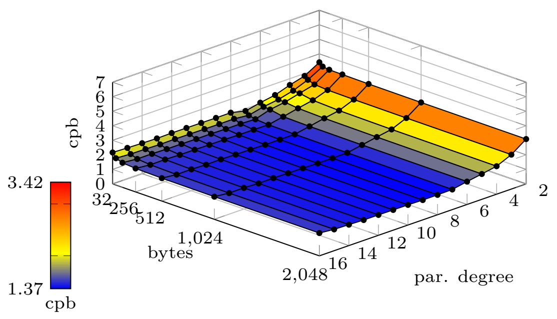
	Message length (bytes)				
Mode	128	256	512	1024	2048
McOE-G	1.91	1.76	1.68	1.64	1.62
POET	2.56	2.23	2.06	1.97	1.93

Discussion

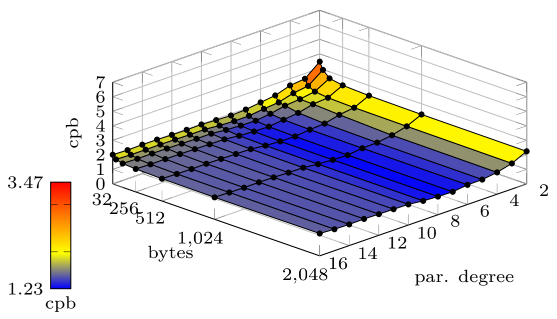
It can be seen that for all modes considered, the performance for longer messages at least slightly improves compared to the realistic message length mix of Table 4.6, although the differences are quite small and do not exceed around 0.2 cpb. For shorter messages, the difference can be more pronounced for a mode with heavy initialization such as POET. Overall, this shows that comb scheduling for a realistic distribution provides a performance which is very comparable to that of comb scheduling of messages with an idealized distribution.

Exploring the Parameter Space

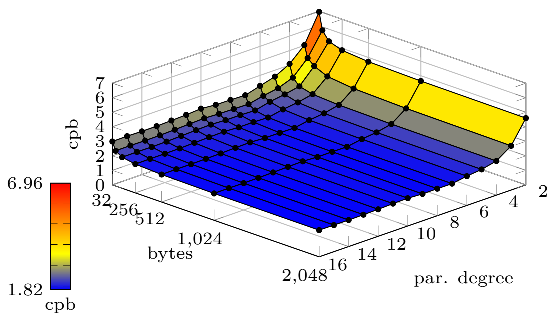
Besides the distribution of the message lengths, the parallelization degree influences the performance of the comb scheduler. Even though $P = 7$ is optimal for Haswell, applications might choose a lower value if typically only few messages are available simultaneously, in order to avoid a latency blowup. The dependency of the performance on both individual parameters is further detailed in Figures 4.5 and 4.6, where the comb scheduling performance is shown for a range of fixed message lengths ranging from 32 bytes to 2048 bytes, and parallelization degrees $P \in \{2, \dots, 16\}$. The horizontal lines in the color key of each plot indicate the integer values in the interval.



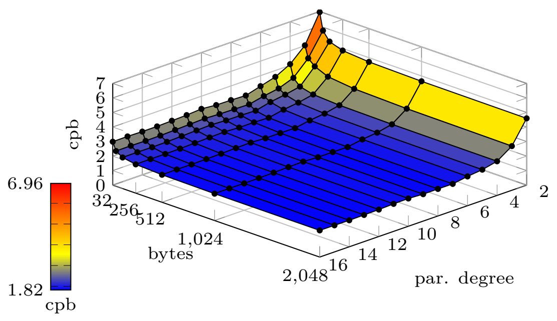
(a) CCM



(b) CLOC

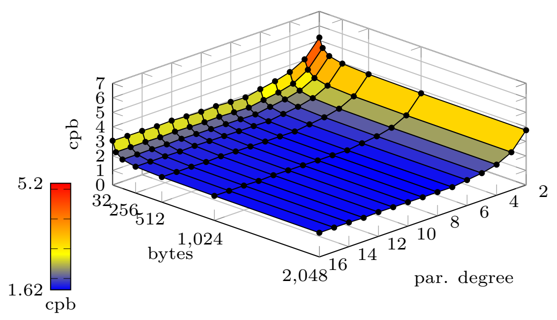


(c) SILC

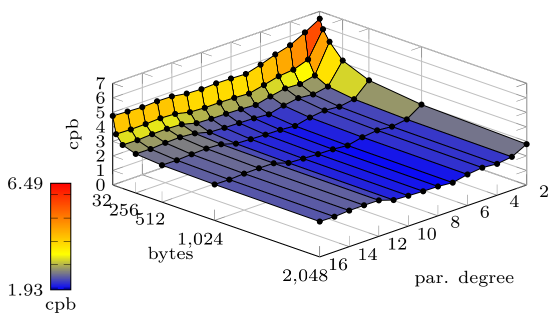


(d) JAMBU

Figure 4.5: Performance of sequential nonce-based AE modes when comb scheduling is used with different parallelization levels for various fixed message lengths



(a) McOE-G



(b) POET

Figure 4.6: Performance of sequential nonce misuse resistant AE modes when comb scheduling is used with different parallelization levels for various fixed message lengths

Impact of Working Set Sizes

It can be seen from the plots of Figures 4.5 and 4.6 that, as expected, most modes achieve their best speed-up in the multiple messages scenario for a parallelization level of around 7 messages. It is worth noting, however, that for each of these messages, a complete working set, i.e. the internal state of the algorithm, has to be maintained. Since only sixteen 128-bit `xmm` registers are available in Haswell, even a working set of three 128-bit words (for instance cipher state, tweak mask, checksum) for $P = 7$ simultaneously processed messages will already exceed the number of available registers. As the parallelization degree P increases, the influence of this factor increases. This can be seen especially for POET, which has a larger internal state per instance. By contrast, CCM, JAMBU and McOE-G suffer a lot less from this effect.

The experimental results also confirm the intuition of Section 4.3.6 that Haswell's improved memory interface can handle fairly large working set sizes efficiently by hiding the stack access latency between the cryptographic operations. This allows more multiple messages to be processed faster despite the increased register pressure, basically until the number of moves exceeds the latency of the other operations, or ultimately the limits of the Level-1 cache are reached.

4.7 Discussion and Conclusions

In this chapter, we have discussed the performance of various block cipher-based symmetric primitives when instantiated with the AES on Intel's recent Haswell architecture.

As a general technique to speed up both inherently sequential modes, and to deal with the typical scenario of having many shorter messages, we proposed our comb scheduler, an efficient algorithm for the scheduling of multiple simultaneous messages, which is based on a look-ahead strategy within a certain window size. This leads to significant speed-ups for essentially all sequential modes, even when taking a realistic bimodal Internet traffic distribution into account. Applied to the NIST-recommended modes CBC, CFB, OFB and CMAC, comb scheduling attains a significant speed-up of factor at least 5, resulting in a performance of around 0.88 cpb, which is within about 10% of the performance of the parallelizable CTR mode on the same message distribution.

We also applied comb scheduling to authenticated encryption modes which typically feature higher initialization and finalization overhead, thus penalizing performance on the frequently occurring short messages. With comb scheduling, we attain speed-ups of the inherently sequential AE modes CCM, CLOC, SILC, JAMBU, McOE-G and POET by factors between 3 and 4.5. This particularly results in a CCM performance comparable to GCM or OCB3, but without being afflicted by issues with weak-key classes such as those of GCM, or being encumbered by patents as OCB3, as discussed in Section 1.2.4.

Our study also establishes that for practitioners wishing to use a nonce misuse resistant AE mode, the POET design with comb scheduling attains better performance than the completely parallelizable mode COPA. Since POET furthermore offers some robustness under release of unverified plaintext (see Section 1.2.4), this suggests that users do *not* have to choose between good performance or stricter notions of security.

5 Conclusions

In this thesis, we have studied symmetric primitives from the perspectives of cryptanalysis, design and implementation aspects. We summarize our contributions, and discuss open problems and future work below.

5.1 Contributions

With respect to analysis, we applied variants of differential cryptanalysis to the recent lightweight block cipher SIMON which was proposed by the NSA in 2013. We also described a connection between differential characteristics and linear trails for SIMON, and discussed the differential effect in the smallest member of the SIMON family. In the framework of linear cryptanalysis, we introduced a theoretical model which allows an adversary to distinguish a block cipher in the key-less setting. Contrary to previous models using differential properties, our model is the first ever to use linear cryptanalysis. We applied the model to the ISO-standardized block cipher PRESENT, and show distinguishers on up to 26 rounds of PRESENT-80 and 27 rounds of PRESENT-128. Finally, we presented structural weaknesses in two authenticated encryption schemes, AVALANCHE and RBS, and leveraged these weaknesses to present very efficient attacks that fully recover the secret key in both cases. Our attacks show that one must be careful with unusual design choices such as having a nonce-dependent encryption key which is the case for AVALANCHE.

In the area of symmetric primitive design, we studied AES-like ciphers with respect to diffusion properties and resistance to differential- and linear cryptanalysis. Our focus was on the ShiftRows-like operation of the round function which, contrary to the linear layer, has not previously received any systematic analysis. With our work, we moved in a direction to close this gap. The study consisted of a range of results effectively classifying the possible operations, allowing for a computer-aided search of significantly lower complexity. As a tangible outcome, we present optimal rotation matrices for AES-like ciphers for a range of geometries, and provide improved parameters for Rijndael-192 and Rijndael-256, as well as the PRIMATEs-80 and PRØST-128 permutations. Next, we presented our new permutation-based approach to authenticated encryption with associated data, called PRØST. The proposal consists of a newly designed and highly secure permutation, combined with three existing third-party modes of operation. We presented our own cryptanalytic results of the PRØST permutation, and discussed analyses made by the cryptographic community. Furthermore, we gave proofs of security for the proposals, based on the security proofs for the modes of operation. PRØST was submitted to the ongoing CAESAR competition, but did not advance to the second round.

Finally, motivated by the improved instructions for AES-NI and carry-less multiplication on Intel's most recent microarchitecture Haswell, we conduct a thorough benchmarking of several block cipher modes on this platform. In particular, we implemented several NIST-recommended modes, as well as authenticated encryption modes from the CAESAR competition, all using AES-128 as the underlying block cipher. We proposed the comb scheduler, a low-overhead look-ahead algorithm for scheduling the processing of multiple messages in a pipelined fashion. With this technique, we saw that especially sequential modes, but also parallelizable modes to a lesser extent, all benefit from this approach, also in a setting using data packet lengths representative of typical Internet traffic.

5.2 Open Problems and Future Work

In the timeline of cryptanalysis on SIMON, the results described in this thesis made an early appearance. As such, our attacks are not close to the best published to this date. Indeed, advances are made frequently, slightly reducing attack complexities and occasionally breaking one more round. With that said, we believe that SIMON is a solid block cipher. Most cryptanalytic focus on SIMON from the community seems to be of the differential- or linear kind. Very few papers deviate from this trend by studying the choices made in *designing* the cipher. We believe much more analysis is required in this area, especially if standardization should be considered.

With respect to the analysis on PRESENT, it will be interesting to see how the approach is applicable to other block ciphers. One interesting target would be KATAN which, despite its very different round function, exhibits similar linear properties. In general, we believe that there is more work to be done on linear distinguishers in the key-less setting. It is an open question whether it is possible to extend the deterministic phase for PRESENT to cover more rounds. More research is needed on the relation between sidestepping rounds in an attack, such as in our deterministic phase or in a rebound attack, and the degrees of freedom spent in doing so. Many existing works investigate this for differential properties, while in contrast the data points for linear cryptanalysis are very few.

The wide-trail design strategy, as also employed in PRØST, gives designers a valuable tool to prove security bounds against e.g. differential- and linear attacks. However, this approach also implies strong word alignment in the rows. It will be interesting to see what future design approaches try to trade off this deficiency with the ability to prove good bounds. Along this line, an open problem from our optimization of the ShiftRows-like operation in AES-like ciphers, is that of incorporating the rotation matrix normal form into the MILP model itself, rather than using the model as a black box solver for a particular rotation matrix.

Finally, it will be interesting to see how approaches such as the comb scheduler find their use in protocol implementations. With regard to the trending concept of the *Internet of Things*, we believe that the increasing volume of clients served by major content providers gives merit to the approach, with its processing of multiple messages in a pipeline. This is especially true with respect to the increased throughput for sequential modes of operation, such as the widely deployed AES-CBC.

A
Truncated Difference Propagations for SIMON

Table A.1 gives the truncated difference propagations for block sizes $n \in \{96, 128\}$. The cases for $n \in \{32, 48, 64\}$ are given in Section 2.1.8.

Table A.1: Truncated differential pattern propagation for SIMON using block sizes $n \in \{96, 128\}$, with an input difference $0 \cdots 01 \| 0 \cdots 0$

(a) $n = 96$

Rounds	Left halves
0	000000000000000000000000000000000000000000000001
1	000000000000000000000000000000000000000*000001*0
2	0000000000000000000000000000000*00000**00001**01
3	00000000000000000000000*00000**0000***0*01***0*0
4	000000000000000*00000**0000***0*0******1******01
5	0000000*00000**0000***0*0********************1*0
6	00000**0000***0*0*****************************0*
7	000***0*0**************************************0
8	0***0*
9	**

Rounds	Right halves
0	000000000000000000000000000000000000000000000000
1	000000000000000000000000000000000000000000000001
2	000000000000000000000000000000000000000*000001*0
3	0000000000000000000000000000000*00000**00001**01
4	00000000000000000000000*00000**0000***0*01***0*0
5	000000000000000*00000**0000***0*0******1******01
6	0000000*00000**0000***0*0********************1*0
7	00000**0000***0*0*****************************0*
8	000***0*0**************************************0
9	0***0*

(b) $n = 128$

Rounds	Left halves
0	0000000000000000000000000000000000000000000000000000000000000001
1	0000000000000000000000000000000000000000000000000000000*000001*0
2	00000000000000000000000000000000000000000000000*00000**00001**01
3	000000000000000000000000000000000000000*00000**0000***0*01***0*0
4	0000000000000000000000000000000*00000**0000***0*0******1******01
5	00000000000000000000000*00000**0000***0*0********************1*0
6	000000000000000*00000**0000***0*0*****************************01
7	0000000*00000**0000***0*0************************************0*0
8	00000**0000***0*0***0*
9	000***0*0**0
10	0***0*
11	**

Rounds	Right halves
0	0000000000000000000000000000000000000000000000000000000000000000
1	0000000000000000000000000000000000000000000000000000000000000001
2	0000000000000000000000000000000000000000000000000000000*000001*0
3	00000000000000000000000000000000000000000000000*00000**00001**01
4	000000000000000000000000000000000000000*00000**0000***0*01***0*0
5	0000000000000000000000000000000*00000**0000***0*0******1******01
6	00000000000000000000000*00000**0000***0*0********************1*0
7	000000000000000*00000**0000***0*0*****************************01
8	0000000*00000**0000***0*0************************************0*0
9	00000**0000***0*0***0*
10	000***0*0**0
11	0***0*

B

Cryptanalysis of PRESENT

B.1 Linear Hull Trail Counts

Table B.1 is the one determined by Ohkuma in [255], giving the number of trails in an optimal hull for T rounds with $T \in \{1, \ldots, 31\}$. Table B.2 gives values $\log_2 \beta$ such that $\Pr[|\mathsf{Corr}_{E_K}| \geq \beta] = \alpha$ for various α and number of rounds T.

B.2 6-round Deterministic Phase for PRESENT

By combining the 3-round deterministic phase of Section 2.2.5 with another 3 rounds appearing *before*, it is possible to construct a 6-round deterministic phase, reminiscent of the rebound approach described by Mendel, Rechberger, Schläffer, and Thomsen [225] and Lamberger, Mendel,

Table B.1: Number of trails λ_T in an optimal hull for T-round PRESENT, with $T \in \{1, \ldots, 31\}$

T	λ_T	T	λ_T	T	λ_T	T	λ_T
1	1	9	512	17	1140480	25	2517252696
2	1	10	1344	18	2985984	26	6590254272
3	1	11	3528	19	7817472	27	17253512704
4	3	12	9261	20	20466576	28	45170283840
5	9	13	24255	21	53582633	29	118257341400
6	27	14	63525	22	140281323	30	309601747125
7	72	15	166375	23	367261713	31	810547899975
8	192	16	435600	24	961504803		

Table B.2: Values $\log_2 \beta$ s.t. $\alpha = \Pr[|\mathrm{Corr}_{\mathcal{E}_K}| \geq \beta]$ for T-round PRESENT

	α												
T	0.01	0.05	0.10	0.20	0.30	0.40	0.50	0.60	0.70	0.80	0.90	0.95	0.99
1	−0.63	−1.03	−1.28	−1.64	−1.95	−2.25	−2.57	−2.93	−3.38	−3.98	−4.99	−6.00	−8.32
2	−2.63	−3.03	−3.28	−3.64	−3.95	−4.25	−4.57	−4.93	−5.38	−5.98	−6.99	−8.00	−10.32
3	−4.63	−5.03	−5.28	−5.64	−5.95	−6.25	−6.57	−6.93	−7.38	−7.98	−8.99	−10.00	−12.32
4	−5.84	−6.24	−6.49	−6.85	−7.16	−7.46	−7.78	−8.14	−8.58	−9.19	−10.20	−11.20	−13.53
5	−7.05	−7.44	−7.70	−8.06	−8.36	−8.66	−8.98	−9.35	−9.79	−10.40	−11.41	−12.41	−14.73
6	−8.26	−8.65	−8.90	−9.26	−9.57	−9.87	−10.19	−10.55	−11.00	−11.60	−12.61	−13.62	−15.94
7	−9.55	−9.94	−10.20	−10.56	−10.86	−11.16	−11.48	−11.85	−12.29	−12.90	−13.91	−14.91	−17.23
8	−10.84	−11.24	−11.49	−11.85	−12.16	−12.46	−12.78	−13.14	−13.58	−14.19	−15.20	−16.20	−18.53
9	−12.13	−12.53	−12.78	−13.14	−13.45	−13.75	−14.07	−14.43	−14.88	−15.48	−16.49	−17.50	−19.82
10	−13.44	−13.83	−14.09	−14.45	−14.75	−15.05	−15.37	−15.74	−16.18	−16.78	−17.80	−18.80	−21.12
11	−14.74	−15.14	−15.39	−15.75	−16.06	−16.36	−16.68	−17.04	−17.48	−18.09	−19.10	−20.10	−22.43
12	−16.05	−16.44	−16.69	−17.05	−17.36	−17.66	−17.98	−18.34	−18.79	−19.39	−20.40	−21.41	−23.73
13	−17.35	−17.75	−18.00	−18.36	−18.67	−18.97	−19.29	−19.65	−20.09	−20.70	−21.71	−22.71	−25.04
14	−18.66	−19.05	−19.30	−19.66	−19.97	−20.27	−20.59	−20.95	−21.40	−22.00	−23.01	−24.02	−26.34
15	−19.96	−20.36	−20.61	−20.97	−21.28	−21.58	−21.90	−22.26	−22.70	−23.31	−24.32	−25.32	−27.65
16	−21.27	−21.66	−21.92	−22.28	−22.58	−22.88	−23.20	−23.56	−24.01	−24.61	−25.63	−26.63	−28.95
17	−22.57	−22.97	−23.22	−23.58	−23.89	−24.19	−24.51	−24.87	−25.32	−25.92	−26.93	−27.93	−30.26
18	−23.88	−24.27	−24.53	−24.89	−25.19	−25.49	−25.81	−26.18	−26.62	−27.23	−28.24	−29.24	−31.56
19	−25.19	−25.58	−25.83	−26.19	−26.50	−26.80	−27.12	−27.48	−27.93	−28.53	−29.54	−30.55	−32.87
20	−26.49	−26.89	−27.14	−27.50	−27.80	−28.11	−28.42	−28.79	−29.23	−29.84	−30.85	−31.85	−34.17
21	−27.80	−28.19	−28.44	−28.80	−29.11	−29.41	−29.73	−30.09	−30.54	−31.14	−32.15	−33.16	−35.48
22	−29.10	−29.50	−29.75	−30.11	−30.42	−30.72	−31.04	−31.40	−31.84	−32.45	−33.46	−34.46	−36.79
23	−30.41	−30.80	−31.06	−31.42	−31.72	−32.02	−32.34	−32.71	−33.15	−33.75	−34.77	−35.77	−38.09
24	−31.71	−32.11	−32.36	−32.72	−33.03	−33.33	−33.65	−34.01	−34.46	−35.06	−36.07	−37.07	−39.40
25	−33.02	−33.41	−33.67	−34.03	−34.33	−34.63	−34.95	−35.32	−35.76	−36.37	−37.38	−38.38	−40.70
26	−34.33	−34.72	−34.97	−35.33	−35.64	−35.94	−36.26	−36.62	−37.07	−37.67	−38.68	−39.69	−42.01
27	−35.63	−36.03	−36.28	−36.64	−36.95	−37.25	−37.57	−37.93	−38.37	−38.98	−39.99	−40.99	−43.31
28	−36.94	−37.33	−37.58	−37.94	−38.25	−38.55	−38.87	−39.23	−39.68	−40.28	−41.30	−42.30	−44.62
29	−38.24	−38.64	−38.89	−39.25	−39.56	−39.86	−40.18	−40.54	−40.98	−41.59	−42.60	−43.60	−45.93
30	−39.55	−39.94	−40.20	−40.56	−40.86	−41.16	−41.48	−41.85	−42.29	−42.90	−43.91	−44.91	−47.23
31	−40.85	−41.25	−41.50	−41.86	−42.17	−42.47	−42.79	−43.15	−43.60	−44.20	−45.21	−46.22	−48.54

Schläffer, Rechberger, and Rijmen [204] (see Figure B.1). The idea is, that for rounds 3 through 5, the same approach as in Section 2.2.5 is used. Also, the same approach is used, but going in the other direction, for rounds 0 through 2.

This describes a construction to *independently* obtain (i) a set of *outputs* from round F_2, for which the inputs follow the trail over the *first* three rounds and (ii) a set of *inputs* to F_3 which follow the trail over the *last* three rounds. These two sets meet at the same point: right around the addition of the round key of round F_3. Thus, one can use said round key to determine a matching between the two sets, to obtain a set which has the desirable property of following the trail over both the top and bottom part. However, as the approaches are independent, there are constraints on the round key of round F_3 due to both parts, and this loss in degrees of freedom must be taken into account.

While the technique described here is not directly applicable with our model, as by nature it needs to use several *different* keys to match the two sets, it could potentially be useful in chosen-key models, which allow an adversary to make a statement using multiple different keys.

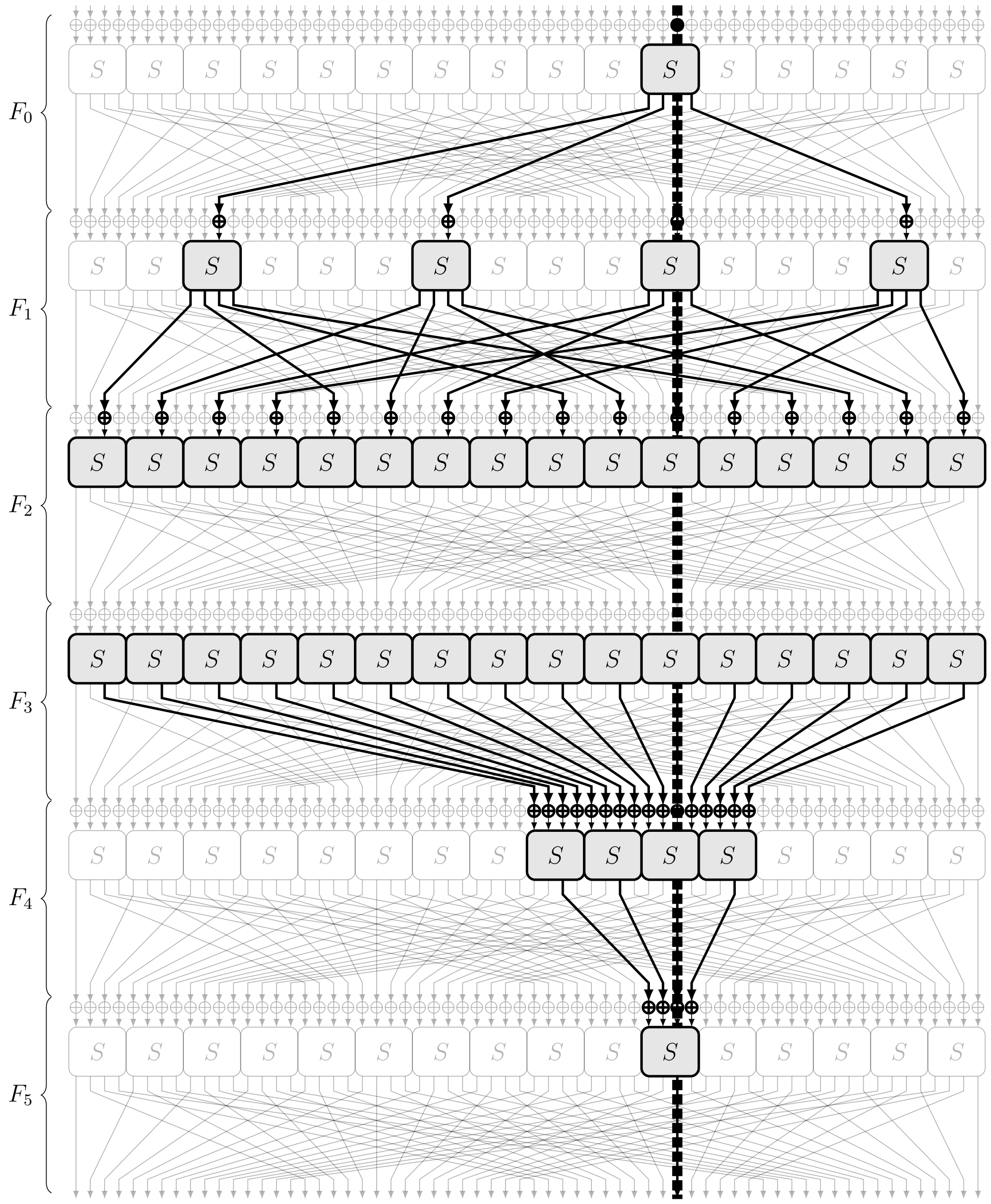


Figure B.1: 6-round deterministic phase for Present using the trail $(\mathbf{e}_{21}, \mathbf{e}_{21}, \mathbf{e}_{21}, \mathbf{e}_{21}, \mathbf{e}_{21}, \mathbf{e}_{21}, \mathbf{e}_{21})$

C

Analysis of Permutations in AES-like Ciphers

C.1 Optimality of the Black-Box Model

One has to make sure that the definition of the tightly guaranteed active S-boxes is independent of the concrete S-box functions within the AES-like ciphers. This is shown in Lemma 10.

Lemma 10. *Let $\theta : (\mathbb{F}_{2^m})^M \to (\mathbb{F}_{2^m})^M$ be a linear automorphism with branch number B_θ. Let $v = (v_1, \ldots, v_M) \in (\mathbb{F}_{2^m})^M \backslash \{0\}$ such that $\theta(v) = w = (w_1, \ldots, w_M)$. Then for all $a_1, \ldots, a_{2M} \in \mathbb{F}_{2^m} \backslash \{0\}$, one can construct a linear automorphism θ' with branch number B_θ such that $\theta'(a_1 v_1, \ldots, a_M v_M) = (a_{M+1} w_1, \ldots, a_{2M} w_M)$.*

Proof. Let $G = (I \mid A)$ be the generator matrix in standard form of the linear $[2M, M, B_\theta]_m$-code C corresponding to θ, and $\alpha_{i,j}, 1 \leq i, j \leq M$ denote the entry in row i and column j of the $M \times M$ sub-matrix A. Now one can construct an equivalent code C' with the same minimum distance by multiplying every column of G by non-zero scalars $a_1, \ldots, a_{2M}$ [295, p. 54–55]. In order to obtain a generator matrix $G' = (I \mid A')$ of C' in standard form, one scales the rows by the non-zero values $a_1^{-1}, \ldots, a_M^{-1}$, so we get

$$G' = \begin{pmatrix} 1 & 0 & \cdots & 0 & a_1^{-1} a_{M+1} \alpha_{1,M+1} & \cdots & a_1^{-1} a_{2M} \alpha_{1,2M} \\ 0 & 1 & \cdots & 0 & a_2^{-1} a_{M+1} \alpha_{2,M+1} & \cdots & a_2^{-1} a_{2M} \alpha_{2,2M} \\ \vdots & \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 1 & a_M^{-1} a_{M+1} \alpha_{M,M+1} & \cdots & a_M^{-1} a_{2M} \alpha_{M,2M} \end{pmatrix}. \tag{C.1}$$

As stated, G' generates a code C' equivalent to C, and in turn defines the new mixing $\theta'(X) = A' \cdot X$. If the matrix A was invertible, then A' is invertible as well since A' is obtained from A by scaling the rows and the columns. □

In order to prove Theorem 6, one will make use of the following two results.

Lemma 11. *Let* $\log_2(M+2) < m$ *and let* C *be a linear* $[2M, M, M+1]_m$*-code which is MDS. For every subset* $S \subseteq \{1, \ldots, 2M\}$ *with* $M+1 \leq \sharp S \leq 2M$*, there exists a vector* $v = (v_1, \ldots, v_{2M}) \in C$ *such that* $v_i \neq 0$ if and only if $i \in S$.

Proof. Define two subsets $V, W \subseteq S$ such that $\sharp V = \sharp W = M+1$ and $V \cup W = S$. This is possible since $\sharp S \geq M+1$. We use the fact that C, being an MDS code, has a code word of weight $M+1$ in any choice of $M+1$ coordinates (see e.g. [215, Chapter 11, Theorem 4]). Thus, there exists two vectors $v = (v_1, \ldots, v_{2M})$ and $w = (w_1, \ldots, w_{2M})$ in C such that $v_i \neq 0$ *if and only if* $i \in V$ and $w_i \neq 0$ *if and only if* $i \in W$. Now, one can construct u as a linear combination $u = v + cw$ with $c \in \mathbb{F}_{2^m}$ as follows. Choose $c \neq 0$ such that for all non-zero components v_i in v the identity

$$c \cdot w_i \neq -v_i \tag{C.2}$$

holds. This is possible because of the field property of $\mathbb{F}_{2^m}$ and since $2^m > M+2$. The vector u now has $\sharp S$ non-zero coordinates. □

Thus, given a concrete MDS transformation (which has a sufficiently large dimension), every activity pattern which fulfills the branch number property can be realized. By applying Lemma 10, one obtains the following result.

Corollary 3. *Let* $\log_2(M+2) < m$ *. Then for all* $v, w \in (\mathbb{F}_{2^m})^M$*, such that* $\mathsf{hw}(\tilde{v}) + \mathsf{hw}(\tilde{w}) \geq M+1$*, where* $\tilde{u}$ *denotes the activity pattern for* $u \in (\mathbb{F}_{2^m})^M$*, there exists an MDS matrix* $A' \in (\mathbb{F}_{2^m})^{M \times M}$ *such that* $w = A'v$.

C.2 Experimental Results

Tables C.1 through C.3 provide the results from our search for optimal rotation matrices. For $\rho \in \{1, 2, 3\}$ and a wide range of dimensions $M \times N$, number of rounds T and *some* trail-optimal choice of σ, we give the number of active S-boxes it tightly guarantees, denoted $\mathscr{B}$. We note that entries marked with † are results restricted to diffusion-optimal σ due to the complexity of the model. As such, the optimal bound with respect to trail weights may be even higher.

Table C.1: Results for $(M,N) \in \{(2,2),(2,4),(2,6),(2,8),(3,3)\}$

			$\rho=1$		$\rho=2$		$\rho=3$	
T	M	N	$\mathscr{B}$	σ	$\mathscr{B}$	σ	$\mathscr{B}$	σ
2	2	2	3	$(0,1)$	3	$(0,1),(0,1)$	3	$(0,1),(0,1),(0,1)$
3			5	$(0,1)$	5	$(0,1),(0,1)$	5	$(0,1),(0,1),(0,1)$
4			9	$(0,1)$	9	$(0,1),(0,1)$	9	$(0,1),(0,1),(0,1)$
5			10	$(0,1)$	10	$(0,1),(0,1)$	10	$(0,1),(0,1),(0,1)$
6			12	$(0,1)$	12	$(0,1),(0,1)$	12	$(0,1),(0,1),(0,1)$
7			14	$(0,1)$	14	$(0,1),(0,1)$	14	$(0,1),(0,1),(0,1)$
8			18	$(0,1)$	18	$(0,1),(0,1)$	18	$(0,1),(0,1),(0,1)$
10			21	$(0,1)$	21	$(0,1),(0,1)$	21	$(0,1),(0,1),(0,1)$
12			27	$(0,1)$	27	$(0,1),(0,1)$	27	$(0,1),(0,1),(0,1)$
2	2	4	3	$(0,1)$	3	$(0,1),(0,1)$	3	$(0,1),(0,1),(0,1)$
3			5	$(0,1)$	5	$(0,1),(0,1)$	5	$(0,1),(0,1),(0,1)$
4			9	$(0,1)$	9	$(0,1),(0,1)$	9	$(0,1),(0,1),(0,1)$
5			13	$(0,1)$	13	$(0,1),(0,1)$	13	$(0,1),(0,1),(0,1)$
6			18	$(0,1)$	18	$(0,1),(0,1)$	18	$(0,1),(0,1),(0,1)$
7			21	$(0,1)$	22	$(0,1),(0,2)$	21	$(0,1),(0,1),(0,1)$
8			24	$(0,1)$	24	$(0,1),(0,1)$	24	$(0,1),(0,1),(0,1)$
10			30	$(0,1)$	30	$(0,1),(0,1)$	30	$(0,1),(0,1),(0,1)$
12			36	$(0,1)$	36	$(0,1),(0,1)$	36	$(0,1),(0,1),(0,1)$
2	2	6	3	$(0,1)$	3	$(0,1),(0,1)$	3	$(0,1),(0,1),(0,1)$
3			5	$(0,1)$	5	$(0,1),(0,1)$	5	$(0,1),(0,1),(0,1)$
4			9	$(0,1)$	9	$(0,1),(0,1)$	9	$(0,1),(0,1),(0,1)$
5			13	$(0,1)$	13	$(0,1),(0,1)$	13	$(0,1),(0,1),(0,1)$
6			18	$(0,1)$	21	$(0,1),(0,2)$	21	$(0,1),(0,2),(0,2)$
7			21	$(0,1)$	29	$(0,2),(0,1)$	28	$(0,1),(0,2),(0,2)$
8			24	$(0,1)$	33	$(0,1),(0,2)$	33	$(0,1),(0,2),(0,1)$
10			30	$(0,1)$	39	$(0,1),(0,2)$	42	$(0,1),(0,1),(0,2)$
12			36	$(0,1)$	45	$(0,1),(0,2)$	48	$(0,1),(0,1),(0,3)$
2	2	8	3	$(0,1)$	3	$(0,1),(0,1)$	3	$(0,1),(0,1),(0,1)$
3			5	$(0,1)$	5	$(0,1),(0,1)$	5	$(0,1),(0,1),(0,1)$
4			9	$(0,1)$	9	$(0,1),(0,1)$	9	$(0,1),(0,1),(0,1)$
5			13	$(0,1)$	13	$(0,1),(0,1)$	13	$(0,1),(0,1),(0,1)$
6			18	$(0,1)$	21	$(0,1),(0,2)$	21	$(0,1),(0,2),(0,2)$
7			21	$(0,1)$	29	$(0,1),(0,3)$	29	$(0,1),(0,2),(0,3)$
8			24	$(0,1)$	39	$(0,1),(0,3)$	42	$(0,1),(0,2),(0,3)$
10			30	$(0,1)$	51	$(0,1),(0,3)$	54	$(0,1),(0,3),(0,2)$
12			36	$(0,1)$	56	$(0,1),(0,3)$	60	$(0,1),(0,2),(0,3)$
2	3	3	4	$(0,1,2)$	4	$(0,1,2),(0,1,2)$	4	$(0,1,2),(0,1,2),(0,1,2)$
3			7	$(0,1,2)$	7	$(0,1,2),(0,1,2)$	7	$(0,1,2),(0,1,2),(0,1,2)$
4			16	$(0,1,2)$	16	$(0,1,2),(0,1,2)$	16	$(0,1,2),(0,1,2),(0,1,2)$
5			17	$(0,1,2)$	17	$(0,1,2),(0,1,2)$	17	$(0,1,2),(0,1,2),(0,1,2)$
6			20	$(0,1,2)$	20	$(0,1,2),(0,1,2)$	20	$(0,1,2),(0,1,2),(0,1,2)$
7			23	$(0,1,2)$	23	$(0,1,2),(0,1,2)$	23	$(0,1,2),(0,1,2),(0,1,2)$
8			32	$(0,1,2)$	32	$(0,1,2),(0,1,2)$	32	$(0,1,2),(0,1,2),(0,1,2)$
10			36	$(0,1,2)$	36	$(0,1,2),(0,1,2)$	36	$(0,1,2),(0,1,2),(0,1,2)$
12			48	$(0,1,2)$	48	$(0,1,2),(0,1,2)$	48	$(0,1,2),(0,1,2),(0,1,2)$

Table C.2: Results for $(M,N) \in \{(3,6),(3,9),(4,4),(4,6),(4,8)\}$

			$\rho = 1$		$\rho = 2$		$\rho = 3$	
T	M	N	$\mathscr{B}$	σ	$\mathscr{B}$	σ	$\mathscr{B}$	σ
2	3	6	4	$(0,1,2)$	4	$(0,1,2),(0,1,2)$	4	$(0,1,2),(0,1,2),(0,1,2)$
3			7	$(0,1,2)$	7	$(0,1,2),(0,1,2)$	7	$(0,1,2),(0,1,2),(0,1,2)$
4			16	$(0,1,2)$	16	$(0,1,2),(0,1,2)$	16	$(0,1,2),(0,1,2),(0,1,2)$
5			20	$(0,1,2)$	25	$(0,1,2),(0,1,3)$	25	$(0,1,2),(0,1,3),(0,1,2)$
6			24	$(0,1,2)$	36	$(0,1,2),(0,1,3)$	36	$(0,1,3),(0,1,2),(0,2,3)$
7			28	$(0,1,2)$	38	$(0,1,2),(0,1,3)$	40	$(0,1,3),(0,2,3),(0,1,2)$
8			32	$(0,1,2)$	41	$(0,1,2),(0,1,3)$	44	$(0,1,2),(0,1,3),(0,2,3)$
10			40	$(0,1,2)$	56	$(0,1,2),(0,1,3)$	56	$(0,1,2),(0,1,3),(0,2,3)$
12			48	$(0,1,2)$	72	$(0,1,2),(0,1,3)$	72	$(0,1,2),(0,1,3),(0,2,3)$
2	3	9	4	$(0,1,2)$	4	$(0,1,2),(0,1,2)$	4	$(0,1,2),(0,1,2),(0,1,2)$
3			7	$(0,1,2)$	7	$(0,1,2),(0,1,2)$	7	$(0,1,2),(0,1,2),(0,1,2)$
4			16	$(0,1,2)$	16	$(0,1,2),(0,1,2)$	16	$(0,1,2),(0,1,2),(0,1,2)$
5			25	$(0,1,3)$	25	$(0,1,2),(0,1,3)$	25	$(0,4,8),(0,4,8),(0,2,8)$
6			36	$(0,1,3)$	44	$(0,1,2),(0,2,5)$	44	$(0,1,2),(0,2,4),(0,3,6)$
7			42	$(0,1,3)$	53	$(0,1,2),(0,2,5)$	55	$(0,1,2),(0,2,4),(0,3,6)$
8			48	$(0,1,3)$	60	$(0,1,2),(0,1,4)$	60	$(0,1,2),(0,1,2),(0,2,5)$
10			60	$(0,1,3)$	69	$(0,1,2),(0,1,4)$		
12			72	$(0,1,3)$	92	$(0,1,2),(0,2,5)$	93	$(0,1,2),(0,2,4),(0,3,6)$
2	4	4	5	$(0,1,2,3)$	5	$(0,1,2,3),(0,1,2,3)$	5	$(0,1,2,3),(0,1,2,3),(0,1,2,3)$
3			9	$(0,1,2,3)$	9	$(0,1,2,3),(0,1,2,3)$	9	$(0,1,2,3),(0,1,2,3),(0,1,2,3)$
4			25	$(0,1,2,3)$	25	$(0,1,2,3),(0,1,2,3)$	25	$(0,1,2,3),(0,1,2,3),(0,1,2,3)$
5			26	$(0,1,2,3)$	26	$(0,1,2,3),(0,1,2,3)$	26	$(0,1,2,3),(0,1,2,3),(0,1,2,3)$
6			30	$(0,1,2,3)$	30	$(0,1,2,3),(0,1,2,3)$	30	$(0,1,2,3),(0,1,2,3),(0,1,2,3)$
7			34	$(0,1,2,3)$	34	$(0,1,2,3),(0,1,2,3)$	34	$(0,1,2,3),(0,1,2,3),(0,1,2,3)$
8			50	$(0,1,2,3)$	50	$(0,1,2,3),(0,1,2,3)$	50	$(0,1,2,3),(0,1,2,3),(0,1,2,3)$
10			55	$(0,1,2,3)$	55	$(0,1,2,3),(0,1,2,3)$	55	$(0,1,2,3),(0,1,2,3),(0,1,2,3)$
12			75	$(0,1,2,3)$	75	$(0,1,2,3),(0,1,2,3)$	75	$(0,1,2,3),(0,1,2,3),(0,1,2,3)$
2	4	6	5	$(0,1,2,3)$	5	$(0,1,2,3),(0,1,2,3)$	5	$(0,1,2,3),(0,1,2,3),(0,1,2,3)$
3			9	$(0,1,2,3)$	9	$(0,1,2,3),(0,1,2,3)$	9	$(0,1,2,3),(0,1,2,3),(0,1,2,3)$
4			25	$(0,1,2,3)$	25	$(0,1,2,3),(0,1,2,3)$	25	$(0,1,2,3),(0,1,2,3),(0,1,2,3)$
5			34	$(0,1,2,3)$	36	$(0,1,2,4),(0,1,2,3)$	37	$(0,1,2,3),(0,1,2,4),(0,1,3,4)$
6			45	$(0,1,3,4)$	45	$(0,1,3,4),(0,1,3,4)$	45	$(0,1,3,4),(0,1,3,4),(0,1,3,4)$
7			48	$(0,1,3,4)$	48	$(0,1,3,4),(0,1,3,4)$	48	$(0,1,2,3),(0,1,2,3),(0,1,3,4)$
8			57	$(0,1,3,4)$	57	$(0,1,3,4),(0,1,3,4)$	57	$(0,1,3,4),(0,1,3,4),(0,1,3,4)$
10			72	$(0,1,2,3)$	73	$(0,1,2,3),(0,1,2,4)$	74	$(0,1,2,3),(0,1,2,3),(0,1,3,4)$
12			90	$(0,1,3,4)$	90	$(0,1,3,4),(0,1,3,4)$	90	$(0,1,3,4),(0,1,3,4),(0,1,3,4)$
2	4	8	5	$(0,1,2,3)$	5	$(0,1,2,3),(0,1,2,3)$	5	$(0,1,2,3),(0,1,2,3),(0,1,2,3)$
3			9	$(0,1,2,3)$	9	$(0,1,2,3),(0,1,2,3)$	9	$(0,1,2,3),(0,1,2,3),(0,1,2,3)$
4			25	$(0,1,2,3)$	25	$(0,1,2,3),(0,1,2,3)$	25	$(0,1,2,3),(0,1,2,3),(0,1,2,3)$
5			41	$(0,1,2,4)$	41	$(0,1,2,3),(0,1,3,4)$		
6			50	$(0,1,2,4)$	55	$(0,1,2,3),(0,1,3,5)$		
7			58	$(0,1,3,4)$	58	$(0,1,2,3),(0,2,3,5)$		
8			65	$(0,1,2,4)$	65	$(0,1,2,3),(0,1,3,4)$		
10			85	$(0,1,2,4)$	90	$(0,1,2,3),(0,2,3,5)$		
12			105	$(0,1,2,4)$	111	$(0,1,2,3),(0,2,3,5)$		
14			120	$(0,1,2,4)$				

Table C.3: Results for $(M,N) \in \{(4,10),(4,12),(4,16),(4,32),(5,8)\}$

			$\rho=1$		$\rho=2$	
T	M	N	$\mathscr{B}$	σ	$\mathscr{B}$	σ
2	4	10	5	$(0,1,2,3)$	5	$(0,1,2,3),(0,1,2,3)$
3			9	$(0,1,2,3)$	9	$(0,1,2,3),(0,1,2,3)$
4			25	$(0,1,2,3)$	25	$(0,1,2,3),(0,1,2,3)$
5			41	$(0,1,2,4)$	41	$(0,1,2,3),(0,1,3,4)$
6			60	$(0,1,2,4)$	65	$(0,1,2,3),(0,1,4,7)$
7			70	$(0,1,3,4)$	72	$(0,1,2,3),(0,1,4,7)$
8			80	$(0,1,3,4)$	82	$(0,1,5,6),(0,2,5,7)$
2	4	12	5	$(0,1,2,3)$	5	$(0,1,2,3),(0,1,2,3)$
3			9	$(0,1,2,4)$	9	$(0,1,2,3),(0,1,2,4)$
4			25	$(0,1,2,3)$		
5			41	$(0,1,3,4)$		
6			65	$(0,1,4,5)$		
7			76	$(0,1,4,5)$		
8			92	$(0,1,4,5)$		
2	4	16	5	$(0,1,2,3)$	5	$(0,1,2,3),(0,1,2,3)$
3			9	$(0,1,2,3)$	9	$(0,1,2,3),(0,1,2,3)$
4			25	$(0,1,2,3)$	$25^{\dagger}$	$(0,1,2,3),(0,1,2,3)$
5			41	$(0,1,2,4)$		
6			75	$(0,1,4,6)$	$90^{\dagger}$	$(0,4,10,14),(0,2,11,13)$
7			100	$(0,1,4,6)$	$111^{\dagger}$	$(0,1,2,3),(0,3,7,11)$
8			120	$(0,1,4,5)$		
2	4	32	5	$(0,1,2,3)$	$5^{\dagger}$	$(0,1,2,3),(0,1,2,3)$
3			9	$(0,1,2,3)$	$9^{\dagger}$	$(0,1,2,3),(0,1,2,3)$
4			25	$(0,1,2,3)$	$25^{\dagger}$	$(0,1,2,3),(0,1,2,3)$
5			41	$(0,1,2,4)$		
6			75	$(0,1,4,6)$		
2	5	8	6	$(0,1,2,3,4)$	6	$(0,1,2,3,4),(0,1,2,3,4)$
3			11	$(0,1,2,3,4)$	11	$(0,1,2,3,4),(0,1,2,3,4)$
4			36	$(0,1,2,3,4)$	36	$(0,1,2,3,4),(0,1,2,3,4)$
5			54	$(0,1,2,3,5)$	56	$(0,1,2,3,4),(0,1,3,5,6)$
6			62	$(0,1,2,3,5)$	62	$(0,1,2,3,4),(0,1,2,3,5)$
7			67	$(0,1,2,3,5)$	67	$(0,1,2,3,4),(0,1,2,3,5)$
8			72	$(0,1,2,3,4)$	72	$(0,1,2,3,4),(0,1,2,3,4)$
9					95	$(0,1,2,3,4),(0,1,2,3,5)$

D

Pseudo-code for PRØST Proposals

In the following, we give pseudo-code for all PRØST proposals. We refer to Section 3.2 for the notation. Algorithms 12 and 13 describe the application and removal of padding. The PRØST-COPA-n proposal is described by Algorithms 14 through 16; PRØST-OTR-n is described by Algorithms 17 through 21; and PRØST-APE-$n[r, c]$ is described by Algorithms 22 and 23.

```
Algorithm 12: $\mathsf{pad}_b(X)$
  Data: $X \in \mathbb{F}_2^*$, $b \in \mathbb{N}$
1 return $X \| 10^{b-(|X| \bmod b)-1}$
```

```
Algorithm 13: StripPadding($X$)
  Data: $X \in \mathbb{F}_2^*$
1 while $X \neq \varepsilon$ and $\mathsf{lsb}_1(X) = 0$ do
2     $X = X \gg 1$                         // Peel off zeroes in the end
3 end
4 if $X \neq \varepsilon$ then
5     $X \leftarrow X \gg 1$                // If $X \neq \varepsilon$ remove the 1-bit
6 end
7 return $X$
```

Algorithm 14: PRØST-COPA-n-$\mathcal{E}(K,N,A,M)$

```
Data: $K \in \mathbb{F}_2^\kappa$, $N \in \mathbb{F}_2^\eta$, $A \in \mathbb{F}_2^*$, $M \in \mathbb{F}_2^*$
$(L,V) \leftarrow$ PRØST-COPA-$n$-ProcessAD$(A,N)$          // First process A and N
$M \leftarrow \mathsf{pad}_{2n}(M)$          // Pad M to positive multiple of 2n bits
$V \leftarrow V \oplus L$
$(\Delta_0,\Delta_1) \leftarrow (3L, 2L)$
$\Sigma \leftarrow 0^{2n}$
for $i = 1,\ldots,\ell$ do
    $V \leftarrow \tilde{P}_{n,K}(M_i \oplus \Delta_0) \oplus V$          // Process message blocks
    $C_i \leftarrow \tilde{P}_{n,K}(V) \oplus \Delta_1$
    $(\Delta_0,\Delta_1) \leftarrow (2\Delta_0, 2\Delta_1)$
    $\Sigma \leftarrow \Sigma \oplus M_i$
end
$T \leftarrow \mathsf{lsb}_\tau(\tilde{P}_{n,K}(\tilde{P}_{n,K}(\Sigma \oplus 2^{\ell-1}3^2L) \oplus V) \oplus 2^{\ell-1}7L)$          // Compute tag
return $(C,T)$
```

Algorithm 15: PRØST-COPA-n-$\mathcal{D}(K,N,A,C,T)$

```
Data: $K \in \mathbb{F}_2^\kappa$, $N \in \mathbb{F}_2^\eta$, $A \in \mathbb{F}_2^*$, $C \in (\mathbb{F}_2^{2n})^+$, $T \in \mathbb{F}_2^\tau$
$(L,V) \leftarrow$ PRØST-COPA-$n$-ProcessAD$(A,N)$          // First process A and N
$V \leftarrow V \oplus L$
$(\Delta_0,\Delta_1) \leftarrow (3L, 2L)$
$\Sigma \leftarrow 0^{2n}$
for $i = 1,\ldots,\ell$ do
    $V' \leftarrow \tilde{P}^{-1}_{n,K}(C_i \oplus \Delta_1)$          // Process ciphertext blocks
    $M_i \leftarrow \tilde{P}^{-1}_{n,K}(V' \oplus V) \oplus \Delta_0$
    $V \leftarrow V'$
    $(\Delta_0,\Delta_1) \leftarrow (2\Delta_0, 2\Delta_1)$
    $\Sigma \leftarrow \Sigma \oplus M_i$
end
$T' \leftarrow \mathsf{lsb}_\tau(\tilde{P}_{n,K}(\tilde{P}_{n,K}(\Sigma \oplus 2^{\ell-1}3^2L) \oplus V) \oplus 2^{\ell-1}7L)$          // Compute verification tag
if $T' = T$ then
    return StripPadding$(M)$
else
    return $\perp$
end
```

Algorithm 16: PRØST-COPA-n-ProcessAD(A,N)

```
Data: $A \in \mathbb{F}_2^*$, $N \in \mathbb{F}_2^\eta$
$X \leftarrow A \| N$                                         // Append $N$ to $A$
$L \leftarrow \tilde{P}_{n,K}(0^{2n})$                        // Set initial values
$\Delta \leftarrow 3^3 L$
$V \leftarrow 0^{2n}$
$k \leftarrow \left\lceil \frac{|X|}{2n} \right\rceil$
for $i = 1, \ldots, k-1$ do
    $V \leftarrow V \oplus \tilde{P}_{n,K}(X_i \oplus \Delta)$      // Process all blocks of $X$ except the last
    $\Delta \leftarrow 2\Delta$
end
if $|X_k| = 2n$ then
    $V \leftarrow \tilde{P}_{n,K}(V \oplus X_k \oplus 3\Delta)$     // Process last block of $X$ if full
else
    $V \leftarrow \tilde{P}_{n,K}(V \oplus \mathsf{pad}_{2n}(X_k) \oplus 3^2\Delta)$   // Process last block of $X$ if partial
end
return $(L,V)$
```

Algorithm 17: PRØST-OTR-n-$\mathscr{E}(K,N,A,M)$

```
Data: $K \in \mathbb{F}_2^\kappa$, $N \in \mathbb{F}_2^\eta$, $A \in \mathbb{F}_2^*$, $M \in \mathbb{F}_2^*$
$(C, TE) \leftarrow$ PRØST-OTR-$n$-Enc$(K,N,M)$
if $A \neq \varepsilon$ then  $TA \leftarrow$ PRØST-OTR-$n$-ProcessAD$(K,A)$
else  $TA \leftarrow 0^{2n}$
$T \leftarrow \mathsf{msb}_\tau(TE \oplus TA)$
return $(C,T)$
```

Algorithm 18: PRØST-OTR-n-$\mathscr{D}(K,N,A,C,T)$

```
Data: $K \in \mathbb{F}_2^\kappa$, $N \in \mathbb{F}_2^\eta$, $A \in \mathbb{F}_2^*$, $C \in (\mathbb{F}_2^{2n})^+$, $T \in \mathbb{F}_2^\tau$
$(M, TE) \leftarrow$ PRØST-OTR-$n$-Dec$(K,N,C)$
if $A \neq \varepsilon$ then  $TA \leftarrow$ PRØST-OTR-$n$-ProcessAD$(K,A)$
else  $TA \leftarrow 0^{2n}$
$T' \leftarrow \mathsf{msb}_\tau(TE \oplus TA)$
if $T' = T$ then
    return StripPadding$(M)$
else
    return $\perp$
end
```

Algorithm 19: PRØST-OTR-n-ProcessAD(K, A)

```
Data: $K \in \mathbb{F}_2^\kappa, A \in \mathbb{F}_2^*$
$\Xi \leftarrow 0^{2n}$                                        // Set initial values
$\gamma \leftarrow \tilde{P}_{n,K}(0^{2n})$
$Q \leftarrow 4\gamma$
for $i = 1, \ldots, k-1$ do
    $\Xi \leftarrow \Xi \oplus \tilde{P}_{n,K}(Q \oplus A_i)$          // Process first $k-1$ blocks of $A$
    $Q \leftarrow 2Q$
end
if $|A_k| = 2n$ then
    $\Xi \leftarrow \Xi \oplus A_k$
    $TA \leftarrow \tilde{P}_{n,K}(Q \oplus 2\gamma \oplus \Xi)$
else
    $\Xi \leftarrow \Xi \oplus \mathsf{pad}_{2n}(A_k)$
    $TA \leftarrow \tilde{P}_{n,K}(Q \oplus \gamma \oplus \Xi)$
end
return $TA$
```

Algorithm 20: PRØST-OTR-n-Enc(K, N, M)

```
Data: $K \in \mathbb{F}_2^\kappa$, $N \in \mathbb{F}_2^\eta$, $M \in \mathbb{F}_2^*$
$M \leftarrow \mathsf{pad}_{2n}(M)$                         // Pad $M$ to positive multiple of $2n$ bits
$\Sigma \leftarrow 0^{2n}$                                   // Set initial values
$\delta \leftarrow \tilde{P}_{n,K}(\mathsf{pad}_{2n}(N))$         // Set $\delta$ to the encryption of the padded nonce
$L \leftarrow 4\delta$
for $i = 1, \ldots, \lfloor \ell/2 \rfloor$ do
    $C_{2i-1} \leftarrow \tilde{P}_{n,K}(L \oplus M_{2i-1}) \oplus M_{2i}$        // Process $M$ blocks in pairs
    $C_{2i} \leftarrow \tilde{P}_{n,K}(L \oplus \delta \oplus C_{2i-1}) \oplus M_{2i-1}$
    $\Sigma \leftarrow \Sigma \oplus M_{2i}$
    $L \leftarrow 2L$
end
if $\ell$ is even then
    $L' \leftarrow L \oplus \delta$
else
    $L' \leftarrow L$                                       // Handle last block if $\ell$ is odd
    $C_\ell \leftarrow \tilde{P}_{n,K}(L') \oplus M_\ell$
    $\Sigma \leftarrow \Sigma \oplus M_\ell$
end
$TE \leftarrow \tilde{P}_{n,K}(3L' \oplus \delta \oplus \Sigma)$           // Compute encryption part of tag
return $(C, TE)$
```

Algorithm 21: PRØST-OTR-n-Dec(K,N,C)

```
   Data: $K \in \mathbb{F}_2^*$, $N \in \mathbb{F}_2^\eta$, $C \in (\mathbb{F}_2^{2n})^+$
1  $\Sigma \leftarrow 0^{2n}$                                    // Set initial values
2  $\delta \leftarrow \tilde{P}_{n,K}(\mathsf{pad}_{2n}(N))$        // Set $\delta$ to the encryption of the padded nonce
3  $L \leftarrow 4\delta$
4  for $i = 1,\ldots,\lfloor \ell/2 \rfloor$ do
5      $M_{2i-1} \leftarrow \tilde{P}_{n,K}(L \oplus \delta \oplus C_{2i-1}) \oplus C_{2i}$      // Process $C$ blocks in pairs
6      $M_{2i} \leftarrow \tilde{P}_{n,K}(L \oplus M_{2i-1}) \oplus C_{2i-1}$
7      $\Sigma \leftarrow \Sigma \oplus M_{2i}$
8      $L \leftarrow 2L$
9  end
10 if $\ell$ is even then
11     $L' \leftarrow L \oplus \delta$
12 else
13     $L' \leftarrow L$                                      // Handle last block if $\ell$ is odd
14     $M_\ell \leftarrow \tilde{P}_{n,K}(L') \oplus C_\ell$
15     $\Sigma \leftarrow \Sigma \oplus M_\ell$
16 end
17 $TE \leftarrow \tilde{P}_{n,K}(3L' \oplus \delta \oplus \Sigma)$              // Compute encryption part of tag
18 return $(M, TE)$
```

Algorithm 22: PRØST-APE-$n[r,c]$-$\mathcal{E}(K,N,A,M)$

```
   Data: $K \in \mathbb{F}_2^\kappa$, $N \in \mathbb{F}_2^\eta$, $A \in \mathbb{F}_2^*$, $M \in \mathbb{F}_2^*$
1  $M \leftarrow \mathsf{pad}_r(M)$                          // Pad $M$ to positive multiple of $r$ bits
2  $V \leftarrow 0^r \| K$
3  $X \leftarrow \mathsf{pad}_r(N\|A)$          // Prepend $N$ to $A$ and pad to positive multiple of $r$ bits
4  for $i = 1,\ldots,k$ do
5      $V \leftarrow P_n((X_i \oplus V_r)\|V_c)$                         // Process nonce and AD
6  end
7  $V_c \leftarrow V_c \oplus (0^{c-1}\|1)$                               // Domain separation
8  for $i = 1,\ldots,\ell$ do
9      $V \leftarrow P_n((M_i \oplus V_r)\|V_c)$                             // Process message
10     $C_i \leftarrow V_r$
11 end
12 $T \leftarrow V_c \oplus K$                                            // Compute tag
13 return $(C,T)$
```

Algorithm 23: PRØST-APE-$n[r,c]$-$\mathscr{D}(K,N,A,C,T)$

```
Data: $K \in \mathbb{F}_2^\kappa$, $N \in \mathbb{F}_2^\eta$, $A \in \mathbb{F}_2^*$, $C \in (\mathbb{F}_2^r)^+$, $T \in \mathbb{F}_2^\tau$
$IV \leftarrow 0^r \| K$
$X \leftarrow \mathsf{pad}_r(N\|A)$          // Prepend $N$ to $A$ and pad to positive multiple of $r$ bits
for $i = 1, \ldots, k$ do
    $IV \leftarrow P_n((X_i \oplus V_r)\|V_c)$          // Process nonce and AD
end
$IV_c \leftarrow IV_c \oplus (0^{c-1}\|1)$          // Domain separation
$C_0 \leftarrow IV_r$          // Set dummy $C_0 = IV_r$ for a smoother loop
$V \leftarrow (0^r\|(K \oplus T))$
for $i = \ell, \ldots, 1$ do
    $V \leftarrow P_n^{-1}(C_i\|V_c)$          // Process ciphertext
    $M_i \leftarrow V_r \oplus C_{i-1}$
end
if $IV_c = V_c$ then
    return StripPadding($M$)
else
    return $\perp$
end
```

Bibliography

[1] Donald Eastlake 3rd. Transport Layer Security (TLS) Extensions: Extension Definitions. RFC 6066 (Proposed Standard), January 2011.

[2] Mohamed Ahmed Abdelraheem, Javad Alizadeh, Hoda A. Alkhzaimi, Mohammad Reza Aref, Nasour Bagheri, Praveen Gauravaram, and Martin M. Lauridsen. Improved Linear Cryptanalysis of Reduced-round SIMON. Cryptology ePrint Archive, Report 2014/681, 2014.

[3] Mohamed Ahmed Abdelraheem, Peter Beelen, Andrey Bogdanov, and Elmar Tischhauser. Twisted Polynomials and Forgery Attacks on GCM. In Oswald and Fischlin [256], pages 762–786.

[4] Masayuki Abe, editor. *Advances in Cryptology - ASIACRYPT 2010 - 16th International Conference on the Theory and Application of Cryptology and Information Security, Singapore, December 5-9, 2010. Proceedings*, volume 6477 of *Lecture Notes in Computer Science*, 2010. Springer.

[5] Farzaneh Abed, Eik List, Stefan Lucks, and Jakob Wenzel. Differential and Linear Cryptanalysis of Reduced-Round Simon. Cryptology ePrint Archive, Report 2013/526, 2013.

[6] Farzaneh Abed, Scott R. Fluhrer, Christian Forler, Eik List, Stefan Lucks, David A. McGrew, and Jakob Wenzel. Pipelineable On-line Encryption. In Cid and Rechberger [96], pages 205–223.

[7] Farzaneh Abed, Eik List, Stefan Lucks, and Jakob Wenzel. Differential Cryptanalysis of Round-Reduced Simon and Speck. In Cid and Rechberger [96], pages 525–545.

[8] National Security Agency. Suite B Cryptography. http://www.nsa.gov/ia/programs/suiteb_cryptography/. Accessed on May 19, 2015.

[9] Martin Ågren, Martin Hell, Thomas Johansson, and Willi Meier. Grain-128a: a new version of Grain-128 with optional authentication. *IJWMC*, 5(1):48–59, 2011.

[10] Zahra Ahmadian, Sahram Rasoolzadeh, Mahmoud Salmasizadeh, and Mohammad Reza Aref. Automated Dynamic Cube Attack on Block Ciphers: Cryptanalysis of SIMON and KATAN. Cryptology ePrint Archive, Report 2015/040, 2015.

[11] Kahraman Akdemir, Martin Dixon, Wajdi Feghali, Patrick Fay, Vinodh Gopal, Jim Guilford, Erdinc Ozturk, Gil Wolrich, and Ronen Zohar. *Breakthrough AES Performance with Intel AES New Instructions*. Intel Corporation, 2010.

[12] Mehdi-Laurent Akkar and Christophe Giraud. An Implementation of DES and AES, Secure against Some Attacks. In Çetin Kaya Koç, David Naccache, and Christof Paar, editors, *Cryptographic Hardware and Embedded Systems - CHES 2001, Third International Workshop, Paris, France, May 14-16, 2001, Proceedings*, volume 2162 of *Lecture Notes in Computer Science*, pages 309–318. Springer, 2001.

[13] Martin R. Albrecht and Gregor Leander. An All-In-One Approach to Differential Cryptanalysis for Small Block Ciphers. In Lars R. Knudsen and Huapeng Wu, editors, *Selected Areas in Cryptography, 19th International Conference, SAC 2012, Windsor, ON, Canada, August 15-16, 2012, Revised Selected Papers*, volume 7707 of *Lecture Notes in Computer Science*, pages 1–15. Springer, 2012.

[14] Nadhem J. AlFardan and Kenneth G. Paterson. Lucky Thirteen: Breaking the TLS and DTLS Record Protocols. In *2013 IEEE Symposium on Security and Privacy, SP 2013, Berkeley, CA, USA, May 19-22, 2013*, pages 526–540. IEEE Computer Society, 2013.

[15] Javad Alizadeh, Nasour Bagheri, Praveen Gauravaram, Abhishek Kumar, and Somitra Kumar Sanadhya. Linear Cryptanalysis of Round Reduced SIMON. Cryptology ePrint Archive, Report 2013/663, 2013.

[16] Javad Alizadeh, Hoda A. AlKhzaimi, Mohammad Reza Aref, Nasour Bagheri, Praveen Gauravaram, Abhishek Kumar, Martin M. Lauridsen, and Somitra Kumar Sanadhya. Cryptanalysis of SIMON Variants with Connections. In Nitesh Saxena and Ahmad-Reza Sadeghi, editors, *Radio Frequency Identification: Security and Privacy Issues - 10th International Workshop, RFIDSec 2014, Oxford, UK, July 21-23, 2014, Revised Selected Papers*, volume 8651 of *Lecture Notes in Computer Science*, pages 90–107. Springer, 2014.

[17] Javad Alizadeh, Mohammad Reza Aref, and Nasour Bagheri. Artemia. Submission to the CAESAR competition, 2014.

[18] Hoda A. Alkhzaimi and Martin M. Lauridsen. SIMON Cryptanalysis Code Repository. https://github.com/mmeh/simon-speck-cryptanalysis, 2013.

[19] Hoda A. Alkhzaimi and Martin M. Lauridsen. Cryptanalysis of the SIMON Family of Block Ciphers. Cryptology ePrint Archive, Report 2013/543, 2013.

[20] Basel Alomair. AVALANCHEv1. Submission to the CAESAR competition, 2014.

[21] Elena Andreeva, Andrey Bogdanov, Atul Luykx, Bart Mennink, Elmar Tischhauser, and Kan Yasuda. Parallelizable and Authenticated Online Ciphers. In Kazue Sako and Palash Sarkar, editors, *Advances in Cryptology - ASIACRYPT 2013 - 19th International Conference on the Theory and Application of Cryptology and Information Security, Bengaluru, India, December 1-5, 2013, Proceedings, Part I*, volume 8269 of *Lecture Notes in Computer Science*, pages 424–443. Springer, 2013.

[22] Elena Andreeva, Andrey Bogdanov, and Bart Mennink. Towards Understanding the Known-Key Security of Block Ciphers. In Moriai [236], pages 348–366.

[23] Elena Andreeva, Begül Bilgin, Andrey Bogdanov, Atul Luykx, Florian Mendel, Bart Mennink, Nicky Mouha, Qingju Wang, and Kan Yasuda. PRIMATEs. Submission to the CAESAR competition, 2014.

[24] Elena Andreeva, Begül Bilgin, Andrey Bogdanov, Atul Luykx, Bart Mennink, Nicky Mouha, and Kan Yasuda. APE: Authenticated Permutation-Based Encryption for Lightweight Cryptography. In Cid and Rechberger [96], pages 168–186.

[25] 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 [279], pages 105–125.

[26] Elena Andreeva, Atul Luykx, Bart Mennink, and Kan Yasuda. COBRA: A Parallelizable Authenticated Online Cipher Without Block Cipher Inverse. In Cid and Rechberger [96], pages 187–204.

[27] Kazumaro Aoki, Tetsuya Ichikawa, Masayuki Kanda, Mitsuru Matsui, Shiho Moriai, Junko Nakajima, and Toshio Tokita. Camellia: A 128-Bit Block Cipher Suitable for Multiple Platforms - Design and Analysis. In Douglas R. Stinson and Stafford E. Tavares, editors, *Selected Areas in Cryptography, 7th Annual International Workshop, SAC 2000, Waterloo, Ontario, Canada, August 14-15, 2000, Proceedings*, volume 2012 of *Lecture Notes in Computer Science*, pages 39–56. Springer, 2000.

[28] Kazumaro Aoki, Tetsu Iwata, and Kan Yasuda. How Fast Can a Two-Pass Mode Go? A Parallel Deterministic Authenticated Encryption Mode for AES-NI. In *DIAC 2012: Directions in Authenticated Ciphers*, 2012.

[29] Jacob Appelbaum. Conflicting roles: the NSA and cryptography. Invited talk at the Fast Software Encryption workshop, 2015.

[30] Frederik Armknecht. Improving Fast Algebraic Attacks. In Roy and Meier [274], pages 65–82.

[31] Tomer Ashur. Improved Linear Trails for the Block Cipher Simon. Cryptology ePrint Archive, Report 2015/285, 2015.

[32] Armen S. Asratian, Tristan M. J. Denley, and Roland Häggkvist. *Bipartite Graphs and Their Applications*. Cambridge Tracts in Mathematics. Cambridge University Press, 1998.

[33] Vijayalakshmi Atluri, editor. *Proceedings of the 9th ACM Conference on Computer and Communications Security, CCS 2002, Washington, DC, USA, November 18-22, 2002*, 2002. ACM.

[34] Jean-Philippe Aumasson, Philipp Jovanovic, and Samuel Neves. NORX. Submission to the CAESAR competition, 2014.

[35] Steve Babbage and Laurent Frisch. On MISTY1 Higher Order Differential Cryptanalysis. In Dongho Won, editor, *Information Security and Cryptology - ICISC 2000, Third International Conference, Seoul, Korea, December 8-9, 2000, Proceedings*, volume 2015 of *Lecture Notes in Computer Science*, pages 22–36. Springer, 2000.

[36] Lear Bahack. Julius. Submission to the CAESAR competition, 2014.

[37] Achiya Bar-On. Improved Higher-Order Differential Attacks on MISTY1. In Leander [210], pages 28–47.

[38] Paulo Barreto and Vincent Rijmen. The Anubis Block Cipher. Submission to the NESSIE project, 2000.

[39] Paulo Barreto and Vincent Rijmen. The Whirlpool Hash Function. Submission to the NESSIE project, 2000.

[40] Ray Beaulieu, Douglas Shors, Jason Smith, Stefan Treatman-Clark, Bryan Weeks, and Louis Wingers. The SIMON and SPECK Families of Lightweight Block Ciphers. Cryptology ePrint Archive, Report 2013/404, 2013.

[41] Ray Beaulieu, Douglas Shors, Jason Smith, Stefan Treatman-Clark, Bryan Weeks, and Louis Wingers. SIMON and SPECK: Block Ciphers for the Internet of Things. Cryptology ePrint Archive, Report 2015/585, 2015.

[42] Christof Beierle, Philipp Jovanovic, Martin M. Lauridsen, Gregor Leander, and Christian Rechberger. Analyzing Permutations for AES-like Ciphers: Understanding ShiftRows. In Kaisa Nyberg, editor, *Topics in Cryptology - CT-RSA 2015, The Cryptographer's Track at the RSA Conference 2015, San Francisco, CA, USA, April 20-24, 2015. Proceedings*, volume 9048 of *Lecture Notes in Computer Science*, pages 37–58. Springer, 2015.

[43] Christof Beierle, Philipp Jovanovic, Martin M. Lauridsen, Gregor Leander, and Christian Rechberger. Code Repository for Analysis of ShiftRows in AES-like Ciphers. https://github.com/mmeh/understanding-shiftrows, 2015.

[44] Mihir Bellare and Chanathip Namprempre. Authenticated Encryption: Relations among Notions and Analysis of the Generic Composition Paradigm. In Tatsuaki Okamoto, editor, *Advances in Cryptology - ASIACRYPT 2000, 6th International Conference on the Theory and Application of Cryptology and Information Security, Kyoto, Japan, December 3-7, 2000, Proceedings*, volume 1976 of *Lecture Notes in Computer Science*, pages 531–545. Springer, 2000.

[45] Mihir Bellare, Alexandra Boldyreva, Lars R. Knudsen, and Chanathip Namprempre. Online Ciphers and the Hash-CBC Construction. In Kilian [187], pages 292–309.

[46] Mihir Bellare, Tadayoshi Kohno, and Chanathip Namprempre. Authenticated encryption in SSH: provably fixing the SSH binary packet protocol. In Atluri [33], pages 1–11.

[47] Mihir Bellare, Phillip Rogaway, and David Wagner. The EAX Mode of Operation. In Roy and Meier [274], pages 389–407.

[48] Ryad Benadjila, Olivier Billet, Henri Gilbert, Gilles Macario-Rat, Thomas Peyrin, Matt Robshaw, and Yannick Seurin. ECHO. Submission to the SHA-3 competition, 2008.

[49] Daniel J. Bernstein. Cache-timing attacks on AES, 2005.

[50] Daniel J. Bernstein and Peter Schwabe. New AES Software Speed Records. In Dipanwita Roy Chowdhury, Vincent Rijmen, and Abhijit Das, editors, *Progress in Cryptology - INDOCRYPT 2008, 9th International Conference on Cryptology in India, Kharagpur, India, December 14-17, 2008. Proceedings*, volume 5365 of *Lecture Notes in Computer Science*, pages 322–336. Springer, 2008.

[51] Guido Bertoni, Joan Daemen, Michaël Peeters, and Gilles Van Assche. The Keccak reference, version 3.0. http://keccak.noekeon.org/. Accessed on July 17, 2015.

[52] Guido Bertoni, Joan Daemen, Michaël Peeters, and Gilles Van Assche. Duplexing the Sponge: Single-Pass Authenticated Encryption and Other Applications. In Ali Miri and Serge Vaudenay, editors, *Selected Areas in Cryptography - 18th International Workshop, SAC 2011, Toronto, ON, Canada, August 11-12, 2011, Revised Selected Papers*, volume 7118 of *Lecture Notes in Computer Science*, pages 320–337. Springer, 2011.

[53] Guido Bertoni, Joan Daemen, Michaël Peeters, Gilles Van Assche, and Ronny Van Keer. Ketje. Submission to the CAESAR competition, 2014.

[54] Guido Bertoni, Joan Daemen, Michaël Peeters, Gilles Van Assche, and Ronny Van Keer. Keyak. Submission to the CAESAR competition, 2014.

[55] Eli Biham, editor. *Fast Software Encryption, 4th International Workshop, FSE '97, Haifa, Israel, January 20-22, 1997, Proceedings*, volume 1267 of *Lecture Notes in Computer Science*, 1997. Springer.

[56] Eli Biham and Adi Shamir. *Differential Cryptanalysis of the Data Encryption Standard*. Springer, 1993.

[57] Eli Biham, Ross J. Anderson, and Lars R. Knudsen. Serpent: A New Block Cipher Proposal. In Serge Vaudenay, editor, *Fast Software Encryption, 5th International Workshop, FSE '98, Paris, France, March 23-25, 1998, Proceedings*, volume 1372 of *Lecture Notes in Computer Science*, pages 222–238. Springer, 1998.

[58] Alex Biryukov and Dmitry Khovratovich. Related-Key Cryptanalysis of the Full AES-192 and AES-256. In Matsui [219], pages 1–18.

[59] Alex Biryukov and Dmitry Khovratovich. PAEQ. Submission to the CAESAR competition, 2014.

[60] Alex Biryukov, Dmitry Khovratovich, and Ivica Nikolic. Distinguisher and Related-Key Attack on the Full AES-256. In Shai Halevi, editor, *Advances in Cryptology - CRYPTO 2009, 29th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 16-20, 2009. Proceedings*, volume 5677 of *Lecture Notes in Computer Science*, pages 231–249. Springer, 2009.

[61] Alex Biryukov, Orr Dunkelman, Nathan Keller, Dmitry Khovratovich, and Adi Shamir. Key Recovery Attacks of Practical Complexity on AES-256 Variants with up to 10 Rounds. In Henri Gilbert, editor, *Advances in Cryptology - EUROCRYPT 2010, 29th Annual International Conference on the Theory and Applications of Cryptographic Techniques, French Riviera, May 30 - June 3, 2010. Proceedings*, volume 6110 of *Lecture Notes in Computer Science*, pages 299–319. Springer, 2010.

[62] Alex Biryukov, Guang Gong, and Douglas R. Stinson, editors. *Selected Areas in Cryptography - 17th International Workshop, SAC 2010, Waterloo, Ontario, Canada, August 12-13, 2010, Revised Selected Papers*, volume 6544 of *Lecture Notes in Computer Science*, 2011. Springer.

[63] Alex Biryukov, Arnab Roy, and Vesselin Velichkov. Differential Analysis of Block Ciphers SIMON and SPECK. In Cid and Rechberger [96], pages 546–570.

[64] John Black and Phillip Rogaway. CBC MACs for Arbitrary-Length Messages: The Three-Key Constructions. In Mihir Bellare, editor, *Advances in Cryptology - CRYPTO 2000, 20th Annual International Cryptology Conference, Santa Barbara, California, USA, August 20-24, 2000, Proceedings*, volume 1880 of *Lecture Notes in Computer Science*, pages 197–215. Springer, 2000.

[65] John Black and Phillip Rogaway. A Block-Cipher Mode of Operation for Parallelizable Message Authentication. In Knudsen [190], pages 384–397.

[66] John Black, Phillip Rogaway, and Thomas Shrimpton. Black-Box Analysis of the Block-Cipher-Based Hash-Function Constructions from PGV. In Moti Yung, editor, *Advances in Cryptology - CRYPTO 2002, 22nd Annual International Cryptology Conference, Santa Barbara, California, USA, August 18-22, 2002, Proceedings*, volume 2442 of *Lecture Notes in Computer Science*, pages 320–335. Springer, 2002.

[67] Céline Blondeau and Kaisa Nyberg. New Links between Differential and Linear Cryptanalysis. In Johansson and Nguyen [173], pages 388–404.

[68] Céline Blondeau and Kaisa Nyberg. Links between Truncated Differential and Multidimensional Linear Properties of Block Ciphers and Underlying Attack Complexities. In Nguyen and Oswald [244], pages 165–182.

[69] Céline Blondeau, Thomas Peyrin, and Lei Wang. Known-Key Distinguisher on Full PRESENT. In Gennaro and Robshaw [140], pages 455–474.

[70] BlueKrypt. Cryptographic Key Length Recommendation. http://www.keylength.com. Accessed on July 19, 2015.

[71] Andrey Bogdanov. On the Differential Trails of Unbalanced Feistel Networks with Contracting MDS Diffusion. In *International Workshop on Coding and Cryptography - WCC 2009*, 2009.

[72] Andrey Bogdanov. On unbalanced Feistel networks with contracting MDS diffusion. *Design, Codes and Cryptography*, 59(1-3):35–58, 2011.

[73] Andrey Bogdanov and Christian Rechberger. A 3-Subset Meet-in-the-Middle Attack: Cryptanalysis of the Lightweight Block Cipher KTANTAN. In Biryukov et al. [62], pages 229–240.

[74] Andrey Bogdanov and Vincent Rijmen. Linear hulls with correlation zero and linear cryptanalysis of block ciphers. *Design, Codes and Cryptography*, 70(3):369–383, 2014.

[75] Andrey Bogdanov, Lars R. Knudsen, Gregor Leander, Christof Paar, Axel Poschmann, Matthew J. B. Robshaw, Yannick Seurin, and Charlotte Vikkelsø. PRESENT: An Ultra-Lightweight Block Cipher. In Paillier and Verbauwhede [257], pages 450–466.

[76] Andrey Bogdanov, Gregor Leander, Christof Paar, Axel Poschmann, Matthew J. B. Robshaw, and Yannick Seurin. Hash Functions and RFID Tags: Mind the Gap. In Elisabeth Oswald and Pankaj Rohatgi, editors, *Cryptographic Hardware and Embedded Systems - CHES 2008, 10th International Workshop, Washington, D.C., USA, August 10-13, 2008. Proceedings*, volume 5154 of *Lecture Notes in Computer Science*, pages 283–299. Springer, 2008.

[77] Andrey Bogdanov, Dmitry Khovratovich, and Christian Rechberger. Biclique Cryptanalysis of the Full AES. In Dong Hoon Lee and Xiaoyun Wang, editors, *Advances in Cryptology - ASIACRYPT 2011 - 17th International Conference on the Theory and Application of Cryptology and Information Security, Seoul, South Korea, December 4-8, 2011. Proceedings*, volume 7073 of *Lecture Notes in Computer Science*, pages 344–371. Springer, 2011.

[78] Andrey Bogdanov, Florian Mendel, Francesco Regazzoni, Vincent Rijmen, and Elmar Tischhauser. ALE: AES-Based Lightweight Authenticated Encryption. In Moriai [236], pages 447–466.

[79] Andrey Bogdanov, Christoph Dobraunig, Maria Eichlseder, Martin M. Lauridsen, Florian Mendel, Martin Schläffer, and Elmar Tischhauser. Key Recovery Attacks on Recent Authenticated Ciphers. In Diego F. Aranha and Alfred Menezes, editors, *Progress in Cryptology - LATINCRYPT 2014 - Third International Conference on Cryptology and Information Security in Latin America, Florianópolis, Brazil, September 17-19, 2014, Revised Selected Papers*, volume 8895 of *Lecture Notes in Computer Science*, pages 274–287. Springer, 2014.

[80] Andrey Bogdanov, Martin M. Lauridsen, and Elmar Tischhauser. Comb to Pipeline: Fast Software Encryption Revisited. In Leander [210], pages 150–171.

[81] Dan Boneh, editor. *Advances in Cryptology - CRYPTO 2003, 23rd Annual International Cryptology Conference, Santa Barbara, California, USA, August 17-21, 2003, Proceedings*, volume 2729 of *Lecture Notes in Computer Science*, 2003. Springer.

[82] Julia Borghoff, Lars R. Knudsen, and Mathias Stolpe. Bivium as a Mixed-Integer Linear Programming Problem. In Matthew G. Parker, editor, *Cryptography and Coding, 12th IMA International Conference, Cryptography and Coding 2009, Cirencester, UK, December 15-17, 2009. Proceedings*, volume 5921 of *Lecture Notes in Computer Science*, pages 133–152. Springer, 2009.

[83] Julia Borghoff, Anne Canteaut, Tim Güneysu, Elif Bilge Kavun, Miroslav Knezevic, Lars R. Knudsen, Gregor Leander, Ventzislav Nikov, Christof Paar, Christian Rechberger, Peter Rombouts, Søren S. Thomsen, and Tolga Yalçın. PRINCE - A Low-Latency Block Cipher for Pervasive Computing Applications - Extended Abstract. In Xiaoyun Wang and Kazue Sako, editors, *Advances in Cryptology - ASIACRYPT 2012 - 18th International Conference on the Theory and Application of Cryptology and Information Security, Beijing, China, December 2-6, 2012. Proceedings*, volume 7658 of *Lecture Notes in Computer Science*, pages 208–225. Springer, 2012.

[84] Christina Boura, María Naya-Plasencia, and Valentin Suder. Scrutinizing and Improving Impossible Differential Attacks: Applications to CLEFIA, Camellia, LBlock and Simon. In Sarkar and Iwata [279], pages 179–199.

[85] Christina Boura, María Naya-Plasencia, and Valentin Suder. Scrutinizing and Improving Impossible Differential Attacks: Applications to CLEFIA, Camellia, LBlock and Simon (Full Version). Cryptology ePrint Archive, Report 2014/699, 2014.

[86] Gilles Brassard, editor. *Advances in Cryptology - CRYPTO '89, 9th Annual International Cryptology Conference, Santa Barbara, California, USA, August 20-24, 1989, Proceedings*, volume 435 of *Lecture Notes in Computer Science*, 1990. Springer.

[87] Stanislav Bulygin. More on linear hulls of PRESENT-like ciphers and a cryptanalysis of full-round EPCBC-96. Cryptology ePrint Archive, Report 2013/028, 2013.

[88] Christophe De Cannière, Orr Dunkelman, and Miroslav Knezevic. KATAN and KTANTAN - A Family of Small and Efficient Hardware-Oriented Block Ciphers. In Clavier and Gaj [97], pages 272–288.

[89] Anne Canteaut, editor. *Fast Software Encryption - 19th International Workshop, FSE 2012, Washington, DC, USA, March 19-21, 2012. Revised Selected Papers*, volume 7549 of *Lecture Notes in Computer Science*, 2012. Springer.

[90] Anne Canteaut and Joëlle Roué. On the Behaviors of Affine Equivalent Sboxes Regarding Differential and Linear Attacks. In Oswald and Fischlin [256], pages 45–74.

[91] Florent Chabaud and Serge Vaudenay. Links Between Differential and Linear Cryptanalysis. In Alfredo De Santis, editor, *Advances in Cryptology - EUROCRYPT '94, Workshop on the Theory and Application of Cryptographic Techniques, Perugia, Italy, May 9-12, 1994, Proceedings*, volume 950 of *Lecture Notes in Computer Science*, pages 356–365. Springer, 1994.

[92] Huaifeng Chen and Xiaoyun Wang. Improved Linear Hull Attack on Round-Reduced Simon with Dynamic Key-guessing Techniques. Cryptology ePrint Archive, Report 2015/666, 2015.

[93] Shan Chen and John P. Steinberger. Tight Security Bounds for Key-Alternating Ciphers. In Nguyen and Oswald [244], pages 327–350.

[94] Zhan Chen, Ning Wang, and Xiaoyun Wang. Impossible Differential Cryptanalysis of Reduced Round SIMON. Cryptology ePrint Archive, Report 2015/286, 2015.

[95] Joo Yeon Cho. Linear Cryptanalysis of Reduced-Round PRESENT. In Pieprzyk [260], pages 302–317.

[96] Carlos Cid and Christian Rechberger, editors. *Fast Software Encryption - 21st International Workshop, FSE 2014, London, UK, March 3-5, 2014. Revised Selected Papers*, volume 8540 of *Lecture Notes in Computer Science*, 2015. Springer.

[97] Christophe Clavier and Kris Gaj, editors. *Cryptographic Hardware and Embedded Systems - CHES 2009, 11th International Workshop, Lausanne, Switzerland, September 6-9, 2009, Proceedings*, volume 5747 of *Lecture Notes in Computer Science*, 2009. Springer.

[98] Don Coppersmith. The Data Encryption Standard (DES) and its strength against attacks. *IBM Journal of Research and Development*, 38(3):243–250, 1994.

[99] Jean-Sébastien Coron and Louis Goubin. On Boolean and Arithmetic Masking against Differential Power Analysis. In Çetin Kaya Koç and Christof Paar, editors, *Cryptographic Hardware and Embedded Systems - CHES 2000, Second International Workshop, Worcester, MA, USA, August 17-18, 2000, Proceedings*, volume 1965 of *Lecture Notes in Computer Science*, pages 231–237. Springer, 2000.

[100] International Business Machines Corporation. *ILOG CPLEX Optimizer*, 1997-2014.

[101] Nicolas Courtois. Fast Algebraic Attacks on Stream Ciphers with Linear Feedback. In Boneh [81], pages 176–194.

[102] Nicolas Courtois and Willi Meier. Algebraic Attacks on Stream Ciphers with Linear Feedback. In Eli Biham, editor, *Advances in Cryptology - EUROCRYPT 2003, International Conference on the Theory and Applications of Cryptographic Techniques, Warsaw, Poland, May 4-8, 2003, Proceedings*, volume 2656 of *Lecture Notes in Computer Science*, pages 345–359. Springer, 2003.

[103] Nicolas Courtois, Gregory V. Bard, and David Wagner. Algebraic and Slide Attacks on KeeLoq. In Nyberg [247], pages 97–115.

[104] Joan Daemen and Vincent Rijmen. AES submission document on Rijndael, June 1998.

[105] Joan Daemen and Vincent Rijmen. The Wide Trail Design Strategy. In Bahram Honary, editor, *Cryptography and Coding, 8th IMA International Conference, Cirencester, UK, December 17-19, 2001, Proceedings*, volume 2260 of *Lecture Notes in Computer Science*, pages 222–238. Springer, 2001.

[106] Joan Daemen and Vincent Rijmen. *The Design of Rijndael*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2002.

[107] Joan Daemen, Lars R. Knudsen, and Vincent Rijmen. The Block Cipher Square. In Biham [55], pages 149–165.

[108] Joan Daemen, Lars R. Knudsen, and Vincent Rijmen. Linear Frameworks for Block Ciphers. *Design, Codes and Cryptography*, 22(1):65–87, 2001.

[109] Ivan Damgård. A Design Principle for Hash Functions. In Brassard [86], pages 416–427.

[110] Donald W. Davies and Wyn L. Price. Digital signatures, an update. In *Proceedings 5th international conference on computer communication*, pages 845–849, October 1984.

[111] Hüseyin Demirci and Ali Aydin Selçuk. A Meet-in-the-Middle Attack on 8-Round AES. In Nyberg [247], pages 116–126.

[112] Patrick Derbez, Pierre-Alain Fouque, and Jérémy Jean. Improved Key Recovery Attacks on Reduced-Round AES in the Single-Key Setting. In Johansson and Nguyen [173], pages 371–387.

[113] Yvo Desmedt, editor. *Advances in Cryptology - CRYPTO '94, 14th Annual International Cryptology Conference, Santa Barbara, California, USA, August 21-25, 1994, Proceedings*, volume 839 of *Lecture Notes in Computer Science*, 1994. Springer.

[114] Tim Dierks and Eric Rescorla. The Transport Layer Security (TLS) Protocol Version 1.2. RFC 5246 (Proposed Standard), August 2008.

[115] Itai Dinur, Orr Dunkelman, Masha Gutman, and Adi Shamir. Improved Top-Down Techniques in Differential Cryptanalysis. Cryptology ePrint Archive, Report 2015/268, 2015.

[116] Christoph Dobraunig, Maria Eichlseder, Florian Mendel, and Martin Schläffer. Ascon. Submission to the CAESAR competition, 2014.

[117] Christoph Dobraunig, Maria Eichlseder, and Florian Mendel. Related-Key Forgeries for Prøst-OTR. In Leander [210], pages 282–296.

[118] Orr Dunkelman, Nathan Keller, and Adi Shamir. Improved Single-Key Attacks on 8-Round AES-192 and AES-256. In Abe [4], pages 158–176.

[119] Orr Dunkelman, Nathan Keller, and Adi Shamir. Minimalism in Cryptography: The Even-Mansour Scheme Revisited. In David Pointcheval and Thomas Johansson, editors, *Advances in Cryptology - EUROCRYPT 2012 - 31st Annual International Conference on the Theory and Applications of Cryptographic Techniques, Cambridge, UK, April 15-19, 2012. Proceedings*, volume 7237 of *Lecture Notes in Computer Science*, pages 336–354. Springer, 2012.

[120] Morris J. Dworkin. SP 800-38C. Recommendation for Block Cipher Modes of Operation: The CCM Mode for Authentication and Confidentiality. Technical report, National Institute of Standards and Technology, Gaithersburg, MD, United States, 2004.

[121] Morris J. Dworkin. SP 800-38D. Recommendation for Block Cipher Modes of Operation: Galois/Counter Mode (GCM) and GMAC. Technical report, National Institute of Standards and Technology, Gaithersburg, MD, United States, 2007.

[122] Shimon Even and Yishay Mansour. A Construction of a Cipher From a Single Pseudorandom Permutation. In Hideki Imai, Ronald L. Rivest, and Tsutomu Matsumoto, editors, *Advances in Cryptology - ASIACRYPT '91, International Conference on the Theory and Applications of Cryptology, Fujiyoshida, Japan, November 11-14, 1991, Proceedings*, volume 739 of *Lecture Notes in Computer Science*, pages 210–224. Springer, 1991.

[123] Horst Feistel. *Cryptography and Computer Privacy*. Scientific American, 1973.

[124] Niels Ferguson. Collision attacks on OCB, February 2002.

[125] Niels Ferguson. Authentication weaknesses in GCM, 2005.

[126] Ewan Fleischmann, Christian Forler, Michael Gorski, and Stefan Lucks. Twister- A Framework for Secure and Fast Hash Functions. In Feng Bao, Hui Li, and Guilin Wang, editors, *Information Security Practice and Experience, 5th International Conference, ISPEC 2009, Xi'an, China, April 13-15, 2009, Proceedings*, volume 5451 of *Lecture Notes in Computer Science*, pages 257–273. Springer, 2009.

[127] Ewan Fleischmann, Christian Forler, Stefan Lucks, and Jakob Wenzel. McOE: A Family of Almost Foolproof On-Line Authenticated Encryption Schemes. Cryptology ePrint Archive, Report 2011/644, 2011.

[128] Ewan Fleischmann, Christian Forler, and Stefan Lucks. McOE: A Family of Almost Foolproof On-Line Authenticated Encryption Schemes. In Canteaut [89], pages 196–215.

[129] Agner Fog. Software Optimization Resources. http://www.agner.org/optimize. Accessed on February 17, 2014.

[130] International Organization for Standardization. ISO/IEC 19772:2009: Information technology – Security techniques – Authenticated encryption, 2009.

[131] International Organization for Standardization. ISO/IEC 10118-2:2010: Information technology – Security techniques – Hash-functions – Part 2: Hash-functions using an n-bit block cipher, 2010.

[132] International Organization for Standardization. ISO/IEC 29192-2:2012: Information technology – Security techniques – Lightweight cryptography – Part 2: Block ciphers, 2012.

[133] Electronic Frontier Foundation. *Cracking DES: Secrets of Encryption Research, Wiretap Politics and Chip Design*. O'Reilly & Associates, Inc., Sebastopol, CA, USA, 1998.

[134] OpenSSL Software Foundation. OpenSSL – Cryptography and SSL/TLS Toolkit. https://www.openssl.org. Accessed on July 17, 2015.

[135] Pierre-Alain Fouque and Pierre Karpman. Security Amplification against Meet-in-the-Middle Attacks Using Whitening. In Martijn Stam, editor, *Cryptography and Coding - 14th IMA International Conference, IMACC 2013, Oxford, UK, December 17-19, 2013. Proceedings*, volume 8308 of *Lecture Notes in Computer Science*, pages 252–269. Springer, 2013.

[136] Pierre-Alain Fouque, Antoine Joux, Gwenaëlle Martinet, and Frédéric Valette. Authenticated On-Line Encryption. In Mitsuru Matsui and Robert J. Zuccherato, editors, *Selected Areas in Cryptography, 10th Annual International Workshop, SAC 2003, Ottawa, Canada, August 14-15, 2003, Revised Papers*, volume 3006 of *Lecture Notes in Computer Science*, pages 145–159. Springer, 2003.

[137] Alan Freier, Philip Karlton, and Paul Kocher. The Secure Sockets Layer (SSL) Protocol Version 3.0. RFC 6101 (Historic), August 2011.

[138] Michael R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. A Series of books in the mathematical sciences. W. H. Freeman and Company, 1979.

[139] Praveen Gauravaram, Lars R. Knudsen, Krystian Matusiewicz, Florian Mendel, Christian Rechberger, Martin Schläffer, and Søren S. Thomsen. Grøstl. Submission to the SHA-3 competition, 2008.

[140] Rosario Gennaro and Matthew Robshaw, editors. *Advances in Cryptology - CRYPTO 2015 - 35th Annual Cryptology Conference, Santa Barbara, CA, USA, August 16-20, 2015, Proceedings, Part I*, volume 9215 of *Lecture Notes in Computer Science*, 2015. Springer.

[141] Henri Gilbert. A Simplified Representation of AES. In Sarkar and Iwata [279], pages 200–222.

[142] Henri Gilbert and Thomas Peyrin. Super-Sbox Cryptanalysis: Improved Attacks for AES-Like Permutations. In Seokhie Hong and Tetsu Iwata, editors, *Fast Software Encryption, 17th International Workshop, FSE 2010, Seoul, Korea, February 7-10, 2010, Revised Selected Papers*, volume 6147 of *Lecture Notes in Computer Science*, pages 365–383. Springer, 2010.

[143] Brian Gladman. AES Implementations. https://github.com/BrianGladman/AES. Accessed on August 3, 2015.

[144] Danilo Gligoroski, Hristina Mihajloska, Simona Samardjiska, Håkon Jacobsen, Mohamed El-Hadedy, and Rune Erlend Jensen. π-Cipher. Submission to the CAESAR competition, 2014.

[145] Jovan Dj. Golic and Christophe Tymen. Multiplicative Masking and Power Analysis of AES. In Burton S. Kaliski Jr., Çetin Kaya Koç, and Christof Paar, editors, *Cryptographic Hardware and Embedded Systems - CHES 2002, 4th International Workshop, Redwood Shores, CA, USA,*

August 13-15, 2002, Revised Papers, volume 2523 of *Lecture Notes in Computer Science*, pages 198–212. Springer, 2002.

[146] Shay Gueron. *Intel Advanced Encryption Standard (AES) New Instructions Set*. Intel Corporation, 2010.

[147] Shay Gueron. AES-GCM software performance on the current high end CPUs as a performance baseline for CAESAR. In *DIAC 2013: Directions in Authenticated Ciphers*, 2013.

[148] Shay Gueron and Michael E. Kounavis. *Intel Carry-Less Multiplication Instruction and its Usage for Computing the GCM Mode*. Intel Corporation, 2010.

[149] Shay Gueron and Vlad Krasnov. The fragility of AES-GCM authentication algorithm. Cryptology ePrint Archive, Report 2013/157, 2013.

[150] Sean Gulley and Vinodh Gopal. *Haswell Cryptographic Performance*. Intel Corporation, 2013.

[151] Jian Guo, Thomas Peyrin, and Axel Poschmann. The PHOTON Family of Lightweight Hash Functions. In Phillip Rogaway, editor, *Advances in Cryptology - CRYPTO 2011 - 31st Annual Cryptology Conference, Santa Barbara, CA, USA, August 14-18, 2011. Proceedings*, volume 6841 of *Lecture Notes in Computer Science*, pages 222–239. Springer, 2011.

[152] Jian Guo, Thomas Peyrin, Axel Poschmann, and Matthew J. B. Robshaw. The LED Block Cipher. In Bart Preneel and Tsuyoshi Takagi, editors, *Cryptographic Hardware and Embedded Systems - CHES 2011 - 13th International Workshop, Nara, Japan, September 28 - October 1, 2011. Proceedings*, volume 6917 of *Lecture Notes in Computer Science*, pages 326–341. Springer, 2011.

[153] Peter Gutmann. Encrypt-then-MAC for Transport Layer Security (TLS) and Datagram Transport Layer Security (DTLS). RFC 7366 (Proposed Standard), September 2014.

[154] Helena Handschuh and Bart Preneel. Key-Recovery Attacks on Universal Hash Function Based MAC Algorithms. In David Wagner, editor, *Advances in Cryptology - CRYPTO 2008, 28th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 17-21, 2008. Proceedings*, volume 5157 of *Lecture Notes in Computer Science*, pages 144–161. Springer, 2008.

[155] Martin E. Hellman. A cryptanalytic time-memory trade-off. *IEEE Transactions on Information Theory*, 26(4):401–406, 1980.

[156] John L. Hennessy and David A. Patterson. *Computer Architecture: A Quantitative Approach*. The Morgan Kaufmann Series in Computer Architecture and Design. Elsevier Science, 2011.

[157] Miia Hermelin, Joo Yeon Cho, and Kaisa Nyberg. Multidimensional Linear Cryptanalysis of Reduced Round Serpent. In Yi Mu, Willy Susilo, and Jennifer Seberry, editors, *Information Security and Privacy, 13th Australasian Conference, ACISP 2008, Wollongong, Australia, July*

7-9, 2008, Proceedings, volume 5107 of *Lecture Notes in Computer Science*, pages 203–215. Springer, 2008.

[158] Viet Tung Hoang, Reza Reyhanitabar, Phillip Rogaway, and Damian Vizár. Online Authenticated-Encryption and its Nonce-Reuse Misuse-Resistance. In Gennaro and Robshaw [140], pages 493–517.

[159] Brent Hollingsworth. *New "Bulldozer" and "Piledriver" Instructions*. Advanced Micro Devices, Inc., 2012.

[160] Russel Housley. Using Advanced Encryption Standard (AES) CCM Mode with IPsec Encapsulating Security Payload (ESP). RFC 4309 (Proposed Standard), December 2005.

[161] Sebastiaan Indesteege, Elena Andreeva, Christophe De Cannière, Orr Dunkelman, Emilia Käsper, Svetla Nikova, Bart Preneel, and Elmar Tischhauser. The LANE hash function. Submission to the SHA-3 competition, 2008.

[162] Institute of Electrical and Electronics Engineers. 802.15.1-2002: IEEE Standard for Telecommunications and Information Exchange Between Systems – LAN/MAN – Specific Requirements – Part 15: Wireless Medium Access Control (MAC) and Physical Layer (PHY) Specifications for Wireless Personal Area Networks (WPANs), 2002.

[163] Institute of Electrical and Electronics Engineers. 802.11i-2004: IEEE Standard for information technology – Telecommunications and information exchange between systems – Local and metropolitan area networks-Specific requirements – Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) specifications: Amendment 6: Medium Access Control (MAC) Security Enhancements, 2004.

[164] Steven Iveson. IPSec Bandwidth Overhead Using AES. http://packetpushers.net/ipsec-bandwidth-overhead-using-aes. Accessed on February 17, 2014.

[165] Tetsu Iwata and Kaoru Kurosawa. OMAC: One-Key CBC MAC. In Thomas Johansson, editor, *Fast Software Encryption, 10th International Workshop, FSE 2003, Lund, Sweden, February 24-26, 2003, Revised Papers*, volume 2887 of *Lecture Notes in Computer Science*, pages 129–153. Springer, 2003.

[166] Tetsu Iwata and Kaoru Kurosawa. Stronger Security Bounds for OMAC, TMAC, and XCBC. In Thomas Johansson and Subhamoy Maitra, editors, *Progress in Cryptology - INDOCRYPT 2003, 4th International Conference on Cryptology in India, New Delhi, India, December 8-10, 2003, Proceedings*, volume 2904 of *Lecture Notes in Computer Science*, pages 402–415. Springer, 2003.

[167] Tetsu Iwata, Keisuke Ohashi, and Kazuhiko Minematsu. Breaking and Repairing GCM Security Proofs. In Reihaneh Safavi-Naini and Ran Canetti, editors, *Advances in Cryptology - CRYPTO 2012 - 32nd Annual Cryptology Conference, Santa Barbara, CA, USA, August 19-23, 2012. Proceedings*, volume 7417 of *Lecture Notes in Computer Science*, pages 31–49. Springer, 2012.

[168] Tetsu Iwata, Kazuhiko Minematsu, Jian Guo, and Sumio Morioka. CLOC: Authenticated Encryption for Short Input. In Cid and Rechberger [96], pages 149–167.

[169] Tetsu Iwata, Kazuhiko Minematsu, Jian Guo, Sumio Morioka, and Eita Kobayashi. SILC. Submission to the CAESAR competition, 2014.

[170] Thomas Jakobsen and Lars R. Knudsen. The Interpolation Attack on Block Ciphers. In Biham [55], pages 28–40.

[171] Krzysztof Jankowski and Pierre Laurent. Packed AES-GCM Algorithm Suitable for AES/PCLMULQDQ Instructions. *IEEE Transactions on Computers*, 60(1):135–138, 2011.

[172] Zahra Jeddi, Esmaeil Amini, and Magdy Bayoumi. A Novel Authenticated Cipher for RFID Systems. In *International Journal on Cryptography and Information Security*, volume 4, 2014.

[173] Thomas Johansson and Phong Q. Nguyen, editors. *Advances in Cryptology - EUROCRYPT 2013, 32nd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Athens, Greece, May 26-30, 2013. Proceedings*, volume 7881 of *Lecture Notes in Computer Science*, 2013. Springer.

[174] Wolfgang John and Sven Tafvelin. Analysis of internet backbone traffic and header anomalies observed. In Constantine Dovrolis and Matthew Roughan, editors, *Proceedings of the 7th ACM SIGCOMM Internet Measurement Conference, IMC 2007, San Diego, California, USA, October 24-26, 2007*, pages 111–116. ACM, 2007.

[175] Thomas R. Johnson. American Cryptology during the Cold War, 1945-1989: Book III: Retrenchment and Reform (declassified document). http://nsarchive.gwu.edu/NSAEBB/NSAEBB260/nsa-6.pdf. Accessed on July 23, 2015.

[176] Antoine Joux. Authentication Failures in NIST version of GCM, 2006.

[177] Burton S. Kaliski Jr. and Matthew J. B. Robshaw. Linear Cryptanalysis Using Multiple Approximations. In Desmedt [113], pages 26–39.

[178] Jorge Nakahara Jr. 3D: A Three-Dimensional Block Cipher. In Matthew K. Franklin, Lucas Chi Kwong Hui, and Duncan S. Wong, editors, *Cryptology and Network Security, 7th International Conference, CANS 2008, Hong-Kong, China, December 2-4, 2008. Proceedings*, volume 5339 of *Lecture Notes in Computer Science*, pages 252–267. Springer, 2008.

[179] David Kahn. *The Codebreakers: The Comprehensive History of Secret Communication from Ancient Times to the Internet*. Scribner, 1996.

[180] Pierre Karpman. From Related-Key Distinguishers to Related-Key-Recovery on Even-Mansour Constructions. Cryptology ePrint Archive, Report 2015/134, 2015.

[181] Elif Bilge Kavun, Martin M. Lauridsen, Gregor Leander, Christian Rechberger, Peter Schwabe, and Tolga Yalçın. Prøst. Submission to the CAESAR competition, 2014.

[182] Stephen Kent and Karen Seo. Security Architecture for the Internet Protocol. RFC 4301 (Proposed Standard), December 2005.

[183] Dmitry Khovratovich. PPAE: Parallelizable Permutation-based Authenticated Encryption. In *DIAC 2013: Directions in Authenticated Ciphers*, 2013.

[184] Dmitry Khovratovich, María Naya-Plasencia, Andrea Röck, and Martin Schläffer. Cryptanalysis of *Luffa* v2 Components. In Biryukov et al. [62], pages 388–409.

[185] Dmitry Khovratovich, Ivica Nikolic, and Christian Rechberger. Rotational Rebound Attacks on Reduced Skein. In Abe [4], pages 1–19.

[186] Dmitry Khovratovich, Christian Rechberger, and Alexandra Savelieva. Bicliques for Preimages: Attacks on Skein-512 and the SHA-2 Family. In Canteaut [89], pages 244–263.

[187] Joe Kilian, editor. *Advances in Cryptology - CRYPTO 2001, 21st Annual International Cryptology Conference, Santa Barbara, California, USA, August 19-23, 2001, Proceedings*, volume 2139 of *Lecture Notes in Computer Science*, 2001. Springer.

[188] Lars R. Knudsen. Truncated and Higher Order Differentials. In Preneel [261], pages 196–211.

[189] Lars R. Knudsen. DEAL - A 128-bit Block Cipher, February 1998.

[190] Lars R. Knudsen, editor. *Advances in Cryptology - EUROCRYPT 2002, International Conference on the Theory and Applications of Cryptographic Techniques, Amsterdam, The Netherlands, April 28 - May 2, 2002, Proceedings*, volume 2332 of *Lecture Notes in Computer Science*, 2002. Springer.

[191] Lars R. Knudsen and Vincent Rijmen. Known-Key Distinguishers for Some Block Ciphers. In Kaoru Kurosawa, editor, *Advances in Cryptology - ASIACRYPT 2007, 13th International Conference on the Theory and Application of Cryptology and Information Security, Kuching, Malaysia, December 2-6, 2007, Proceedings*, volume 4833 of *Lecture Notes in Computer Science*, pages 315–324. Springer, 2007.

[192] Lars R. Knudsen and Matthew J. B. Robshaw. *The Block Cipher Companion*. Information Security and Cryptography. Springer-Verlag Berlin Heidelberg, 2011.

[193] Paul C. Kocher, Joshua Jaffe, and Benjamin Jun. Differential Power Analysis. In Michael J. Wiener, editor, *Advances in Cryptology - CRYPTO '99, 19th Annual International Cryptology Conference, Santa Barbara, California, USA, August 15-19, 1999, Proceedings*, volume 1666 of *Lecture Notes in Computer Science*, pages 388–397. Springer, 1999.

[194] Hugo Krawczyk. Communication on TLS mailing list. http://www.ietf.org/mail-archive/web/tls/current/msg12766.html. Accessed on May 18, 2015.

[195] Hugo Krawczyk. The Order of Encryption and Authentication for Protecting Communications (or: How Secure Is SSL?). In Kilian [187], pages 310–331.

[196] Ted Krovetz and Phillip Rogaway. The Software Performance of Authenticated-Encryption Modes. In Antoine Joux, editor, *Fast Software Encryption - 18th International Workshop, FSE 2011, Lyngby, Denmark, February 13-16, 2011, Revised Selected Papers*, volume 6733 of *Lecture Notes in Computer Science*, pages 306–327. Springer, 2011.

[197] Sandeep S. Kumar, Christof Paar, Jan Pelzl, Gerd Pfeiffer, and Manfred Schimmler. Breaking Ciphers with COPACOBANA - A Cost-Optimized Parallel Code Breaker. In Louis Goubin and Mitsuru Matsui, editors, *Cryptographic Hardware and Embedded Systems - CHES 2006, 8th International Workshop, Yokohama, Japan, October 10-13, 2006, Proceedings*, volume 4249 of *Lecture Notes in Computer Science*, pages 101–118. Springer, 2006.

[198] Emilia Käsper and Peter Schwabe. Faster and Timing-Attack Resistant AES-GCM. In Clavier and Gaj [97], pages 1–17.

[199] Stefan Kölbl, Gregor Leander, and Tyge Tiessen. Observations on the SIMON Block Cipher Family. In Gennaro and Robshaw [140], pages 161–185.

[200] Xuejia Lai. *On the design and security of block ciphers*. PhD thesis, Swiss Federal Institute of Technology, 1992.

[201] Xuejia Lai. Higher Order Derivatives and Differential Cryptanalysis. In Richard E. Blahut, Daniel J. Costello Jr., Ueli Maurer, and Thomas Mittelholzer, editors, *Communications and Cryptography*, volume 276 of *The Springer International Series in Engineering and Computer Science*, pages 227–233. Springer US, 1994.

[202] Xuejia Lai, James L. Massey, and Sean Murphy. Markov Ciphers and Differential Cryptanalysis. In Donald W. Davies, editor, *Advances in Cryptology - EUROCRYPT '91, Workshop on the Theory and Application of of Cryptographic Techniques, Brighton, UK, April 8-11, 1991, Proceedings*, volume 547 of *Lecture Notes in Computer Science*, pages 17–38. Springer, 1991.

[203] Mario Lamberger, Florian Mendel, Christian Rechberger, Vincent Rijmen, and Martin Schläffer. Rebound Distinguishers: Results on the Full Whirlpool Compression Function. In Matsui [219], pages 126–143.

[204] Mario Lamberger, Florian Mendel, Martin Schläffer, Christian Rechberger, and Vincent Rijmen. The Rebound Attack and Subspace Distinguishers: Application to Whirlpool. *J. Cryptology*, 28(2):257–296, 2015.

[205] Adam Langley. Overclocking SSL. https://www.imperialviolet.org/2010/06/25/overclocking-ssl.html. Accessed on August 25, 2015.

[206] Martin M. Lauridsen. Security Proofs for Prøst. http://proest.compute.dtu.dk/proofs.pdf, 2015.

[207] Martin M. Lauridsen and Christian Rechberger. Linear Distinguishers in the Key-less Setting: Application to PRESENT. In Leander [210], pages 217–240.

[208] Martin M. Lauridsen and Christian Rechberger. PRESENT Cryptanalysis Code Repository. https://github.com/mmeh/present-keyless, 2015.

[209] Gregor Leander. On Linear Hulls, Statistical Saturation Attacks, PRESENT and a Cryptanalysis of PUFFIN. In Kenneth G. Paterson, editor, *Advances in Cryptology - EUROCRYPT 2011 - 30th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Tallinn, Estonia, May 15-19, 2011. Proceedings*, volume 6632 of *Lecture Notes in Computer Science*, pages 303–322. Springer, 2011.

[210] Gregor Leander, editor. *Fast Software Encryption - 22nd International Workshop, FSE 2015, Istanbul, Turkey, March 8-11, 2015, Revised Selected Papers*, volume 9054 of *Lecture Notes in Computer Science*, 2015. Springer.

[211] Chae Hoon Lim and Tymur Korkishko. mCrypton - A Lightweight Block Cipher for Security of Low-Cost RFID Tags and Sensors. In JooSeok Song, Taekyoung Kwon, and Moti Yung, editors, *Information Security Applications, 6th International Workshop, WISA 2005, Jeju Island, Korea, August 22-24, 2005, Revised Selected Papers*, volume 3786 of *Lecture Notes in Computer Science*, pages 243–258. Springer, 2005.

[212] Chae Hoon Lim and Pil Joong Lee. More Flexible Exponentiation with Precomputation. In Desmedt [113], pages 95–107.

[213] Helger Lipmaa, Phillip Rogaway, and David Wagner. Comments to NIST concerning AES Modes of Operations: CTR-Mode Encryption, 2000.

[214] Jiqiang Lu. On the Security of the COPA and Marble Authenticated Encryption Algorithms against (Almost) Universal Forgery Attack. Cryptology ePrint Archive, Report 2015/079, 2015.

[215] Florence J. MacWilliams and Neil J. A. Sloane. *The Theory of Error-Correcting Codes*. North-Holland Publishing Company, 2nd edition, 1978.

[216] Stefan Mangard, Elisabeth Oswald, and Thomas Popp. *Power Analysis Attacks: Revealing the Secrets of Smart Cards*. Advances in information security. Springer US, 2008.

[217] Mitsuru Matsui. Linear Cryptanalysis Method for DES Cipher. In Tor Helleseth, editor, *Advances in Cryptology - EUROCRYPT '93, Workshop on the Theory and Application of of Cryptographic Techniques, Lofthus, Norway, May 23-27, 1993, Proceedings*, volume 765 of *Lecture Notes in Computer Science*, pages 386–397. Springer, 1993.

[218] Mitsuru Matsui. New Block Encryption Algorithm MISTY. In Biham [55], pages 54–68.

[219] Mitsuru Matsui, editor. *Advances in Cryptology - ASIACRYPT 2009, 15th International Conference on the Theory and Application of Cryptology and Information Security, Tokyo, Japan, December 6-10, 2009. Proceedings*, volume 5912 of *Lecture Notes in Computer Science*, 2009. Springer.

[220] Mitsuru Matsui and Junko Nakajima. On the Power of Bitslice Implementation on Intel Core2 Processor. In Paillier and Verbauwhede [257], pages 121–134.

[221] Stephen M. Matyas, Carl H. Meyer, and Jeffrey Oseas. Generating strong one-way functions with cryptographic algorithm. In *IBM Technical Disclosure Bulletin*, volume 27(10A), pages 5658–5659, 1985.

[222] David A. McGrew and Daniel V. Bailey. AES-CCM Cipher Suites for Transport Layer Security (TLS). RFC 6655 (Proposed Standard), July 2012.

[223] David A. McGrew and John Viega. The Security and Performance of the Galois/Counter Mode (GCM) of Operation. In Anne Canteaut and Kapalee Viswanathan, editors, *Progress in Cryptology - INDOCRYPT 2004, 5th International Conference on Cryptology in India, Chennai, India, December 20-22, 2004, Proceedings*, volume 3348 of *Lecture Notes in Computer Science*, pages 343–355. Springer, 2004.

[224] David A. McGrew and John Viega. The Galois/Counter Mode of Operation (GCM), 2004.

[225] Florian Mendel, Christian Rechberger, Martin Schläffer, and Søren S. Thomsen. The Rebound Attack: Cryptanalysis of Reduced Whirlpool and Grøstl. In Orr Dunkelman, editor, *Fast Software Encryption, 16th International Workshop, FSE 2009, Leuven, Belgium, February 22-25, 2009, Revised Selected Papers*, volume 5665 of *Lecture Notes in Computer Science*, pages 260–276. Springer, 2009.

[226] Florian Mendel, Christian Rechberger, Martin Schläffer, and Søren S. Thomsen. Rebound Attacks on the Reduced Grøstl Hash Function. In Pieprzyk [260], pages 350–365.

[227] Florian Mendel, Bart Mennink, Vincent Rijmen, and Elmar Tischhauser. A Simple Key-Recovery Attack on McOE-X. In Josef Pieprzyk, Ahmad-Reza Sadeghi, and Mark Manulis, editors, *Cryptology and Network Security, 11th International Conference, CANS 2012, Darmstadt, Germany, December 12-14, 2012. Proceedings*, volume 7712, pages 23–31. Springer, 2012.

[228] Alfred J. Menezes, Paul C. van Oorschot, and Scott A. Vanstone. *Handbook of Applied Cryptography*. CRC Press, 1997.

[229] Bart Mennink. XPX: Generalized Tweakable Even-Mansour with Improved Security Guarantees. Cryptology ePrint Archive, Report 2015/476, 2015.

[230] Ralph C. Merkle. *Secrecy, Authentication, and Public Key Systems*. PhD thesis, Department of Electrical Engineering, Stanford University, 1979.

[231] Ralph C. Merkle. A Certified Digital Signature. In Brassard [86], pages 218–238.

[232] Kazuhiko Minematsu. Parallelizable Rate-1 Authenticated Encryption from Pseudorandom Functions. In Nguyen and Oswald [244], pages 275–292.

[233] Shoji Miyaguchi. The FEAL Cipher Family. In Alfred Menezes and Scott A. Vanstone, editors, *Advances in Cryptology - CRYPTO '90, 10th Annual International Cryptology Conference, Santa Barbara, California, USA, August 11-15, 1990, Proceedings*, volume 537 of *Lecture Notes in Computer Science*, pages 627–638. Springer, 1990.

[234] Shoji Miyaguchi, Masahiko Iwata, and Kazuo Ohta. New 128-bit hash function. In *Proceeding of 4th international joint workshop on computer communications*, pages 279–288, July 1989.

[235] Paweł Morawiecki, Kris Gaj, Ekawat Homsirikamol, Krystian Matusiewicz, Josef Pieprzyk, Marcin Rogawski, Marian Srebrny, and Marcin Wójcik. ICEPOLE. Submission to the CAESAR competition, 2014.

[236] Shiho Moriai, editor. *Fast Software Encryption - 20th International Workshop, FSE 2013, Singapore, March 11-13, 2013. Revised Selected Papers*, volume 8424 of *Lecture Notes in Computer Science*, 2014. Springer.

[237] Nicky Mouha, Qingju Wang, Dawu Gu, and Bart Preneel. Differential and Linear Cryptanalysis Using Mixed-Integer Linear Programming. In Chuankun Wu, Moti Yung, and Dongdai Lin, editors, *Information Security and Cryptology - 7th International Conference, Inscrypt 2011, Beijing, China, November 30 - December 3, 2011. Revised Selected Papers*, volume 7537 of *Lecture Notes in Computer Science*, pages 57–76. Springer, 2011.

[238] Theodosis Mourouzis, Guangyan Song, Nicolas Courtois, and Michalis Christofii. Advanced Differential Cryptanalysis of Reduced-Round SIMON64/128 Using Large-Round Statistical Distinguishers. Cryptology ePrint Archive, Report 2015/481, 2015.

[239] David Murray and Terry Koziniec. The state of enterprise network traffic in 2012. In *Communications (APCC), 2012 18th Asia-Pacific Conference on Communications*, pages 179–184. IEEE, 2012.

[240] Bodo Möller, Thai Duong, and Krzysztof Kotowicz. This POODLE Bites: Exploiting The SSL 3.0 Fallback. https://www.openssl.org/~bodo/ssl-poodle.pdf. Accessed on May 18, 2015.

[241] Mridul Nandi. XLS is Not a Strong Pseudorandom Permutation. In Sarkar and Iwata [279], pages 478–490.

[242] Mridul Nandi. Revisiting Security Claims of XLS and COPA. Cryptology ePrint Archive, Report 2015/444, 2015.

[243] National Institute of Standards and Technology. NIST Computer Security Division – The SHA-3 Cryptographic Hash Algorithm Competition, November 2007 - October 2012. http://csrc.nist.gov/groups/ST/hash/sha-3. Accessed on July 17, 2015.

[244] Phong Q. Nguyen and Elisabeth Oswald, editors. *Advances in Cryptology - EUROCRYPT 2014 - 33rd Annual International Conference on the Theory and Applications of Cryptographic*

Techniques, Copenhagen, Denmark, May 11-15, 2014. Proceedings, volume 8441 of *Lecture Notes in Computer Science*, 2014. Springer.

[245] Svetla Nikova, Christian Rechberger, and Vincent Rijmen. Threshold Implementations Against Side-Channel Attacks and Glitches. In Peng Ning, Sihan Qing, and Ninghui Li, editors, *Information and Communications Security, 8th International Conference, ICICS 2006, Raleigh, NC, USA, December 4-7, 2006, Proceedings*, volume 4307 of *Lecture Notes in Computer Science*, pages 529–545. Springer, 2006.

[246] Kaisa Nyberg. S-boxes and Round Functions with Controllable Linearity and Differential Uniformity. In Preneel [261], pages 111–130.

[247] Kaisa Nyberg, editor. *Fast Software Encryption, 15th International Workshop, FSE 2008, Lausanne, Switzerland, February 10-13, 2008, Revised Selected Papers*, volume 5086 of *Lecture Notes in Computer Science*, 2008. Springer.

[248] Kaisa Nyberg and Lars R. Knudsen. Provable Security Against a Differential Attack. *J. Cryptology*, 8(1):27–37, 1995.

[249] Philippe Oechslin. Making a Faster Cryptanalytic Time-Memory Trade-Off. In Boneh [81], pages 617–630.

[250] European Network of Excellence in Cryptology. eSTREAM: the ECRYPT Stream Cipher Project. http://www.ecrypt.eu.org/stream/. Accessed on May 13, 2015.

[251] National Institute of Standards and Technology. *FIPS PUB 46-3: Data Encryption Standard (DES)*. NIST, 1995.

[252] National Institute of Standards and Technology. *FIPS PUB 197: Announcing the ADVANCED ENCRYPTION STANDARD (AES)*. NIST, 2001.

[253] National Institute of Standards and Technology. *FIPS PUB 180-4: Secure Hash Standard (SHS)*. NIST, 2012.

[254] National Institute of Standards and Technology. *FIPS PUB 202: SHA-3 Standard: Permutation-Based Hash and Extendable-Output Functions*. NIST, 2015.

[255] Kenji Ohkuma. Weak Keys of Reduced-Round PRESENT for Linear Cryptanalysis. In Michael J. Jacobson Jr., Vincent Rijmen, and Reihaneh Safavi-Naini, editors, *Selected Areas in Cryptography, 16th Annual International Workshop, SAC 2009, Calgary, Alberta, Canada, August 13-14, 2009, Revised Selected Papers*, volume 5867 of *Lecture Notes in Computer Science*, pages 249–265. Springer, 2009.

[256] Elisabeth Oswald and Marc Fischlin, editors. *Advances in Cryptology - EUROCRYPT 2015 - 34th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Sofia, Bulgaria, April 26-30, 2015, Proceedings, Part I*, volume 9056 of *Lecture Notes in Computer Science*, 2015. Springer.

[257] Pascal Paillier and Ingrid Verbauwhede, editors. *Cryptographic Hardware and Embedded Systems - CHES 2007, 9th International Workshop, Vienna, Austria, September 10-13, 2007, Proceedings*, volume 4727 of *Lecture Notes in Computer Science*, 2007. Springer.

[258] Jacques Patarin. The "Coefficients H" Technique. In Roberto Maria Avanzi, Liam Keliher, and Francesco Sica, editors, *Selected Areas in Cryptography, 15th International Workshop, SAC 2008, Sackville, New Brunswick, Canada, August 14-15, Revised Selected Papers*, volume 5381 of *Lecture Notes in Computer Science*, pages 328–345. Springer, 2008.

[259] Kostas Pentikousis and Hussein G. Badr. Quantifying the deployment of TCP options - a comparative study. *IEEE Communications Letters*, 8(10):647–649, 2004.

[260] Josef Pieprzyk, editor. *Topics in Cryptology - CT-RSA 2010, The Cryptographers' Track at the RSA Conference 2010, San Francisco, CA, USA, March 1-5, 2010. Proceedings*, volume 5985 of *Lecture Notes in Computer Science*, 2010. Springer.

[261] Bart Preneel, editor. *Fast Software Encryption: Second International Workshop. Leuven, Belgium, 14-16 December 1994, Proceedings*, volume 1008 of *Lecture Notes in Computer Science*, 1995. Springer.

[262] Bart Preneel, René Govaerts, and Joos Vandewalle. Cryptographically secure hash functions: an overview. ESAT Internal Report. KU Leuven, 1989.

[263] Gordon Procter and Carlos Cid. On Weak Keys and Forgery Attacks Against Polynomial-Based MAC Schemes. In Moriai [236], pages 287–304.

[264] Christian Rechberger. On Bruteforce-Like Cryptanalysis: New Meet-in-the-Middle Attacks in Symmetric Cryptanalysis. In Taekyoung Kwon, Mun-Kyu Lee, and Daesung Kwon, editors, *Information Security and Cryptology - ICISC 2012 - 15th International Conference, Seoul, Korea, November 28-30, 2012, Revised Selected Papers*, volume 7839 of *Lecture Notes in Computer Science*, pages 33–36. Springer, 2012.

[265] Thomas Ristenpart and Phillip Rogaway. How to Enrich the Message Space of a Cipher. In Alex Biryukov, editor, *Fast Software Encryption, 14th International Workshop, FSE 2007, Luxembourg, Luxembourg, March 26-28, 2007, Revised Selected Papers*, volume 4593 of *Lecture Notes in Computer Science*, pages 101–118. Springer, 2007.

[266] Ronald Rivest. The MD5 Message-Digest Algorithm. RFC 1321 (Informational), April 1992.

[267] Dorothy Elizabeth Robling Denning. *Cryptography and Data Security*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1982.

[268] Phillip Rogaway. OCB – An Authenticated-Encryption Scheme – Licensing. http://web.cs.ucdavis.edu/~rogaway/ocb/license.htm. Accessed on May 19, 2015.

[269] Phillip Rogaway. Authenticated-encryption with associated-data. In Atluri [33], pages 98–107.

[270] Phillip Rogaway. Efficient Instantiations of Tweakable Blockciphers and Refinements to Modes OCB and PMAC. In Pil Joong Lee, editor, *Advances in Cryptology - ASIACRYPT 2004, 10th International Conference on the Theory and Application of Cryptology and Information Security, Jeju Island, Korea, December 5-9, 2004, Proceedings*, volume 3329 of *Lecture Notes in Computer Science*, pages 16–31. Springer, 2004.

[271] Phillip Rogaway and Thomas Shrimpton. A Provable-Security Treatment of the Key-Wrap Problem. In Serge Vaudenay, editor, *Advances in Cryptology - EUROCRYPT 2006, 25th Annual International Conference on the Theory and Applications of Cryptographic Techniques, St. Petersburg, Russia, May 28 - June 1, 2006, Proceedings*, volume 4004 of *Lecture Notes in Computer Science*, pages 373–390. Springer, 2006.

[272] Phillip Rogaway and David Wagner. A Critique of CCM. Cryptology ePrint Archive, Report 2003/070, 2003.

[273] Phillip Rogaway, Mihir Bellare, John Black, and Ted Krovetz. OCB: a block-cipher mode of operation for efficient authenticated encryption. In Michael K. Reiter and Pierangela Samarati, editors, *CCS 2001, Proceedings of the 8th ACM Conference on Computer and Communications Security, Philadelphia, Pennsylvania, USA, November 6-8, 2001.*, pages 196–205. ACM, 2001.

[274] Bimal K. Roy and Willi Meier, editors. *Fast Software Encryption, 11th International Workshop, FSE 2004, Delhi, India, February 5-7, 2004, Revised Papers*, volume 3017 of *Lecture Notes in Computer Science*, 2004. Springer.

[275] Markku-Juhani O. Saarinen. Cycling Attacks on GCM, GHASH and Other Polynomial MACs and Hashes. In Canteaut [89], pages 216–225.

[276] Markku-Juhani O. Saarinen. CBEAM. Submission to the CAESAR competition, 2014.

[277] Markku-Juhani O. Saarinen. STRIBOB. Submission to the CAESAR competition, 2014.

[278] Joseph Salowey, Abhijit Choudhury, and David A. McGrew. AES Galois Counter Mode (GCM) Cipher Suites for TLS. RFC 5288 (Proposed Standard), August 2008.

[279] Palash Sarkar and Tetsu Iwata, editors. *Advances in Cryptology - ASIACRYPT 2014 - 20th International Conference on the Theory and Application of Cryptology and Information Security, Kaoshiung, Taiwan, R.O.C., December 7-11, 2014. Proceedings, Part I*, volume 8873 of *Lecture Notes in Computer Science*, 2014. Springer.

[280] Yu Sasaki, Yosuke Todo, Kazumaro Aoki, Yusuke Naito, Takeshi Sugawara, Yumiko Murakami, Mitsuru Matsui, and Shoichi Hirose. Minalpher. Submission to the CAESAR competition, 2014.

[281] Bruce Schneier, John Kelsey, Doug Whiting, David Wagner, Chris Hall, and Niels Ferguson. Twofish: a 128-bit block cipher. In *Proceedings of the first AES candidate conference*, 1998.

[282] Kai Schramm and Christof Paar. Higher Order Masking of the AES. In David Pointcheval, editor, *Topics in Cryptology - CT-RSA 2006, The Cryptographers' Track at the RSA Conference 2006, San Jose, CA, USA, February 13-17, 2006, Proceedings*, volume 3860 of *Lecture Notes in Computer Science*, pages 208–225. Springer, 2006.

[283] Pascale Serf. The degrees of completeness, of avalanche effect, and of strict avalanche criterion for MARS, RC6, Rijndael, Serpent, and Twofish with reduced number of rounds, 2000.

[284] Claude E. Shannon. Communication Theory of Secrecy Systems. *Bell System Technical Journal*, 28(4):656–715, 1949.

[285] Danping Shi, Lei Hu, Siwei Sun, Ling Song, Kexin Qiao, and Xiaoshuang Ma. Improved Linear (hull) Cryptanalysis of Round-reduced Versions of SIMON. Cryptology ePrint Archive, Report 2014/973, 2014.

[286] Nigel P. Smart, Vincent Rijmen, Benedikt Gierlichs, Kenneth G. Paterson, Martijn Stam, Bogdan Warinschi, and Gaven Watson. ENISA: Algorithms, Key Sizes and Parameters Report, November 2014.

[287] Gurobi Software. Gurobi Optimizer. http://www.gurobi.com. Accessed on July 31, 2015.

[288] Siwei Sun, Lei Hu, Peng Wang, Kexin Qiao, Xiaoshuang Ma, and Ling Song. Automatic Security Evaluation and (Related-key) Differential Characteristic Search: Application to SIMON, PRESENT, LBlock, DES(L) and Other Bit-Oriented Block Ciphers. In Sarkar and Iwata [279], pages 158–178.

[289] Siwei Sun, Lei Hu, Meiqin Wang, Peng Wang, Kexin Qiao, Xiaoshuang Ma, Danping Shi, Ling Song, and Kai Fu. Constructing Mixed-integer Programming Models whose Feasible Region is Exactly the Set of All Valid Differential Characteristics of SIMON. Cryptology ePrint Archive, Report 2015/122, 2015.

[290] Yosuke Todo and Kazumaro Aoki. FFT Key Recovery for Integral Attack. In Dimitris Gritzalis, Aggelos Kiayias, and Ioannis G. Askoxylakis, editors, *Cryptology and Network Security - 13th International Conference, CANS 2014, Heraklion, Crete, Greece, October 22-24, 2014. Proceedings*, volume 8813 of *Lecture Notes in Computer Science*, pages 64–81. Springer, 2014.

[291] Harshal Tupsamudre, Shikha Bisht, and Debdeep Mukhopadhyay. Differential Fault Analysis on the families of SIMON and SPECK ciphers. Cryptology ePrint Archive, Report 2014/267, 2014.

[292] Serge Vaudenay. Security Flaws Induced by CBC Padding - Applications to SSL, IPSEC, WTLS ... In Knudsen [190], pages 534–546.

[293] Ning Wang, Xiaoyun Wang, Keting Jia, and Jingyuan Zhao. Differential Attacks on Reduced SIMON Versions with Dynamic Key-guessing Techniques. Cryptology ePrint Archive, Report 2014/448, 2014.

[294] Qingju Wang, Zhiqiang Liu, Kerem Varici, Yu Sasaki, Vincent Rijmen, and Yosuke Todo. Cryptanalysis of Reduced-Round SIMON32 and SIMON48. In Willi Meier and Debdeep Mukhopadhyay, editors, *Progress in Cryptology - INDOCRYPT 2014 - 15th International Conference on Cryptology in India, New Delhi, India, December 14-17, 2014, Proceedings*, volume 8885 of *Lecture Notes in Computer Science*, pages 143–160. Springer, 2014.

[295] Dominic J. A. Welsh. *Codes and Cryptography*. Clarendon Press, 1988.

[296] Doug Whiting, Russel Housley, and Niels Ferguson. Counter with CBC-MAC (CCM). RFC 3610 (Informational), September 2003.

[297] Robert S. Winternitz. Producing a One-Way Hash Function from DES. In David Chaum, editor, *Advances in Cryptology, Proceedings of CRYPTO '83, Santa Barbara, California, USA, August 21-24, 1983.*, pages 203–207. Plenum Press, New York, 1983.

[298] Robert S. Winternitz. A Secure One-Way Hash Function Built from DES. In *Proceedings of the 1984 IEEE Symposium on Security and Privacy, Oakland, California, USA, April 29 - May 2, 1984*, pages 88–90. IEEE Computer Society, 1984.

[299] Hongjun Wu and Tao Huang. JAMBU. Submission to the CAESAR competition, 2014.

[300] Shengbao Wu and Mingsheng Wang. Security Evaluation against Differential Cryptanalysis for Block Cipher Structures. Cryptology ePrint Archive, Report 2011/551, 2011.