Structured Randomness

Daniël Kuijsters

Institute for Computing
and Information Sciences

RADBOUD
UNIVERSITY
PRESS

Radboud
Dissertation
Series

Structured Randomness

Daniël Wilhelmus Cornelis Kuijsters

Structured Randomness
Daniël Kuijsters

Radboud Dissertation Series
ISSN: 2950-2772 (Online); 2950-2780 (Print)

Published by RADBOUD UNIVERSITY PRESS
Postbus 9100, 6500 HA Nijmegen, The Netherlands
www.radbouduniversitypress.nl

Design: Daniël Kuijsters
Cover: Proefschrift AIO | Guntra Laivacuma
Printing: DPN Rikken/Pumbo

ISBN: 9789465151083
DOI: 10.54195/9789465151083
Free download at: https://doi.org/10.54195/9789465151083

RADBOUD
UNIVERSITY
PRESS

Structured Randomness

Proefschrift ter verkrijging van de graad van doctor
aan de Radboud Universiteit Nijmegen
op gezag van de rector magnificus prof. dr. J.M. Sanders,
volgens besluit van het college voor promoties
in het openbaar te verdedigen op

donderdag 3 juli 2025
om 10.30 uur precies

door

Daniël Wilhelmus Cornelis Kuijsters

Promotor:

Prof. dr. Joan Daemen

Manuscriptcommissie:

Prof. dr. Peter Schwabe (voorzitter)
Prof. dr. Lilya Budaghyan (Universitetet i Bergen, Noorwegen)
Prof. dr. Gregor Leander (Ruhr-Universität Bochum, Duitsland)
Dr. Anne Canteaut (Inria, Frankrijk)
Dr. Yu Sasaki (NTT R&D, Japan)

Acknowledgements

This thesis would not have come together without the help and support of many people. I am grateful to my supervisor for his guidance throughout the process. I also appreciate the ideas, questions, and conversations shared by colleagues, collaborators, and others in the academic community. I would like to thank the committee for taking the time to read and evaluate my thesis. Finally, I thank my family and friends for their support along the way. This research was funded by the ESCADA project.

Contents

Part I

Theory

This part is composed of an introduction to the area of research and to the theory upon which the research chapters are based.

I Introduction

This is a thesis about the design and analysis of *secret-key cryptography*. In accordance with standard practice, we begin by dissecting each of these words and setting the scene.

The word *cryptography* is derived from the Ancient Greek words κρυπτός, which translates to "hidden" or "secret", and γράφειν, which translates to "writing" [21]. Indeed, since ancient times, cryptography has been used to make sure that the contents of a message that is transmitted between a sender and a receiver through an insecure channel are not disclosed to an unauthorized entity, the *adversary*. This objective is called *confidentiality*. Usually, a receiver also needs to detect if a message has been tampered with. Hence, cryptography is also used to protect against message forgery. This objective is called *integrity*. A broader objective that encompasses integrity is *authentication*, which additionally verifies the origin of the message. Further objectives can be defined, e.g., preventing denial-of-service attacks (to protect *availability* of systems), stopping traffic analysis, etc. While these are important for security, we do not consider them in this thesis.

We distinguish between two types of communication. In the first type of communication, sender and receiver are separated in *space*. For example, the sender may be a client, the receiver may be a server and the message may be a private datagram that the client sends to the server over the public Internet. In the second type of communication, sender and receiver represent the same entity that communicates with itself over *time*. For example, the sender may be storing the message, a private file, on a disk and try to recover it at a later time.

Cryptography achieves its objectives by transforming the message in a way that is difficult to predict by the adversary. In the general setting, a message is divided into two parts that form the inputs to the transformation: a part for which both confidentiality and integrity are required, and a part for which only integrity is needed. The part requiring confidentiality and integrity is called the *plaintext*, and the part requiring only integrity is called *associated data*.

To achieve confidentiality, the plaintext is transformed into a *ciphertext* through a process called *encryption*; the inverse process, which recovers the original plaintext, is called *decryption*. The methods used for encryption and decryption are specified by a system called a *cryptosystem*, an *encryption scheme*, or a *cipher*. In this context, the goal of a cryptosystem is to ensure that an adversary cannot (efficiently) learn any information about the plaintext from public information, such as the ciphertext, except for some properties such as length or format, depending on the scheme.

To achieve integrity, there exist two popular approaches. The first approach is for the cryptosystem to produce a short piece of data called an *authenticator*, which serves as a fingerprint for both the plaintext and the associated data. An authenticator is also called a *tag* or a *message authentication code* (MAC). The

second approach is to treat the entire message as plaintext and add a highly structured part to it. With high probability, a modified ciphertext decrypts to a plaintext for which the structured part is modified.

To make this possible, sender and receiver need to have access to a random element about which the adversary has more uncertainty than they do. Hence, for simplicity, we make the assumption that the sender and receiver have access to the same *known* sequence of randomly and uniformly generated symbols from some alphabet set, e.g., a string of bits. We call this sequence the *secret*, *key*, or *secret key*, and call this setting *secret-key* cryptography. Although secret-key cryptography primarily concerns systems with a shared secret, its building blocks and techniques are also employed in settings without one, for example in the construction of hash functions.

We assume that all details of the cryptosystem are known to the adversary. This is called *Kerckhoffs's principle* [32]. Accordingly, we may think of secret-key cryptography as a tool for reducing security problems to the problem of protecting secret keys.

We do not concern ourselves with how this secret was generated and shared in the first place. This *key distribution* is a problem in itself. In practice, it often (but not necessarily) makes use of *public-key* cryptography, e.g., the Diffie-Hellman key exchange protocol [23].

Different cryptosystems can achieve the same security objectives. The deciding factor frequently revolves around *implementation cost*. In the remainder of this thesis, we explore how such cryptosystems are designed, analyzed, and optimized for real-world constraints.

2 Preliminaries

This chapter outlines the fundamental mathematical and theoretical concepts necessary for understanding the research chapters. We assume that the reader has a working knowledge of the most basic notions of set theory. Moreover, we assume that the reader has had exposure to algebraic structures and probability theory.

2.1 Structure and randomness

This section provides the language to discuss structure and randomness through algebra, probability theory, and discrete mathematics. For (universal) algebra, we refer to [1] and [12]. For probability theory as it applies to secret-key cryptography, a good summary is found in the appendix of [7]. For general discrete mathematics, we refer to [39].

Functions between sets. Let S, T, and U be sets. A *function* or *mapping* f from S to T is a subset of $S \times T$ such that for each $s \in S$ there is exactly one $t \in T$ with $(s, t) \in f$. This is abbreviated as $f \colon S \to T$ and $f(s) = t$. In some cases, we do not want to give a name to a function. An *anonymous* function is defined by its assignment rule. We write this as $s \mapsto t$. If S equals T, we may call it a *transformation*. We write $\mathrm{Maps}[S, T]$ for the set of all functions from S to T. For a subset $A \subseteq S$, the *restriction* of f to A is the function $f|_A \colon A \to T$ given by $a \mapsto f(a)$ for all $a \in A$. The kernel of f is the set $\ker(f) := \{(a, b) \in S \times S : f(a) = f(b)\}$. If $\ker(f) = \{(s, s) : s \in S\}$, then f is called *injective* or *one-to-one*. The f-image of A is defined as $f(A) := \{f(a) : a \in A\}$. If $f(S) = T$, then f is called *surjective* or *onto*. The function is called *bijective* if it is both injective and surjective. A bijective transformation of a finite set S is called a *permutation*. The set of all permutations of S is denoted as $\mathrm{Perms}[S]$. The *domain* of f is S, its *codomain* is T, and its *range* is $f(S)$.

Let $f \colon S \to T$ and $g \colon T \to U$, then the *composition* of f and g is the function $g \circ f \colon S \to U$ defined by $(g \circ f)(s) = g(f(s))$ for all $s \in S$. The identity function e_S on a set S is defined by $\mathrm{e}_S(s) = s$ for all $s \in S$. If f is bijective, then there is a unique *inverse* function $f^{-1} \colon T \to S$ defined by $f^{-1} \circ f = \mathrm{e}_S$. In that case, we also call f *invertible*. An invertible function for which f^{-1} equals f is called an *involution*.

Two functions are the same if they have the same domain, the same range, and are equal as sets.

Predicates. A predicate ϕ is a mathematical assertion that contains variables and that may be true or false depending on the assignment of values from a set S to these variables. Denoting true as $\top$ and false as $\bot$, a predicate is thus a function $\phi \colon S \to \{\top, \bot\}$.

Orders. Let S be a set. A *partial order* $\leq$ on S is a subset of $S \times S$ that satisfies the following properties:

- (Reflexivity) For each $s \in S$, $(s, s) \in \leq$.
- (Antisymmetry) For each $r, s \in S$, $(r, s) \in \leq$ and $(s, r) \in \leq$ implies $s = r$.
- (Transitivity) For each $r, s, t \in S$, $(r, s) \in \leq$ and $(s, t) \in \leq$ implies $(r, t) \in \leq$.

For example, $\leq = \{(0, 0), (0, 1), (1, 1)\}$ is a partial order on the subset $\{0, 1\}$ of the integers. Instead of writing $(r, s) \in \leq$, we use the notation $r \leq s$.

A *total order* $\leq$ on S is a partial order on S with the additional property that $r \leq s$ or $s \leq r$ for all $r, s \in S$. In other words, any two elements are comparable.

A *well-order* $\leq$ on S is a total order on S with the property that every nonempty subset of S has a least element with respect to $\leq$.

An example of a total order that is not a well-order is the usual order $\leq$ on the real numbers.

Function families and sequences. Let I be a set. A *family of elements* of S is a function from I to S. Technically, families of elements and functions are the same object, except that we view them differently. Write s_i for the element of S corresponding to $i \in I$. We denote the family as $s = (s_i : i \in I)$, we call i the *index*, and we call I the *index set*. We call the element s_i the ith *component* of s.

If $I = \{0, \dots, n-1\}$ for some positive integer $n \geq 1$, then we call s a finite *sequence* or *tuple*. We may write n-tuple if we want to make the number of elements explicit. We identify 1-tuples with the single element of which they are comprised. The total order on I (inherited by the integers) induces an order on the elements of s. Hence, it makes sense to speak of a smallest or largest element of s that satisfies some predicate. To avoid confusion in the case that S is already ordered, we write leftmost for smallest and rightmost for largest with respect to the order that is determined by I.

If $s \colon I \to (X \to Y)$ is a family of functions from a set X to a set Y, then s can also be seen as a function $F \colon I \times X \to Y$. Indeed, $F(i, x) := s_i(x)$ for all $x \in X$. By abuse of language, we may call s invertible if s_i is invertible for each $i \in I$.

Equivalence relations, partitions, and quotients. An *equivalence relation* $\sim$ on a set S is a subset of $S \times S$ that satisfies the following properties:

- (Reflexivity) For each $s \in S$, $(s, s) \in \sim$.
- (Symmetry) For each $r, s \in S$, $(r, s) \in \sim$ implies $(s, r) \in \sim$.
- (Transitivity) For each $r, s, t \in S$, $(r, s) \in \sim$ and $(s, t) \in \sim$ implies $(r, t) \in \sim$.

Instead of writing $(r, s) \in \sim$, we use the notation $r \sim s$. The *equivalence class* of s modulo $\sim$ is the set $s/\sim := \{t \in S : s \sim t\}$. We may also write this as $[s]_\sim$ or $[s]$ if the relation used it clear from the context. We can pick any $t \in s/\sim$ as a *representative* of that class. The *quotient space* is the set $S/\sim := \{s/\sim : s \in S\}$. The *quotient map* is the function $\phi_\sim \colon S \to S/\sim$ defined by $\phi_\sim(s) = s/\sim$ for all $s \in S$. Importantly, the

equivalence classes form a *partition* of S. That is, $S = \cup_{B \in S/\sim} B$ and for all $B, C \in S/\sim$ such that $B \neq C$, we have $B \cap C = \emptyset$. Each partition of a set arises in this way. Hence, if we want to study a function $f : S \to T$ with the property that $f(r) = f(s)$ whenever $r \sim s$, then we need only to study it on a set of representatives. A function with this property is called an *invariant* of $\sim$.

Operations. Let S be a nonempty set. Define $S^0 := \{\emptyset\}$ and $S^n := S^{n-1} \times S$ for integers $n \geq 1$. An n-ary *operation* on S is a function $f : S^n \to S$. The number n is called the *arity* of f. A *finitary* operation is an n-ary operation for some fixed n. For example, a 0-ary operation is a function from $\{\emptyset\}$ to S, which can be identified with an element of S. In practice, we mostly work with 2-ary (or *binary*) operations.

Congruence relations. Let f be an n-ary *operation* on a set S for some positive integer $n \geq 1$. Moreover, let $a_i, b_i \in S$ for $i = 0, \ldots, n-1$. An equivalence relation $\sim$ on S is called a *congruence relation* with respect to f if $f(a_0, \ldots, a_{n-1}) \sim f(b_0, \ldots, b_{n-1})$ whenever $a_i \sim b_i$ for $i = 0, \ldots, n-1$. For example, fix a positive integer $m \geq 1$. Define a relation $\sim$ on the integers by

$$a \sim b \text{ if and only if } m \mid (a - b).$$

Then $\sim$ is a congruence relation with respect to the usual addition and multiplication on the integers. Typically, this is written as $a = b \pmod{m}$.

Algebraic structures. An *algebraic structure* is an ordered pair $A = (S, F)$ where S is the *underlying set* and $F = (f_o : o \in I)$ is a family of finitary operations on S indexed by I. The index set contains *operation symbols* and for each such symbol $o \in I$ we write o^A for the operation in A indexed by o. Typically, we use the same symbol to refer to the algebraic structure and its underlying set and assume that the operations are understood from the context. We call A *finite* if S is a finite set. The *signature* of A is the function $\sigma : I \to \mathbb{Z}$ where $\sigma(o)$ is equal to the arity of o^A for each $o \in I$. Two algebraic structures are called *similar* if they have the same signature.

Let S and T be similar algebraic structures. Then S is a *substructure* of T if $S \subseteq T$ and if $o^S = o^T|_S$ for each $o \in I$.

Direct products. Let $(A_i : i \in I)$ be a family of algebraic structures with the same signature. The *direct product* of $(A_i : i \in I)$ is the algebraic structure $A = (\prod_{i \in I} A_i, (f_o : o \in J))$ with the same signature. The family of operations $(f_o : o \in J)$ is defined as follows. Put $n_o := \sigma(o)$. For each $i \in I$, for each $o \in J$, and for all $a_0, \ldots, a_{n_o - 1} \in \prod_{i \in I} A_i$, we have

$$(o^A(a_0, \ldots, a_{n_o-1}))_i = o^{A_i}(a_{0,i}, \ldots, a_{(n_o-1),i}) .$$

For each $i \in I$ there is a projection operator $\mathrm{Proj}_i : \prod_{i \in I} A_i \to A_i$ that is defined by $\mathrm{Proj}_i(a) = a_i$.

Homomorphisms between algebraic structures. Let S and T be two similar algebraic structures. Put $n_o := \sigma(o)$. A function $f\colon S \to T$ is called a *homomorphism* if for every $o \in I$ we have

$$f(o^S(a_0, \dots, a_{n_o-1})) = o^T(f(a_0), \dots, f(a_{n_o-1}))$$

for all $a_0, \dots, a_{n_o-1} \in S$. It is called an *isomorphism* if f is a bijection. In this case, we call S and T isomorphic and denote this as $S \simeq T$.

Suppose that $f\colon S \to T$ is a homomorphism. Let R be a congruence relation on S. Let ϕ_R be the quotient map from S to S/R. Then $\ker(f)$ is a congruence relation, the map ϕ_R is a homomorphism, and the unique function g from $S/\ker(f)$ to T satisfying $g \circ \phi_{\ker(f)} = f$ is an isomorphism.

Semigroups. A *semigroup* is an algebraic structure $(S, \cdot)$ such that $(a \cdot b) \cdot c = a \cdot (b \cdot c)$ for all $a, b, c \in S$

Monoids. A *monoid* is an algebraic structure $(M, \cdot, e)$ such that $(M, \cdot)$ is a semigroup and $a \cdot e = e \cdot a = a$ for all $a \in M$

Groups. A *group* is an algebraic structure $(G, \cdot, {}^{-1}, e)$ such that $(G, \cdot, e)$ is a monoid and $g \cdot g^{-1} = g^{-1} \cdot g = e$ for all $g \in G$. A group is called *abelian* if $g \cdot h = h \cdot g$ for all $g, h \in G$. The *exponent* of a group is the smallest positive integer $n \geq 1$ such that $g^n = e$ for all $g \in G$, where $g^n = g \cdot \dots \cdot g$ (n times). If such an n does not exist, the exponent is defined as 0. For each $g \in G$, the *coset* of a subgroup H by g is the set $g \cdot H := \{g \cdot h : h \in H\}$. The subgroup H defines an equivalence relation on G. Indeed, define $g \sim h$ if $g^{-1} \cdot h \in H$. The equivalence classes of this relation are the cosets of H. The *order* of a group is its cardinality as a set. If G has finite order, then $g^{|G|} = e$ for all $g \in G$ by Lagrange's theorem. We will make implicit use of this identity when we perform arithmetic on exponents.

Rings. A *ring* is an algebraic structure $(R, +, \cdot, -, 0, 1)$ such that $(R, +, -, 0)$ is an abelian group, $(R, \cdot, 1)$ is a monoid, and for all $r, s, t \in R$ we have

$$r \cdot (s + t) = r \cdot s + r \cdot t\,, \text{and}$$
$$(r + s) \cdot t = r \cdot t + s \cdot t\,.$$

The ring R is called *commutative* if $r \cdot s = s \cdot r$ for all $r, s \in R$. The *characteristic* of R is the exponent of the additive group of R.

An *ideal* of R is a subset $I \subseteq R$ that is a subgroup of $(R, +, -, 0)$ and satisfies $r \cdot s \in I$ and $s \cdot r \in I$ for each $r \in R$ and $s \in I$. The set $\sqrt{I} := \{r \in R : r^n \in I \text{ for some } n \geq 1\}$ is called the *radical* of I. If $I = \sqrt{I}$, then we say that I is radical. The ideal *generated* by a subset $S \subseteq R$ is the smallest ideal that contains S. We denote it as $\langle S \rangle$.

There is a correspondence between ideals of R and congruence relations on R. On the one hand, if I is an ideal, then we can define a congruence relation $\sim$ on R with $I = 0/\sim$ by $r \sim s$ if and only if $r - s \in I$ for all $r, s \in R$. On the other hand, if $\sim$ is a congruence relation on R, then $0/\sim$ is an ideal of R.

The standard example of a ring is the set $\mathbb{Z}$ of integers with the usual addition, subtraction, and multiplication. In many cases, we consider subsets of $\mathbb{Z}$ that do not form a ring (or even a group). For example, we write $\mathbb{Z}_{\geq 0}$ for the set of nonnegative integers and $\mathbb{Z}_{>0}$ for the set of positive integers.

Modules over a ring. Suppose that R is a ring. An *R-module* is an algebraic structure $(M, +, -, 0, (f_r)_{r\in R})$ such that $(M, +, -, 0)$ is an abelian group and for all $a, b \in M$ and $r, s, t \in R$ we have

$$\begin{aligned} f_{rs}(a) &= f_r(f_s(a)) \,, \\ f_r(a+b) &= f_r(a) + f_r(b) \,, \\ f_{r+s}(a) &= f_r(a) + f_s(a) \,, \\ f_1(a) &= a \,. \end{aligned}$$

The *span* of a subset $S \subseteq M$, denoted as $\mathrm{Span}(S)$, is the subset of all finite R-linear combinations of elements of S. We call S *linearly independent* if for all $n \geq 1$, $r_1, \dots, r_n \in R$, and distinct $m_1, \dots, m_n \in S$, we have

$$r_1 m_1 + \cdots + r_n m_n = 0 \text{ if and only if } r_1 = \cdots = r_n = 0 \,.$$

If M contains a finite subset S with $\mathrm{Span}(S) = M$, then M is called *finitely generated*. We call S a *basis* if $\mathrm{Span}(S) = M$ and S is linearly independent. If M has a basis, then it is called a *free* R-module.

A finitely generated free R-module M is isomorphic to R^n for some integer $n \geq 1$. The *standard basis* of R^n is the set $\{e_i^n : 1 \leq i \leq n\}$ with

$$e_{i,j}^n := \begin{cases} 1 & \text{if } i = j \,, \\ 0 & \text{if } i \neq j \,. \end{cases}$$

Let M be a finitely generated free R-module with basis $B = \{b_1, \dots, b_n\}$. Moreover, let N be a finitely generated R-module with generating set $S = \{s_1, \dots, s_m\}$. Suppose that $\phi \colon M \to N$ is a homomorphism (also called an R-linear map), then it can be represented by an *$m \times n$ matrix* A with coefficients in the ring R. The entries of A are determined by applying ϕ to the elements of B and writing the images in terms of the elements of S. In particular, the matrix depends on the choice of both B and S. The *transpose* of A, denoted as $A^\top$, is defined by $A^\top_{i,j} = A_{j,i}$. Typically, we identify the function ϕ and the matrix A.

Fields, vector spaces, and affine subspaces. A *field* is a commutative ring F with the property that for every $a \in F$ with $a \neq 0$ there exists a $b \in F$ such that $ab = 1$. A *vector space* is a free F-module that we denote as V. Its elements are called *vectors*. The cardinality of any basis of V is called its *dimension*; it can be finite or infinite. Substructures of V are called *(linear) subspaces*. An *affine subspace* of V is a coset A of a subspace U of V. In other words, $A = v + U$ for some $v \in V$.

Table 2.1: Truth table of $f(x) = (x_0, x_0x_1)$.

x	00	01	10	11
$f(x)$	00	00	10	11

Polynomial rings. A *monomial* in the variables $x_1, \dots, x_n$ is a product

$$x^u := (x_1, \dots, x_n)^{(u_1,\dots,u_n)} := \prod_{i=1}^{n} x_i^{u_i}$$

where the $u_i \geq 0$ are nonnegative integers. The *degree* of x^u is equal to $\sum_{i=1}^{n} u_i$. If R is a ring, we can form finite R-linear combinations of monomials. The resulting objects are called *polynomials*. In other words, a polynomial in the variables $x_1, \dots, x_n$ with coefficients in R is an expression of the form $\sum_{u \in \mathbb{Z}_{\geq 0}^n} a_u x^u$ with $a_u \in R$ and only finitely many of the a_u are non-zero. This ring is denoted as $R[x_1, \dots, x_n]$. The degree of a polynomial is the largest of the degrees of its monomials. It is defined as $-\infty$ if the polynomial is zero. A polynomial is called *homogeneous* if its monomials all have the same degree.

Finite fields. The order of any finite field is of the form $q = p^n$ for some prime number p and positive integer $n \geq 1$. Because all finite fields of order q are isomorphic, we denote any one of them by $\mathbb{F}_q$. In the case that n equals 1, the ring $\mathbb{Z}/\langle p \rangle$ is a finite field of order p.

A nonconstant polynomial $f \in \mathbb{F}_p[x]$ is called *irreducible* over $\mathbb{F}_p$ if it is impossible to find nonconstant polynomials $g, h \in \mathbb{F}_p[x]$ such that $f = gh$. Given an irreducible polynomial $f \in \mathbb{F}_p[x]$ of degree $n \geq 2$, the ring $\mathbb{F}_p[x]/\langle f \rangle$ is a finite field of order p^n.

For both constructions of finite fields, we identify the set of equivalence classes with a set of representatives. For example, we take $\{0, \dots, p-1\}$ as the underlying set of $\mathbb{F}_p$.

Boolean functions. Functions $\mathbb{F}_2^n \to \mathbb{F}_2$ are called *Boolean* functions. More generally, functions $\mathbb{F}_2^n \to \mathbb{F}_2^m$ are called *vectorial* Boolean functions. They are completely specified by a *truth table*, which is a tabular array, where each column corresponds to an input and the corresponding output of the function. For example, the function $x \mapsto (x_0, x_0x_1)$ is specified by the truth table in Table 2.1.

Strings. Let S be a nonempty finite set. In this context, its elements are called *symbols*. Suppose a designated *zero symbol* exists. A *string* over S is a finite sequence of symbols from S. The *Hamming weight* of a string is equal to the number of its symbols that differ from the zero symbol. The set of strings over S of length $n \geq 0$ is denoted as S^n. The (unique) string of length 0 has no symbols. We denote it as ϵ. The set of all strings over S of length at most L is equal to $S^{\leq L} := \cup_{n=0}^{L} S^n$. The set of all strings over S is denoted as $S^* := \cup_{n=0}^{\infty} S^n$. Moreover, the set $S^* \setminus \{\epsilon\}$ is denoted as S^+. If s and t are two strings, then their concatenation is denoted as $s \parallel t$. A string s is a *substring* of a string t if there exist strings u and v (possibly of length 0) such that $t = u \parallel s \parallel v$. In applications, a string $s \in S^{\leq L}$ is often divided into substrings of a fixed length, called *blocks*. We may need to *pad* s to align it with the block

size by concatenating it with a padding string $p(s)$, which is typically dependent on the length of s. For example, the padding might involve appending a nonzero symbol followed by as many zero symbols as needed to reach the next multiple of the block size. In this thesis, we often assume that S is a finite field $\mathbb{F}_q$ with q elements. Hence, we will typically view a string $s \in S^n$ as a vector in $\mathbb{F}_q^n$.

Finite probability spaces. We model a random process with finitely many outcomes as a finite probability space. A *finite probability space* is an ordered pair (Ω, μ) consisting of a nonempty finite set Ω and a probability measure $\mu \colon \mathcal{P}(\Omega) \to [0, 1]$. That is, μ satisfies

- $\mu(\Omega) = 1$, and
- $\mu(A \cup B) = \mu(A) + \mu(B)$ for any two disjoint subsets A and B of Ω.

The set Ω is called the *sample space* and its elements are the possible outcomes of the random process. Subsets of Ω are called *events*. An event occurs if the outcome of the random process is included in the event. The *probability* that the event A occurs is $\mu(A)$.

Let B be an event with $\mu(B) > 0$ and define $\mu(A \mid B) \coloneqq \mu(A \cap B)/\mu(B)$. This is called the *conditional probability* that the event A occurs *given* that B occurs. With this definition, $(\Omega, A \mapsto \mu(A \mid B))$ is a finite probability space.

Two events are said to be *independent* if $\mu(A \cap B) = \mu(A)\mu(B)$.

Random elements. Let S be a nonempty finite set. A function $X \colon \Omega \to S$ is called a *random element* of S. If S contains "objects", we may abbreviate "random element of S" as "random object." For example, a random string is a random element of the set of strings. As the name suggests, we think of X as an element of S and use it in expressions as such. We abstract away the process that is responsible for producing the random element.

For any predicate $\phi \colon S \to \{\top, \bot\}$, we write $\phi(X)$ for the event $\{\omega \in \Omega : \phi(X(\omega))\}$. We define the *distribution* of X as the (induced) probability measure $\mu_X(B) \coloneqq \mu(X \in B)$ for all $B \subseteq S$. This makes S into a finite probability space. The random element X is called *uniform* if $\mu_X(B) = |B|/|S|$. We denote a uniform random element of S as $X \xleftarrow{\$} S$.

Graphs. A *graph* is an ordered pair $G = (V, E)$ of disjoint sets that satisfy $V \cap E = \emptyset$ and $E \subseteq \{\{u, v\} : u \in V, v \in V\}$. The elements of V are its *vertices* and the elements of E are its *edges*. For an edge $\{u, v\}$, we call the nodes u and v its ends. To depict a graph, we draw its vertices as dots and connect two dots with a line if they form an edge. A *path* P is a graph of the form $V = \{v_0, \dots, v_{n-1}\}$ and $E = \{\{v_0, v_1\}, \dots, \{v_{n-2}, v_{n-1}\}\}$. We say that P is a path from v_0 to v_{n-1}. A graph $G' = (V', E')$ is called a *subgraph* of $G = (V, E)$ if $V' \subseteq V$ and $E' \subseteq E$. A graph is called *connected* if it contains (as a subgraph) a path from u to v for every $u, v \in V$. Let $n \geq 2$ be an integer. A graph is called *n-partite* if V can be partitioned into n subsets $V_0, \dots, V_{n-1}$ called *levels* such that every edge has its ends in different levels. In the case that n equals 2, then we call the graph *bipartite*. If we only allow edges between V_i and V_{i+1} for each $i = 0, \dots, n-2$, then we call the n-partite graph *ordered*, although this is not standard terminology.

2.2 Provable security

In this section, we present two cryptosystems that are *provably secure* for some notion of security. Importantly, their security analysis makes no assumptions about the computation time budget of an adversary.

2.2.1 Secret-key encryption

Fix a finite abelian group G. Suppose that the sender and receiver want to communicate a message $m \in G$ over an insecure communication channel. How do we guarantee the confidentiality of the message? To that end, the sender and receiver agree on a uniform random secret $k \xleftarrow{\$} G$ for use as a so-called *one-time pad* [42]. The sender encrypts the message m as $c := m + k$ and transmits the ciphertext c. The receiver recovers m by computing $m = c - k$.

How do we formalize that the one-time pad cryptosystem achieves confidentiality of the message? Here is one attempt.

We view any function f as an *oracle* that prints $f(x)$ when *queried* with an input x. We allow queries to be *adaptive*, i.e., the ith query may depend on the $(i-1)$th query, for some positive integer $i \geq 1$. In practice, the number of times that f may be queried is upper bounded by some positive integer $n \geq 1$. If this is the case, we refer to the oracle as an n-time oracle. An *oracle algorithm* is an algorithm that uses one or more oracles during its execution and returns either 0 or 1. Throughout this thesis, we model adversaries as oracle algorithms, which may be probabilistic.

Let $\mathcal{O}$ and $\mathcal{P}$ be two l-tuples of oracles for some integer $l \geq 1$ with the property that for each $i = 0, \ldots, l-1$, the oracles $\mathcal{O}_i$ and $\mathcal{P}_i$ have the same domain and codomain. Once we introduce the notion of pseudorandomness, it will become apparent why we sometimes give $\mathcal{A}$ access to multiple oracles. Based on the outcome of a random experiment in which a fair coin is flipped, an adversary $\mathcal{A}$ is given oracle access to either $\mathcal{O}$ or $\mathcal{P}$. The goal of $\mathcal{A}$ is to determine what the outcome was. Loosely speaking, its output value, say d, encodes a binary statement of the form "I believe the outcome of the experiment to be d." The quality of $\mathcal{A}$ is *measured* by the *$\mathcal{A}$-distance* between $\mathcal{O}$ and $\mathcal{P}$, which is defined as

$$\Delta_{\mathcal{A}}(\mathcal{O}, \mathcal{P}) := \left|\Pr[\mathcal{A}(\mathcal{O}) = 1] - \Pr[\mathcal{A}(\mathcal{P}) = 1]\right|.$$

Assume that the adversary is capable of choosing the message that is encrypted. Hence, fix any message. From the point of view of the adversary, the corresponding ciphertext is a random element, because it is a function of the uniform random secret and the message. By eavesdropping on the channel, the adversary is *sampling* from the distribution of this ciphertext. If different messages induce *identical* ciphertext distributions, then the ciphertext gives no information on the message.

Given a one-time pad $k \xleftarrow{\$} G$, let $m \mapsto m + k$ for all $m \in G$ be the corresponding encryption function. Moreover, given a uniform random element $r \xleftarrow{\$} G$, let $m \mapsto r$ for all $m \in G$ be a uniform random constant function. Importantly, we view both functions as *one-time* oracles. The *advantage* of $\mathcal{A}$ in

distinguishing between a ciphertext of the one-time pad cryptosystem and a uniform random element of G is defined as

$$\mathrm{Adv}^{\text{random ciphertexts}}_{m\mapsto m+k}(\mathscr{A}) := \Delta_{\mathscr{A}}(m \mapsto m+k, m \mapsto r)\,,$$

where the probabilities are computed with respect to $k, r \xleftarrow{\$} G$. We claim that the advantage of any adversary is 0. Indeed, for all $m, s \in G$, we have

$$\begin{aligned}\Pr[c = s] &= \Pr[m + k = s] \\ &= \Pr[k = s - m] \\ &= 1/|G|\,.\end{aligned}$$

In words, the distribution of the ciphertext is uniform for any given message. It does not matter whether the adversary has oracle access to $m \mapsto m + k$ or $m \mapsto r$; in both cases it receives a uniform random element. As a consequence, the advantage is 0.

To conclude, we illustrate the importance of never reusing the one-time pad for multiple messages. Suppose that the adversary knows two messages $m_0, m_1 \in G$ that are encrypted with the same one-time pad k. It follows that

$$c_1 - c_0 = (m_1 + k) - (m_0 + k) = m_1 - m_0\,.$$

Hence, knowing either m_0 or m_1 immediately reveals the value of the other.

2.2.2 Secret-key message authentication

In addition to eavesdropping, we allow the adversary to actively corrupt the message in the communication channel. To address this problem, sender and receiver compute a one-time authenticator in the following way. We assume that the message is a polynomial $m \in \mathbb{F}_q[x]$ with $m(0) = 0$ and $\deg(m) \leq d$ for some positive integer $d \geq 1$. The sender and receiver agree on a pair of uniform random secrets $(r, s) \xleftarrow{\$} \mathbb{F}_q \times \mathbb{F}_q$. The sender computes the authenticator $a := m(r) + s$ and sends (m, a) to the receiver.

The security of the authenticator is closely related to the probability of the adversary winning the following game.

MAC security

- The adversary chooses a message $m \in \mathbb{F}_q[x]$ and gives it to the challenger.
- The challenger computes secrets $(r, s) \xleftarrow{\$} \mathbb{F}_q \times \mathbb{F}_q$, the authenticator $a := m(r) + s$, and gives a to the adversary.
- The adversary computes (m', a') for a new message $m' \neq m$.

The adversary wins the game if (m', a') is a valid message-authenticator pair, i.e., if $a' = m'(r) + s$.

From the point of view of the adversary, given the pair (m, a), each of the possible values for r is equally likely. This is due to the addition of the one-time pad s. A forgery attempt (m', a') with $m' \neq m$ succeeds if $m'(r)+s-a' = m(r)+s-a$. Equivalently, it succeeds if $m'(r)-m(r)+a-a' = 0$. This is a polynomial in the variable r of degree at most d. Hence, it has at most d roots. It follows that the probability of correctly guessing r is at most d/q. Sender and receiver can make this probability arbitrarily small by choosing a suitable q.

2.2.3 Multiple messages

Suppose that sender and receiver want to protect the integrity of an n-tuple of messages. To that end, they share a secret $r \in \mathbb{F}_q$ and a secret sequence $(s_0, \ldots, s_{n-1}) \in \mathbb{F}_q^n$. The message number is used to index into the sequence, e.g., the ith message selects the secret s_i. The sender computes an authenticator a_i for the ith message as $a_i := m_i(r) + s_i$. Because it is important that each s_i is used only once (as it is a one-time pad), the index is called a *nonce*.

2.3 Security assurance through cryptanalysis

We have seen that the confidentiality of a message can be protected by the application of the one-time pad cryptosystem. We have also seen that forgeries of a message can be detected with high probability by computing an authenticator and transmitting it along. Is this the whole story? The problem with these cryptosystems is *scalability*; the number of secrets that the sender and receiver need to agree on is proportional to the number of messages. By assuming that the amount of resources that is available to an adversary is *bounded*, we are able to design cryptosystems that rely on a much smaller number of random symbols for their security. Let us call such an adversary a *bounded adversary*.

How do we design a cryptosystem that relies on a *short* secret key? The usual approach is to define a *mode* of operation, which is an algorithm that calls some underlying *primitive*. It may have other responsibilities. For example, it may pad the input string, partition the padded string into blocks, or include *diversifier* symbols. We define a primitive to be a function family that is indexed by a *large* set of keys that have a *short* description, e.g., a single key may be described as 256 bits, but the cardinality of the set of keys is 2^{256}. Examples of primitives are *block ciphers* and *deck functions*. A nonexample of a primitive is a *cryptographic permutation*. Indeed, for our purposes, primitives allow for clearly specified security definitions. Primitives may follow a *dedicated design*. Alternatively, they may be constructed on top of some *building block* like a *cryptographic permutation* or an existing primitive.

How do we reason about the security of such a cryptosystem? In the so-called *standard model*, the security of the mode is reduced to the *pseudorandomness* of the underlying primitive. Loosely speaking, a primitive is *PRF secure* if it is statistically "close" to a uniform random function, as measured by a particular distance function. Substituting "permutation" for "function" leads to the notion of a *PRP-secure* primitive. Then there are *SPRP-secure* primitives, which allow for querying the inverses. If the

primitive relies on some building block, then this building block is often modeled as a uniform random element to be able to prove a *bound* on the aforementioned distance. For example, if the building block is a cryptographic permutation, then this leads to the *random permutation model*, which is an example of an *ideal model*.

What is the point of a proof in some ideal model? The answer is that it makes it possible to reason about security against *generic attacks*. These are attacks that do not exploit the *structure* of the building block. For a concrete cryptosystem, the designers make a *security claim* and rely on *cryptanalysis* to falsify this claim. An example of a claim may be "there is no attack that is better than exhaustive key search" or "we believe the distance function to be bounded as follows." Hence, any such cryptosystem is only *conjectured* to be secure against a bounded adversary. This is true for all practically usable cryptosystems.

2.3.1 Random oracles

Suppose that an abelian group G is the n-fold direct product of an abelian group H with itself, i.e., suppose that $G = H^n$ for some positive integer $n \geq 1$. In this case, we can view the one-time pad cryptosystem of Section 2.3 as a mode of operation for a uniform random function. The one-time pad is of the form $k = (k_0, \ldots, k_{n-1})$, which we can think of as a uniform random element of $\mathrm{Maps}[\{0, \ldots, n-1\}, H]$ by mapping the index i to the corresponding secret k_i. It is an example of a fixed-length *random oracle* [6]. Because the indices can be obtained by incrementing a counter, this mode is a simple variant of the so-called *counter mode* of operation [24].

2.3.2 Pseudorandomness

Fix *finite* sets K, X, and Y. Note that this finiteness assumption is not a restriction in practice, as we can always choose sets of sufficiently large cardinality. We generalize the previous discussion to elements of $\mathrm{Maps}[X, Y]$. Suppose that we *replace* the uniform random element of $\mathrm{Maps}[X, Y]$, i.e., the random oracle, in counter mode with a nonuniform random element of $\mathrm{Maps}[X, Y]$. Loosely speaking, if no adversary is able to detect this nonuniformity with high probability, then the replacement does not change the security properties of the mode much. If this is the case, then we say that the nonuniform random element is *pseudorandom* against any adversary. Let us formalize this, using the *concrete security* approach [5].

PRF security. Suppose that $F\colon K \times X \to Y$ is a family of elements of $\mathrm{Maps}[X, Y]$ that is indexed by a finite set K of keys. Moreover, we assume that it is based on some building block, e.g., a cryptographic permutation. We view F_k as a (nonuniform) random element of $\mathrm{Maps}[X, Y]$ by selecting a uniform random element $k \xleftarrow{\$} K$. The pseudorandomness of F against an adversary $\mathscr{A}$ is measured by the advantage of $\mathscr{A}$ in distinguishing between F_k and a uniform random function, i.e.,

$$\mathrm{Adv}^{\mathrm{prf}}_F(\mathscr{A}) := \Delta_{\mathscr{A}}(F_k, f)\,,$$

where the probabilities are computed with respect to $k \xleftarrow{\$} K, f \xleftarrow{\$} \mathrm{Maps}[X, Y]$, and the random choices that the adversary $\mathscr{A}$ makes (if any).

Recall that we are only considering bounded adversaries. We specify two important resource measures that are associated with an adversary.

First, there is its *online* or *data* complexity, which is equal to the amount of data that is exchanged between the adversary and the oracle, either F_k or f. For example, if $X = Y = \{0,1\}^{\leq 2^{128}}$, then the data complexity can be measured as the number of input bits to and output bits of the oracle. The quantity $|X| + |Y|$ is a natural upper bound on the online complexity, as the unknown function is completely specified by the set of all input-output pairs. In practice, an upper bound on the online complexity is determined by the use case. For example, a system that implements the oracle may update the secret after every n calls to the oracle, for some positive integer $n \geq 1$ that is relatively small.

Second, there is its *offline* or *computational* complexity, which is determined by all the computations that the adversary can perform that do not require knowledge of the secret (this definition does not exclude guessing the secret). It is often measured in a number of computationally equivalent calls to the building block that it used to implement F. The cardinality of K is a natural upper bound on the offline complexity, as trying out all the possible values for the secret is sufficient to break the scheme, assuming that the online complexity is sufficiently large. We will make this more precise shortly. In practice, a more useful upper bound that we may assume is determined by the financial resources that an adversary is able and willing to invest.

Consider the set $\mathbb{A}(M, N)$ of all adversaries that have online complexity at most M and have offline complexity at most N. The *PRF advantage function* of F is defined as

$$\mathrm{Adv}^{\mathrm{prf}}_F(M, N) := \sup\{\mathscr{A} \in \mathbb{A}(M, N) : \mathrm{Adv}^{\mathrm{prf}}_F(\mathscr{A})\}\,.$$

Loosely speaking, we call F a "PRF-secure function" if $\mathrm{Adv}^{\mathrm{prf}}_F(M, N)$ is "small" for "practical" values of M and N. Clearly, the meaning of these words depends on the use case. PRF-secure functions are indistinguishable from uniform random functions by suitably bounded adversaries.

Exhaustive key search. Clearly, lower bounds of the PRF advantage function are determined by concrete adversaries (i.e., attacks). Here is an example of a generic attack on F that highlights the importance of an upper bound on the offline complexity. The attack $\mathscr{A}$ is called *exhaustive key search*. Suppose that $\mathscr{A}$ is given an n-time oracle. It queries that oracle for inputs $x_0, \ldots, x_{n-1}$, obtains the corresponding outputs $y_0, \ldots, y_{n-1}$, and determines, offline, the subset of keys $k \in K$ for which $y_i = F_k(x_i)$ holds, for $i = 0, \ldots, n-1$. If this subset of keys is nonempty, then it outputs 0, i.e., it believes that the oracle is an oracle for F, as opposed to a uniform random function. For this particular $\mathscr{A}$, we find that

$$\mathrm{Adv}^{\mathrm{prf}}_F(\mathscr{A}) \geq 1 - |K|/|Y|^n\,.$$

To reiterate, without an upper bound on the offline complexity, the adversary can trivially distinguish between F and a fixed-length random oracle.

Random permutation model. On the other hand, upper bounds of the PRF advantage function can typically be derived only when the underlying primitives are replaced by their uniform random counterparts. Here is an example. Suppose that F is built around an n-tuple $P = (P_1, \ldots, P_n)$ of permutations on some finite set Z for some positive integer $n \geq 1$. Denote the n-tuple of their inverses by $P^{-1} = (P_1^{-1}, \ldots, P_n^{-1})$. To define the pseudorandomness of F, the oracles in the definition are changed from F_k and f to (F_k, P, P^{-1}) and (f, P, P^{-1}), respectively. This model is called the *random permutation model*.

Security claims. If the analysis is performed with respect to the actual building blocks (as opposed to the uniform random counterpart), upper bounds of the PRF advantage function are conjectural and based on cryptanalysis. Such upper bounds typically appear in a so-called *security claim*. Security claims may also refer to the infeasibility of specific adversaries. For example, a claim may state that there exists no attack that is "better" than exhaustive key search.

Security strength. Comparing the pseudorandomness of two function families is often bothersome to do by comparing the upper bounds of the respective PRF advantage functions. Instead, we would like to have a single number, the *security strength* [38]. The security strength of F is n bits if

$$\log_2\left(\frac{M+N}{\mathrm{Adv}_F^{\mathrm{prf}}(M,N)}\right) < n\,.$$

Deck functions. In this paragraph, we specialize to the case of variable-length input and variable-length output. To that end, let S be a nonempty finite set of symbols. A *doubly-extendable cryptographic keyed (deck) function* [16] is a function family $F\colon K \times \mathbb{Z}_{\geq 0} \times \mathbb{Z}_{\geq 0} \times (S^*)^+ \to S^*$ that satisfies the following properties. It outputs an infinite sequence of symbols. From this infinite sequence, a finite subsequence is obtained by specifying an offset and a length. It must allow for efficient *incremental computation*; given any two sequences $s, t \in (S^*)^+$, if s has already been processed, then the processing time of $s \parallel t$ should depend only t. Moreover, the time it takes to generate additional output symbols should depend only on the number of additional symbols that is requested. Both are accomplished by remembering the internal state. The security of a deck function is measured by its PRF advantage function against a random oracle with the same interface, and the security claim is expressed as an upper bound on this function.

Block ciphers. In contrast with the counter mode of operation, some modes of operation require F to be invertible. In this paragraph, we therefore specialize to the case that X equals Y and F is invertible. This is called a *block cipher*. Invertibility implies additional structure. Hence, we require a slightly different notion of pseudorandomness to formalize what it means for F to be secure. The pseudorandomness

of F against an adversary $\mathscr{A}$ is the advantage of $\mathscr{A}$ in distinguishing between F_k and a uniform random permutation, i.e.,

$$\mathrm{Adv}^{\mathrm{prp}}_F(\mathscr{A}) := \Delta_{\mathscr{A}}(F_k, p)\,,$$

where the probabilities are computed with respect to $k \xleftarrow{\$} K$, $p \xleftarrow{\$} \mathrm{Perms}[X]$, and the random choices that $\mathscr{A}$ makes (if any).

Consider the set $\mathbb{A}(M, N)$ of all adversaries that have online complexity at most M and have offline complexity at most N. The *PRP advantage function* of F is defined as

$$\mathrm{Adv}^{\mathrm{prp}}_F(M, N) := \sup\{\mathscr{A} \in \mathbb{A}(M, N) : \mathrm{Adv}^{\mathrm{prp}}_F(\mathscr{A})\}\,.$$

Loosely speaking, we call F a "PRP-secure block cipher" if $\mathrm{Adv}^{\mathrm{prp}}_F(M, N)$ is "small" for "practical" values of M and N. PRP-secure block ciphers are indistinguishable from uniform random permutations by suitably bounded adversaries.

While PRP security models the block cipher as an efficiently computable permutation that is indistinguishable from a random permutation, it only considers access to the forward direction of the cipher. The security definition implicitly relies on the invertibility of F_k, but the inverse function itself does not appear in it. To strengthen the model, one can also give the adversary access to the inverse oracle. The resulting notion, where (F_k, F_k^{-1}) must be indistinguishable from (p, p^{-1}), is known as SPRP security.

Formally, the *SPRP advantage* of an adversary $\mathscr{A}$ against F is its advantage in distinguishing between oracle access to (F_k, F_k^{-1}) and oracle access to (p, p^{-1}), where $k \xleftarrow{\$} K$ and $p \xleftarrow{\$} \mathrm{Perms}[X]$. We write

$$\mathrm{Adv}^{\mathrm{sprp}}_F(\mathscr{A}) := \Delta_{\mathscr{A}}\big((F_k, F_k^{-1}), (p, p^{-1})\big)\,.$$

Loosely speaking, we call F an "SPRP-secure block cipher" if $\mathrm{Adv}^{\mathrm{sprp}}_F(M, N)$ is "small" for "practical" values of M and N.

PRP-PRF switching lemma. Sometimes, a mode that requires a PRF-secure function is instantiated with a block cipher. How is the PRF security of a block cipher related to its PRP security? By definition, a permutation has no collisions, whereas a uniform random function is expected to have collisions. Indeed, the probability that at least one collision occurs after n queries to a uniform random function is approximately equal to $n^2/2|X|$. This follows from the solution to the famous *birthday problem* [28]. The *PRP-PRF switching lemma* [11] uses this fact to bound the PRF advantage of a block cipher F in terms of its PRP advantage. In particular, it states that

$$\mathrm{Adv}^{\mathrm{prf}}_F(\mathscr{A}) \le \mathrm{Adv}^{\mathrm{prp}}_F(\mathscr{A}) + \frac{n^2}{2|X|}\,,$$

for all adversaries $\mathcal{A}$ with access to an n-time oracle. This observation forms the foundation of the Sweet32 attacks on the TLS protocol [9]. However, if $|X|$ is sufficiently large, then the PRP-PRF switch poses no problems.

2.3.3 Cryptanalysis

Loosely speaking, cryptanalysis is about trying to falsify some security claim. Arguably the most important branches of cryptanalysis of secret-key primitives are differential [10], linear [15, 37], algebraic [3], integral [17], and higher-order differential cryptanalysis [35]. If a primitive is defined over a field of characteristic 2, then the latter two are essentially the same. A detailed explanation of these techniques is presented in later sections. What follows now is a very brief summary of the core ideas that underlie each technique.

Differential cryptanalysis. *Differential cryptanalysis* is the branch of cryptanalysis that deals with the propagation of *differences* through a function. Concretely, let $f\colon G \to H$ be a function between finite abelian groups G and H. Moreover, let $x \xleftarrow{\$} G$ be a uniform random element, let $a \in G$ be an input difference, and let $b \in H$ be an output difference. The cryptanalyst is interested in the probability that the predicate $f(x) - f(x - a) = b$ is true. This is called the *differential probability* (DP) of the *differential* (a, b) over f.

For a uniform random f, the DP of any differential over f is a random element of $[0, 1]$ with expected value $\mu = 1/|H|$ and variance $\sigma^2 = 1/(|G||H|)$. Indeed, the number of elements $x \in G$ that satisfy the predicate is a random element of $[0, |G|]$ that has, approximately, a Poisson distribution with expected value and variance equal to $\lambda = |G|/|H|$ [20] if $|G|$ is sufficiently large and $|H|$ is sufficiently small. Hence, any significant deviation from λ could lead to a distinguishing attack. The number of samples that is required to detect this deviation is approximately equal to the inverse of the DP of the differential.

Instead of considering individual input and output differences, the cryptanalyst may consider the propagation of sets of differences. Given subsets $A \subseteq G$ and $B \subseteq H$, we now consider the conditional probability that the predicate $f(x) - f(x') \in B$ is true given that $x - x' \in A$. This generalization is called *truncated* differential cryptanalysis.

Linear cryptanalysis. *Linear cryptanalysis* is the branch of cryptanalysis that deals with linear approximations of functions. Concretely, let $f, g\colon \mathbb{F}_2^n \to \mathbb{F}_2$ be two Boolean functions. The *distance*, say δ, between f and g is defined as the probability that $f(x) \neq g(x)$ for a uniform random element $x \xleftarrow{\$} \mathbb{F}_2^n$. The distance between f and a subset P of Boolean functions is defined as $\delta(f, P) := \min\{\delta(f, g) : g \in P\}$. In particular, the cryptanalyst is interested in the case that $P = \{x \mapsto \sum_{i=0}^{n-1} a_i x_i : a \in \mathbb{F}_2^n\}$ is the set of linear functionals.

It turns out that a convenient quantity to work with is the *correlation*, say c, between f and g, which is equal to $c = 1 - 2\delta$. Let $g \in P$ and let f be a uniform random Boolean function, then c is a random element of $[0, 1]$ that is approximated by a normal distribution with expected value $\mu = 0$ and variance

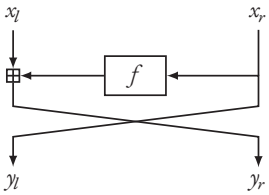


Figure 2.1: The Feistel structure.

$\sigma^2 = 2^{-n}$ [20]. Any significant deviation from 0 could potentially lead to a distinguishing attack. The number of samples that is required to detect this deviation is approximately equal to the inverse of c^2. We should note that zero-correlation attacks exist as well.

Shortly, we will see how to generalize this to functions between any two finite abelian groups and how linear cryptanalysis effectively is Fourier analysis. In the Fourier domain, the Fourier transform turns translations (e.g., by a subkey or a round constant) into modulations, which are much easier to deal with and understand.

Integral cryptanalysis. Let X and Y be sets. *Integral cryptanalysis* is the branch of cryptanalysis that deals with predicting the sum of outputs of a function $f : X \to Y$ over a multiset of inputs $S \subseteq X$. In other words, the cryptanalyst is interested in the value of $\sum_{s \in S} f(s)$. Predictions are typically based on the propagation of some integral property, e.g., knowing that if some input variable takes on all the values, then some output variable will take on all the values. If X and Y are finite fields of characteristic 2, then the "best" multisets S are often affine subspaces of X. In this case, an important subclass of integral attacks is formed by the so-called *higher-order* differential attacks. These attacks rely on estimates of the degree of a polynomial representation of f.

2.3.4 Constructing block ciphers

A block cipher is designed with two properties in mind. First, for every key $k \in K$, both F_k and F_k^{-1} should be efficiently implementable. Second, it should withstand known cryptanalysis techniques.

Internally, the key is usually, but not always, processed by a *key schedule*, which is responsible for the derivation of *subkeys*. The data is processed by a *data path*, which makes use of the subkeys. Hence, subkeys flow from the key schedule into the data path, but data does not flow from the data path into the key schedule.

The data path is usually obtained as the composition of a number of relatively simple *round functions*, which are often the same, perhaps up to the addition of a round constant or subkey. There is a lot of freedom in designing a round function. Importantly, any design should try to balance the computational complexity, which depends on the computing environment, and the security that it offers. We summarize important *structures* that frequently make an appearance in the design of round functions.

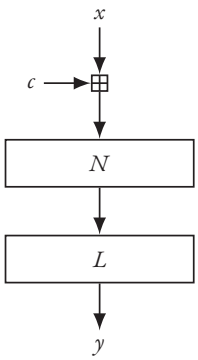


Figure 2.2: The SPN structure.

Suppose that the set of inputs forms a direct product of an abelian group G with itself. In that case, we can design a function on that set that has the *Feistel* structure [36]. Input pairs $(x_l, x_r) \in G \times G$ are transformed by the function $(x_l, x_r) \mapsto (x_r, x_l + f(x_r))$ that is parameterized by some function $f\colon G \to G$ that may depend on a subkey. This is illustrated in Figure 2.1. An important property of the Feistel structure is that it is invertible, regardless of the invertibility of f. Indeed, the inverse is equal to $(y_l, y_r) \mapsto (y_r - f(y_l), y_l)$. Moreover, observe that the function $(x_l, x_r) \mapsto (x_r, x_l)$ is, in fact, an involution. If the exponent of G is 2, then the function $(x_l, x_r) \mapsto (x_l + f(x_r), x_r)$ is also an involution. In that case, encryption and decryption are essentially the same operation, the difference lying in the order in which the subkeys are supplied. In other words, encryption and decryption can use the same circuit in hardware. Another important property is that an SPRP-secure function can be obtained from a PRF-secure function f (in that case, f should depend on a subkey) by repeating the structure 4 times [36].

Some functions have the *SPN* structure, which stands for substitution permutation network. It usually consists of the following layers, not necessarily in that order.

- (constant addition layer): This layer adds round constants and/or subkeys to the internal *state*. Round constants introduce asymmetry between the different rounds. Subkeys add uncertainty about the function's internal state.

- (nonlinear layer): This layer prevents straightforward linear modeling of the function. Usually, it is composed of a number of *S-boxes*. An S-box has certain algebraic properties, e.g., its degree, and statistical properties, e.g., its differential probabilities.

- (linear layer): This layer consists of a transposition step that *shuffles* the symbols of the internal state, and, optionally, a mixing step, that adds internal state symbols together.

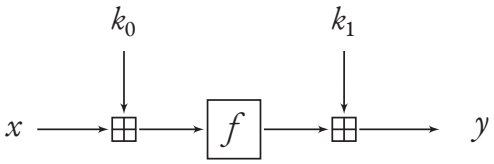


Figure 2.3: The Even-Mansour construction.

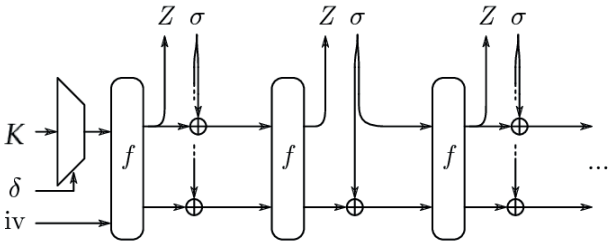


Figure 2.4: The full-state keyed duplex construction.

It is the *interaction* between the various layers that gives the function good cryptographic properties. Informally, *diffusion* ensures that each output symbol is a function of every input symbol, and *confusion* means that this function is highly complex. The SPN structure is illustrated in Figure 2.2.

Some functions are based on the operations of modular addition, rotation, and bitwise addition (XOR). They are said to have the *ARX* structure. One advantage of ARX-based functions is that they tend to have efficient software implementations, each operation usually taking only a single CPU cycle to compute. In particular, this is true for the addition operation, which exhibits good differential, linear, and algebraic complexity. There are also several disadvantages. A hardware implementation of the addition operation requires a circuit for the propagation of carry bits. Such a circuit has either a small area or a short critical path, but not both. Moreover, the security of ARX structures is more difficult to analyze compared to structures that are based on S-boxes.

Suppose that $k = (k_0, k_1)$. The Even-Mansour construction [25] can be used to obtain a block cipher from a cryptographic permutation. It is illustrated in Figure 2.3.

Note that some of these structures may appear as substructures of the others. For example, the f-function that appears in the Feistel structure could have the SPN structure. In turn, the S-boxes that appear in an SPN structure could have an ARX structure.

2.3.5 Constructing deck functions

We present a number of constructions of deck functions that are useful to know about for better understanding of the contents of this thesis.

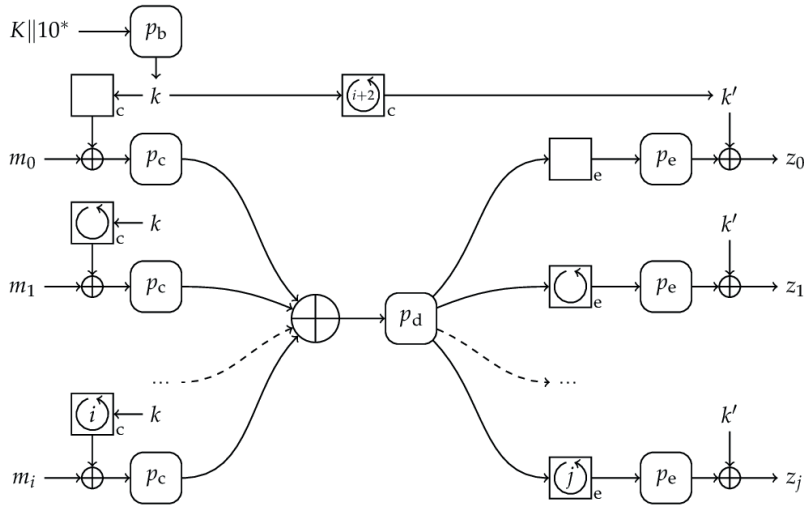


Figure 2.5: The Farfalle construction.

Full-state keyed duplex construction. Figure 2.4 illustrates the *full-state keyed duplex* (FKD) construction [18]. FKD is a *serial* construction that is based on a cryptographic permutation f that operates on a b-bit state. A user can make two types of calls: *initialization* calls and *duplex* calls. In an initialization call, a key index δ is used to select a k-bit key from a key array K and it is loaded together with a $(b - k)$-bit string iv into the state. Next, an r-bit string Z is returned and a b-bit user-supplied string σ is injected into the state. In a duplex call, the state is transformed by f, an r-bit string Z is returned to the user and a b-bit user-supplied string σ is injected into the state. Loosely speaking, the PRF security of FKD is given in terms of a somewhat complicated looking upper bound on its $\mathcal{A}$-distance to the Ideal eXtendable Input Function (IXIF). The IXIF has the same interface as the FKD, but is implemented with a random oracle "under the hood."

Farfalle construction. Figure 2.5 illustrates the *Farfalle* construction [8]. Farfalle is a *parallel* construction that is based on a cryptographic permutation and it consists of a *mask derivation* layer, a *compression* layer, and an *expansion* layer. Masks are derived from the input key and added to input blocks. Each input block is transformed by a permutation in the compression layer, independently from the other blocks. The resulting output blocks are added to each other into an accumulator. After being transformed by another permutation, the accumulator is used to generate pseudorandom output blocks in the expansion layer by adding different masks to it.

2.3.6 Constructing cryptographic permutations

A cryptographic permutation P is an element of the set $\mathrm{Perms}[X]$ that is efficiently implementable. In the construction of some primitives, there is the requirement that its inverse P^{-1} is efficiently implementable as well. Typically, the design of P closely follows that of the data path of a block cipher; it uses the same structures that we have discussed in the corresponding subsection. Importantly, there is *no* dedicated secret key input.

What it means for a cryptographic permutation to be secure cannot be formalized. Indeed, it is only at the level of the primitive that uses P, where there *is* a secret key involved, that we are able to formally define security, such as PRF security and PRP security. Loosely speaking, however, P should not have any structural properties that allow for falsifying the security claim of the primitive that uses it.

2.4 Implementation security

So far, security has been defined in terms of an adversary that queries an oracle a number of times and needs to determine which object it has access to. Crucially, the oracle is treated as a *black box*. Although its specification is typically known, its secret key is not. In practice, an adversary may also be able to exploit the interaction between the algorithm and its implementation.

Such implementation attacks are best classified along two independent axes. The first axis is the method of access: *non-invasive*, *semi-invasive*, or *invasive*. Non-invasive methods operate entirely through standard interfaces or external observation. Semi-invasive methods expose the chip's internals without damaging its internal structures. Invasive methods involve physically tampering with the chip.

The second axis is the type of technique used, such as *side-channel analysis* or *fault injection*. Side-channel attacks rely on passively observing physical leakages, such as power consumption or electromagnetic emissions, which may depend on secret data. Fault attacks, by contrast, involve actively disturbing the device's normal operation, e.g., by injecting voltage glitches or laser pulses, so that internal secrets can be inferred from incorrect outputs. Both types of attacks can be mounted using any of the three access methods, depending on the attacker's capabilities.

Countermeasures against side-channel attacks often involve *masking* techniques that randomize intermediate computations. Fault attacks are typically mitigated by introducing redundancy or error detection mechanisms. Physical protections, such as tamper sensors or shielding, can further increase resistance against invasive and semi-invasive attacks.

2.5 Algebraic cryptanalyis

In this section, we introduce the notion of a Gröbner basis for an ideal and its application to computing the solutions of a system of polynomial equations. As a motivating example, the cryptosystem itself can be modeled as a system of equations in the secret symbols, which may allow for recovery of the secret. Throughout, we write $R_n := k[x_0, \dots, x_{n-1}]$ for the set of polynomials in the variables $x_0, \dots, x_{n-1}$ and

with coefficients in the field k. The interpretation of the variables depends on the model being used; typically, the ith variable represents the ith input symbol.

Loosely speaking, a Gröbner basis is a set of polynomials that generates the ideal and for which multivariate division always leads to a unique remainder, regardless of the order in which reductions are applied. The standard algorithm for computing a Gröbner basis is Buchberger's algorithm. Most of the state-of-the-art algorithms are essentially (sophisticated) optimizations of this algorithm. Buchberger's algorithm can be seen as a generalization of both the Gaussian elimination algorithm from linear algebra and Euclid's algorithm for computing the polynomial greatest common divisor. The contents of this section are based on the exposition of [34].

2.5.1 Monomial orders

It is useful to have a unique representation of the polynomials in R_n. To that end, we equip R_n with a *monomial order* by identifying x^α with the exponent vector $\alpha \in \mathbb{Z}_{\geq 0}^n$, and imposing an order on $\mathbb{Z}_{\geq 0}^n$.

Definition 1. *A monomial order on R_n is a relation $>$ on the set $\mathbb{Z}_{\geq 0}^n$ that satisfies:*

- *$>$ is a well-order.*
- *If $\alpha > \beta$ and $\gamma \in \mathbb{Z}_{\geq 0}^n$, then $\alpha + \gamma > \beta + \gamma$.*

We write $x^\alpha > x^\beta$ if $\alpha > \beta$ and use the convention that $x^\alpha > 0$ for all $\alpha \in \mathbb{Z}_{\geq 0}^n$. An example of a monomial order is the *lexicographic* order. It is important for reasoning about the shape of a Gröbner basis.

Definition 2. *We have $\alpha >_{lex} \beta$ if the leftmost nonzero component of $\alpha - \beta$ is positive.*

A second example of a monomial order is the *graded reverse lexicographic* (grevlex) order. This order is important for the computation of Gröbner bases, as it leads to the shortest computation times.

Definition 3. *We have $\alpha >_{grevlex} \beta$ if either $\sum_{i=0}^{n-1} \alpha_i > \sum_{i=0}^{n-1} \beta_i$ or $\sum_{i=0}^{n-1} \alpha = \sum_{i=0}^{n-1} \beta$ and the rightmost nonzero entry of $\alpha - \beta$ is negative.*

Now that R_n is ordered, we can speak of leading monomial, leading term, etc.

Definition 4. *Let $f = \sum_\alpha c_\alpha x^\alpha$ be a non-zero polynomial in R_n and let $>$ be a monomial order.*

- *We call* $\operatorname{multideg}(f) = \max_>\{\alpha \in \mathbb{Z}_{\geq 0}^n : c_\alpha \neq 0\}$ *the multidegree of f.*
- *The leading coefficient of f is* $\operatorname{lc}_>(f) = \operatorname{lc}(f) = c_{\operatorname{multideg}(f)}$.
- *The leading monomial of f is* $\operatorname{lm}_>(f) = \operatorname{lm}(f) = x^{\operatorname{multideg}(f)}$.
- *The leading term of f is* $\operatorname{lt}_>(f) = \operatorname{lt}(f) = \operatorname{lc}(f) \operatorname{lm}(t)$.

Example 1. *Consider $f = x_0 + x_1x_2 + x_2 \in \mathbb{F}_2[x_0, x_1, x_2]$ with the lexicographic order. Its multidegree is equal to $(1, 0, 0)$, its leading coefficient is equal to 1, its leading monomial is equal to x_0, and its leading term is equal to x_0.*

2.5.2 Multivariate polynomial division

Originally, Gröbner bases were introduced to solve the ideal membership problem, which asks to decide whether a polynomial f is in the ideal $\langle F \rangle$ that is generated by polynomials $F := \{f_0, \dots, f_{m-1}\}$ or not. If we suppose that f and the f_i are elements of the univariate polynomial ring $k[x]$, then the problem is easily solved. Indeed, the ideal $\langle F \rangle$ is generated by $g = \gcd(f_0, \dots, f_{m-1})$. It follows that we only need to check whether f is a multiple of g. This is a simple application of the univariate division algorithm. It is this concept of division that we want to generalize to R_n.

As a first step, we recall how the univariate division algorithm works. Suppose that we wish to divide $f = a_0 + a_1 x + \cdots + a_n x^n$ by $g = b_0 + b_1 x + \cdots + b_m x^m$, assuming that both a_n and b_m are non-zero. Notice that the terms of f and g are ordered by degree. We divide the leading term (with respect to this ordering) of f by the leading term of g. That is, we calculate $q_i := (a_n x^n)/(b_m x^m)$ and recursively apply this principle to $f_i := f - q_i g$ and g. When the leading term of g no longer divides f_i we end the recursion and store $\sum_{i=0}^{m-1} q_i$ in q and f_i in r. The polynomial q is called the quotient and the polynomial r is called the remainder.

Let us now try to mimic this procedure in R_n. Some definitions are in order. Let f and g be polynomials in R_n. We say that f is *top-reducible* by g if $\mathrm{lm}(g)$ divides $\mathrm{lm}(f)$. The corresponding top-reduction is given by $f - (\mathrm{lt}(f)/\mathrm{lt}(g))g$. The effect of a top-reduction is that the leading term of f is canceled. We say that f is top-reducible by F if there exists an $i \in [0, m-1]$ such that f is top-reducible by f_i. When no f_i top-reduces f we say that f is *top-irreducible* by F. When f is top-irreducible by F, we may proceed and try to reduce $f - \mathrm{lt}(f)$. If there is a term of f that is divisible by a leading monomial of some f_i, we say that f is *reducible* by F. When f is no longer *reducible*, we call it irreducible and we end up with a remainder r. In general, this remainder is not unique, as it depends on the order in which the reductions were applied.

Proposition 1. *Let $F = (f_0, \dots, f_{m-1})$ be an m-tuple of polynomials in R_n and fix a monomial order $>$. Every polynomial $f \in R_n$ can be written as*

$$f = \sum_{i=0}^{m-1} q_i f_i + r$$

where $q_i, r \in R_n$ and either $q_i f_i = 0$ or $\mathrm{lt}(f) > \mathrm{lt}(q_i f_i)$ for $i \in [0, m-1]$. Moreover, we either have r equal to 0 or r is a linear combination of monomials that are not divisible by any $\mathrm{lt}(f_i)$ for $i \in [0, m-1]$. We will call r a remainder of f on division by F.

To obtain the quotients q_i and the remainder r, we apply Algorithm 1.

Algorithm 1 Multivariate division

Input: A sequence $F = (f_0, \dots, f_{m-1})$ of polynomials in R_n, a polynomial $f \in R_n$, and a monomial order $>$.
Output: Polynomials $r, q_0, \dots, q_{m-1} \in R_n$ such that $f = \sum_{i=0}^{m-1} q_i f_i + r$.
$r \leftarrow 0$
$h \leftarrow f$
for $i \leftarrow 0, m-1$ **do**
    $q_i \leftarrow 0$
end for
while $h \neq 0$ **do**
    $j \leftarrow 0$
    $divided \leftarrow False$
    while $j \leq m \wedge \neg divided$ **do**
        if $\mathrm{lt}(f_j)$ divides $\mathrm{lt}(h)$ **then**
            $h \leftarrow h - \frac{\mathrm{lt}(h)}{\mathrm{lt}(f_j)} \mathrm{lt}(f_j)$
            $q_j \leftarrow q_j + \frac{\mathrm{lt}(h)}{\mathrm{lt}(f_j)}$
            $divided \leftarrow True$
        else
            $j \leftarrow j + 1$
        end if
    end while
    if $\neg divided$ **then**
        $h \leftarrow h - \mathrm{lt}(h)$
        $r \leftarrow r + \mathrm{lt}(h)$
    end if
end while
return $r, q_0, \dots, q_{m-1}$

2.5.3 Gröbner bases

Throughout, let I be any ideal of R_n and fix some monomial order $>$ on R_n. We are now ready to define what a Gröbner basis is.

Definition 5. *A finite subset $G = \{g_0, \dots, g_{l-1}\} \subseteq I$ is said to be a* Gröbner basis *for I with respect to $>$ if every polynomial in I is top-reducible by G.*

Proposition 2. *There exists a Gröbner basis G for I with respect to $>$ and it generates I.*

Multivariate division leads to a unique remainder if the order of the reductions is fixed. If we are reducing modulo a Gröbner basis, then the order does not matter. This is the content of the following proposition.

Proposition 3. *Let $G = \{g_0, \dots, g_{l-1}\}$ be a Gröbner basis for I with respect to $>$. There exists a unique polynomial $r \in R_n$ that satisfies the following.*

- *The polynomial r is irreducible by G, and*
- *there is a $g \in I$ such that $f = g + r$.*

The polynomial r is often called the normal form *of f with respect to G and we denote it by f* rem *G.*

Let us apply this to the ideal membership problem. We compute a Gröbner basis G for the ideal $\langle F \rangle$ with respect to $>$ and compute f rem G. The normal form will tell us whether f is in the ideal or not.

Corollary 1. *The polynomial f is in I if and only if f* rem $G = 0$.

Proof. Suppose that $f \in I$. By proposition 3 f rem $G = f - g$ for some $g \in I$. It follows that f rem $G \in I$. However, no term of f rem G is divisible by any $\mathrm{lt}(g)$ with $g \in G$. The fact that G is a Gröbner basis then implies that f rem $G = 0$. To prove the converse, we assume that f rem $G = 0$. Again, by proposition 3, we deduce that there exists a $g \in I$ such that $f = g + (f \text{ rem } G)$. Our assumption then implies that $f = g \in I$. □

In general, an ideal can have many different Gröbner bases with respect to $>$. However, by imposing some restrictions, it is possible to guarantee uniqueness.

Definition 6. *A Gröbner basis G for I with respect to $>$ is said to be* reduced *if*

- $\mathrm{lc}(g) = 1$ *for all $g \in G$, and*
- g rem $G \setminus \{g\} = g$ *for all $g \in G$.*

Proposition 4. *The ideal I has a* unique *reduced Gröbner basis with respect to $>$.*

2.5.4 Systems of polynomial equations

Suppose that we are given a system of polynomial equations of the form $f_0 = \cdots = f_{m-1} = 0$. The equations define an ideal $I := \langle \{f_0, \dots, f_{m-1}\} \rangle$. The set $Z(I) := \{a \in k^n : f(a) = 0 \text{ for each } f \in I\}$ contains the solutions to the equations. How do we obtain $Z(I)$ from I? It turns out that this is easy to do when we have the reduced Gröbner basis for I with respect to the lexicographic order.

To explain this, we need a few more definitions. An ideal I is called *zero-dimensional* if $Z(I)$ contains only finitely many points. Suppose that I is a zero-dimensional ideal and let $i \in [0, n-1]$. We say that I is in normal x_i-position if any two points $a, b \in Z(I)$ satisfy $a_i \neq b_i$.

Proposition 5. *Let k be a field of characteristic 0 or a finite field. Moreover, let F be a sequence of polynomials in R_n. Put $I = \langle F \rangle$. Assume that I is zero-dimensional, radical, and in normal x_{n-1}-position. Write m for $|Z(I)|$. Let G be the reduced Gröbner basis for I with respect to the lexicographic order with $x_0 > x_1 > \cdots > x_{n-1}$. Under these assumptions, G is of the form*

$$\{x_0 - g_0(x_{n-1}), x_1 - g_1(x_{n-1}), \dots, x_{n-2} - g_{n-2}(x_{n-1}), x_{n-1}^m - g_{n-1}(x_{n-1})\}$$

where each g_i is a univariate polynomial in $k[x_{n-1}]$ of degree at most $m-1$. In particular, this shows that

$$Z(I) = \{(g_0(a_i), \dots, g_{n-2}(a_i), a_i) : i \in [0, m-1]\},$$

where $a_0, \dots, a_{m-1}$ are the roots of $x_{n-1}^m - g_{n-1}(x_{n-1})$.

The assumptions of Proposition 5 are not too restrictive. Indeed, if k is a finite field, say $\mathbb{F}_q$, then the ideal $I_q := I + \langle x_0^q - x_0, \dots, x_{n-1}^q - x_{n-1} \rangle$ is zero-dimensional and radical. Ensuring that I_q is in normal x_{n-1}-position may require moving to an extension field of $\mathbb{F}_q$ and a linear change of coordinates. However, this is not a problem, as the solutions are guaranteed to be in $\mathbb{F}_q$ due to the relations $x_i^q - x_i$ for $i = 0, \dots, n-1$.

2.5.5 Algorithms

Given the usefulness of Gröbner bases, we address how to find such a basis. Buchberger gave the first algorithm for computing a Gröbner basis. His algorithm relies heavily on the concept of the S-polynomial of two polynomials.

Definition 7. *Let $f, g \in R_n$ be non-zero polynomials. The* S-polynomial *of f and g is defined as*

$$S(f,g) = \frac{x^\gamma}{\mathrm{lt}(f)} \cdot f - \frac{x^\gamma}{\mathrm{lt}(g)} \cdot g$$

where $x^\gamma = \mathrm{lcm}(\mathrm{lm}(f), \mathrm{lm}(g))$.

The S-polynomial of two polynomials is constructed in such a way that their leading terms are canceled. Buchberger's criterion, which is stated in Proposition 6, gives us an algorithmic test for checking whether a set of polynomials is a Gröbner basis or not.

Proposition 6. *A finite subset $F = \{f_0, \dots, f_{l-1}\}$ is a Gröbner basis for I with respect to $>$ if and only if for all pairs (i, j) of distinct indices the S-polynomial $S(f_i, f_j)$ reduces to 0 modulo G.*

Using Proposition 6, one readily obtains Buchberger's algorithm, which is stated in Algorithm 2.

Algorithm 2 Buchberger's algorithm

Input: A sequence of polynomials $F = \{f_0, \dots, f_{m-1}\}$ and a monomial order $>$.
Output: A Gröbner basis G for $\langle F \rangle$ with respect to $>$.

```
G ← F
P ← Sort({(p, q) : p, q ∈ G, p ≠ q})
while P ≠ ∅ do
    (p, q) ← the first element of P
    P ← P \ {(p, q)}
    r ← S(p, q) rem G
    if r ≠ 0 then
        P ← Sort(P ∪ {(g, r) : g ∈ G})
        G ← G ∪ {r}
    end if
end while
return G
```

Proposition 7. *Buchberger's algorithm 2 terminates in a finite number of steps and outputs a Gröbner basis for $\langle \{f_0, \dots, f_{m-1}\} \rangle$.*

Proof. Write $I = \langle F \rangle$. We first prove correctness. We want to use proposition 6 to prove this. To this end, we need to show two things. First, that G is a subset of I during the entire execution of the algorithm. Second, at the end $S(g, h)$ rem $G = 0$ for all $g, h \in G$ with $g \neq h$. At the start of the algorithm $G = F \subseteq I$. During each iteration of the while loop G is augmented with r. Since $p, q \in I$ it follows that $S(p, q) \in I$ and since $G \subseteq I$ we deduce that $r \in I$. Therefore, $G \cup \{r\} \subseteq I$. Whenever we process a new pair, if the remainder of the S-polynomial by G didn't already equal zero, then we add the remainder. This ensures that subsequent computation of the remainder will yield zero. Hence if the algorithm terminates we have that $S(g, h)$ rem $G = 0$ for all $g, h \in G$ with $g \neq h$. Next, we prove that the algorithm indeed terminates. Every time a nonzero remainder is added to G the ideal $\langle \mathrm{lt}(G) \rangle$ strictly increases. This leads to an ascending chain of ideals in R_n. By Noetherianity of R_n this chain eventually stabilizes. This means that eventually the if-branch is never executed. Therefore the set P eventually becomes empty and the algorithm terminates. □

In practice. Algorithm 2 is not very efficient. Every time an S-polynomial is reduced to zero, we do not discover any new element of the Gröbner basis. As the reduction step is the most time consuming step, it is natural to consider strategies that avoid these useless reductions. Since it is impossible to avoid reducing S-polynomials altogether, much effort has also gone in speeding up the reduction step. Finally, the order in which critical pairs are processed has an effect on performance as well. This becomes important when the input polynomials are comprised of terms having different degrees.

From a theoretical point of view, state-of-the-art Gröbner basis algorithms are simply improvements to Buchberger's algorithm that include enhanced selection criteria, a faster reduction step that makes use of fast linear algebra, and an attempt to predict reductions to zero. A fast algorithm is Faugère's F5 algorithm [4, 26].

Experiments highlighted that computing a Gröbner basis with respect to the lexicographic order is a slow process. Computing a Gröbner basis with respect to the grevlex order can be done in a faster manner. However, we need the lexicographic order to be able to apply Proposition 5. Fortunately, the FGLM algorithm [27] makes it possible to transform a Gröbner basis with respect to the grevlex order to another with respect to the lexicographic order. To summarize, an adversary adopts the following strategy:

1. Using the F5 algorithm, compute a Gröbner basis with respect to the grevlex order.
2. Using the FGLM algorithm, transform the previous basis into a Gröbner basis with respect to the lexicographic order.
3. Using polynomial factorization and back substitution, solve the resulting system of equations.

Cost of the F5 Algorithm. In the best adversarial scenario, we assume that the sequence of polynomials associated with the system of equations is *regular*. A sequence of polynomials $(f_0, \dots, f_{m-1}) \in R_n^m$ is called a regular sequence on R_n if the multiplication map

$$m_{f_i} \colon R_n/\langle\{f_0, \dots, f_{i-2}\}\rangle \to R_n/\langle\{f_0, \dots, f_{i-2}\}\rangle$$

given by $m_{f_i}([g]) = [g][f_i] = [gf_i]$ is injective for all $i \in [3, m-1]$. In this case, the F5 algorithm does not perform any redundant reductions to zero.

Write $R_{n,d}$ for the set of homogeneous polynomials of degree d in R_n and I_d for the set of homogeneous polynomials of degree d in I. The *Hilbert function* is defined as $F_{R_n/I}(d) = \dim(R_{n,d}/I_d)$ for all $d \geq 0$, i.e., it maps d to the dimension of $R_{n,d}/I_d$ as a vector space over k. Define the *Hilbert series* by $H_{R_n/I}(t) = \sum_{d=0}^{\infty} F_{R_n/I}(d)t^d$. For sufficiently large d, the Hilbert function agrees with a univariate polynomial in d over the rational numbers, called the *Hilbert polynomial*. The *degree of regularity* D_{reg} is the smallest integer such that this is true. The quantity D_{reg} plays an important role in the cost of the algorithm. The *degree* of I, denoted by $\deg(I)$, is the dimension of R_n/I as a k-vector space. If the ideal I is generated by a regular sequence of degrees $d_0, \dots, d_{m-1}$, then its Hilbert series equals

$$\begin{aligned} H_{R_n/I}(t) &= \frac{\prod_{i=0}^{m-1}(1 + t + t^2 + \cdots + t^{d_i - 1})}{(1-t)^{n-m+1}} \\ &= \frac{\prod_{i=0}^{m-1}(1 - t^{d_i})}{(1-t)^n}. \end{aligned}$$

From this, we deduce that $\deg(I) = \prod_{i=0}^{m-1} d_i$ and $D_{\text{reg}} = 1 + \sum_{i=0}^{m-1}(d_i - 1)$.

The main result is that if $f_0, \ldots, f_{m-1}$ is a regular sequence on R_n, then computing a Gröbner basis with respect to the grevlex order using the F5 algorithm can be performed within

$$\mathcal{O}\left(\binom{n + D_{\text{reg}}}{D_{\text{reg}}}^{\omega}\right)$$

operations in k, where $\omega \in [2, 3]$ is the matrix multiplication exponent.

Costs of Gröbner basis conversion and of back substitution. FGLM is an algorithm that converts a Gröbner basis for I with respect to one order, to a Gröbner basis for I with respect to a second order in $\mathcal{O}(n \deg(I)^3)$ operations in k. Finally, if k equals $\mathbb{F}_{p^n}$, the cost of factoring a univariate polynomial in $\mathbb{F}_{p^n}[x]$ of degree d is $\mathcal{O}(d^3n^2 + dn^3)$, as proved in [29].

2.6 Integral cryptanalysis

Throughout this paper, we use *integral attacks* as an umbrella term for attacks relying on summing the outputs of a function over a well-chosen input set, each using a different heuristic for constructing the set. We restrict ourselves to input sets that form an affine space over the field $\mathbb{F}_2$.

This section is organized as follows. In subsection 2.6.1, we make explicit the link between functions defined on an affine space and their representation on this space as a multivariate polynomial, called the algebraic normal form. In subsection 2.6.2, we introduce an intuitive notion of the derivative of a function and show how it can be computed by means of summation of outputs of the function. Finally, we present the high-level idea behind an integral attack in subsection 2.6.4.

2.6.1 Algebraic normal form

To understand how to find input spaces for an integral attack, we need to explain how to represent the restriction of a vectorial Boolean function to some affine space as a tuple of multivariate polynomials: the algebraic normal form (ANF). We present the necessary tools and results from computational commutative algebra and make the relation between the algebraic normal form and substitutions, which determine the input sets, explicit.

Variables correspond to the bits which are controlled by an adversary, e.g., the diversifier symbols. We also refer to these symbols as the input symbols. Let $p_0, \ldots, p_{n-1}$ be polynomials of the form $p_i = x_i$ or $p_i = c_i + \sum_{j=i+1}^{n-1} a_{ij}x_j$ for constants $c_i \in \mathbb{F}_2$ and coefficients $a_{ij} \in \mathbb{F}_2$. During cryptanalysis, we make use of a set of rewrite rules of the form $x_i \to p_i$, i.e., we *substitute* x_i with the polynomial p_i. Rules of the form $x_i \to x_i$ are said to be *trivial* in the sense that no substitution is performed. A set of rewrite rules defines a set of polynomials of the form $x_i - p_i$, which is completely specified by a tuple (A, c), where $A = (a_{ij})$ is an $n \times n$ matrix over $\mathbb{F}_2$ and $c = (c_0, \ldots, c_{n-1})$ is a vector in $\mathbb{F}_2^n$. The matrix A is in row echelon form, up to a permutation of its rows, which implies that the order in which the corresponding rewrite rules are

applied does not matter. The tuple (A, c) defines the affine space $V = \{v \in \mathbb{F}_2^n : Av = c\}$ of points that satisfy the equation $Av = c$.

We have seen that a rewrite rule of the form $x_i \to p_i$ give us a relation of the form $x_i = p_i$. Moreover, we have relations of the form $x_i^2 = x_i$ due to the fact that the square on $\mathbb{F}_2$ is the identity map. We can introduce these relations by working with polynomials modulo the ideal I generated by the set

$$G = \{x_0^2 - x_0, \dots, x_{n-1}^2 - x_{n-1}, x_0 - p_0, \dots, x_{n-1} - p_{n-1}\}.$$

For our purposes, the central algebraic object is the quotient ring R_n/I.

Polynomials in R_n give rise to elements of Maps$[V, \mathbb{F}_2]$. Indeed, for any point $a \in V$, there is a unique ring homomorphism $\varepsilon_a \colon R_n \to \mathbb{F}_2$ with $\varepsilon_a(x_i) = a_i$ given by substituting x_i by a_i. This leads to a map $\phi \colon R_n \to \mathbb{F}_2^V$ that is defined by $\phi(p) = f$ with $f(a) = \varepsilon_a(p)$ for all $a \in V$. The kernel of ϕ is equal to I. By the first isomorphism theorem for rings [13, p. 247], there is an isomorphism $\overline{\phi}$ between $\mathbb{F}_2^V$ and R_n/I.

The set G forms a Gröbner basis [13, p. 78] for I with respect to the lexicographic order. Define $W = \{u \in \mathbb{F}_2^n : u_i = 0 \text{ if } x_i \neq p_i\}$ as the set of vectors for which the ith component is zero if x_i is eliminated by a substitution. The remainder of any polynomial $p \in R_n$ on division by G is unique and of the form

$$p \operatorname{rem} G = \sum_{u \in W} \alpha_u x^u,$$

for certain constant bits $\alpha_u \in \mathbb{F}_2$ [13, p. 83]. Therefore, the set of all possible remainders after division by G, which we denote as R_G, forms a complete set of coset representatives of I in R_n. Indeed, let $\psi \colon R_n \to R_n$ be defined by $\psi(p) = p \operatorname{rem} G$ for all $p \in R_n$. The kernel of ψ is equal to I. By the first isomorphism theorem for rings, there is an isomorphism $\overline{\psi}$ between R_n/I and R_G.

To conclude, we have an isomorphism $\mathcal{N}_G = \overline{\psi} \circ \overline{\phi}$ between the set of Boolean functions defined on V and the set of remainders R_G. We are now able to make precise how a function is represented on V.

Definition 8. *Let $f \colon V \to \mathbb{F}_2$ be a Boolean function defined on V. The representation of f as a multivariate polynomial, called the algebraic normal form (ANF) of f, is defined as $\mathcal{N}_G(f)$.*

The degree of a remainder $p \in R_G$ with $p \neq 0$ is equal to $\deg(p) = \max\{\mathrm{HW}(u) : u \in W \text{ and } \alpha_u \neq 0\}$, which follows from the definition and the algebraic relations between the variables.

Definition 9. *Let $f \colon V \to \mathbb{F}_2$ be a Boolean function defined on V. The algebraic degree of f, denoted by* $\deg(f)$*, is defined as the degree of its ANF.*

If f depends on a secret vector $s \in \mathbb{F}_2^\kappa$, for some integer $k \geq 1$, e.g., a secret key or state, then the coefficients α_u of $\mathcal{N}(f)$ are Boolean functions of the secret bits, i.e., α_u maps the secret s to some bit $\alpha_u(s) \in \mathbb{F}_2$. In this case, we can rewrite the definition of the degree as $\deg(f) = \max\{\mathrm{HW}(u) : u \in W \text{ and there exists an } s \in \mathbb{F}_2^\kappa \text{ with } \alpha_u(s) \neq 0\}$. Note that our definitions coincide with the usual definitions of algebraic normal form and algebraic degree in the case that both A and c are zero.

Table 2.2: Truth table of f.

x	000	001	010	011	100	101	110	111
$f(x)$	0	1	0	0	1	0	1	1

There is a straightforward generalization of these notions to vectorial Boolean functions defined on V.

Definition 10. *The algebraic normal form of* $F = (f_0, \dots, f_{m-1}) \colon V \to \mathbb{F}_2^m$ *is defined as* $\mathcal{N}(F) := (\mathcal{N}(f_0), \dots, \mathcal{N}(f_{m-1})) \in R_n^m$. *Its algebraic degree is defined as* $\deg(F) := \max\{\deg(f_0), \dots, \deg(f_{m-1})\}$.

We illustrate how to apply rewrite rules to $\mathcal{N}(f)$, where f is some Boolean function, in order to change its properties, such as the presence of certain monomials. The resulting polynomial is the ANF of the restriction of f to the affine space determined by the rewrite rules.

Example 2. *The function* $f \colon \mathbb{F}_2^3 \to \mathbb{F}_2$ *is defined by the truth table in Table 2.2. It follows that*

$$\mathcal{N}(f)(x_0, x_1, x_2) = x_0 + x_2 + x_1 x_2 \,.$$

Therefore, the algebraic degree of f *is* 2. *Now we make the isomorphism* $\mathcal{N}$ *implicit.*

We apply the rewrite rule $x_1 \to x_2$. *This rule, together with the trivial rules, defines the matrix*

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

and the constant $c = (0, 0, 0)$. *Clearly,* A *is in row echelon form, up to a permutation of its rows. Moreover,* $V = \{v \in \mathbb{F}_2^3 : Av = 0\} = \{v \in \mathbb{F}_2^3 : v_1 = v_2\}$. *When we restrict* f *to* V, *i.e., when we consider* $f|_V \colon V \to \mathbb{F}_2$, *we find that its ANF is equal to* x_0. *The restriction has algebraic degree* 1 *and it depends on a single variable.*

An alternative way of wording this is that we compose f *with the map* $L \colon \mathbb{F}_2^2 \to V$ *given by* $(x_0, x_1) \mapsto (x_0, x_1, x_1)$ *and that the algebraic normal form of* $f \circ L$ *is equal to* x_0.

Like in the example, we will make the correspondence between Boolean functions and their representation as a tuple of remainders implicit in the following sections.

2.6.2 Properties of derivatives

The integral attacks that we consider in this section, rely on practically computable properties of the derivative of a Boolean function. All definitions and results are extended to the case of vectorial Boolean functions by applying them to each coordinate Boolean function. We define a partial order $\leq$ on $\mathbb{F}_2^n$ by declaring $u \leq v$ if and only if $u_i \leq v_i$ for $i = 0, \ldots, n-1$, interpreting elements of $\mathbb{F}_2$ as the integers 0 and 1 and employing the standard total order on $\mathbb{Z}$.

Definition 11. *For vectors $u, v \in \mathbb{F}_2^n$, define the derivative of the monomial x^v with respect to u by*

$$\partial_u x^v = \begin{cases} x^{v-u} & \text{if } u \leq v\,, \\ 0 & \text{otherwise}\,, \end{cases}$$

and extend linearly to functions $f\colon \mathbb{F}_2^n \to \mathbb{F}_2$. We call $\partial_u f$ the derivative of f with respect to u.

Note that this definition coincides with that of the usual partial derivative.

Example 3. *Let $f\colon \mathbb{F}_2^3 \to \mathbb{F}_2$ be given by $f(x) = x_0 + x_2 + x_1x_2$. Its derivatives are equal to*

$$\begin{aligned}
\partial_{(0,0,0)} f(x) &= x_0 + x_2 + x_1x_2 \\
\partial_{(0,0,1)} f(x) &= x_1 + 1 \\
\partial_{(0,1,0)} f(x) &= x_2 \\
\partial_{(0,1,1)} f(x) &= 1 \\
\partial_{(1,0,0)} f(x) &= 1 \\
\partial_{(1,0,1)} f(x) &= 0 \\
\partial_{(1,1,0)} f(x) &= 0 \\
\partial_{(1,1,1)} f(x) &= 0
\end{aligned}$$

The first important property of the derivative is the duality between the derivatives of f and outputs of f on an affine space by means of summation.

Proposition 8. *Let $f\colon \mathbb{F}_2^n \to \mathbb{F}_2$ and $a, u \in \mathbb{F}_2^n$. We have*

$$\begin{aligned}
f(x+a) &= \sum_{0 \leq u \leq a} \partial_u f(x)\,, \text{ and} \\
\partial_u f(x) &= \sum_{0 \leq a \leq u} f(x+a)\,.
\end{aligned}$$

Proof. The first equality can be seen as follows. Using the ANF of f, we find that

$$\begin{aligned} f(x+a) &= \sum_{0\le w} \alpha_w (x+a)^w \\ &= \sum_{0\le w} \alpha_w \left(\sum_{0\le u\le w} x^{w-u} a^u \right) \\ &= \sum_{0\le w} \left(\sum_{0\le u\le w} \alpha_w x^{w-u} a^u \right) \\ &= \sum_{0\le u} \left(\sum_{u\le w} \alpha_w x^{w-u} a^u \right) \\ &= \sum_{0\le u} \left(\sum_{u\le w} \alpha_w x^{w-u} \right) a^u \\ &= \sum_{0\le u\le a} \left(\sum_{u\le w} \alpha_w x^{w-u} \right) \\ &= \sum_{0\le u\le a} \partial_u f(x)\,, \end{aligned}$$

where we have applied the definition of the derivative and used the fact that $a^u = 1$ if and only if $0 \le u \le a$. The second equality follows from the Möbius inversion formula [40, p. 264] applied to the first. □

The following corollary shows how to compute the coefficient α_u of x^u in f by summing over the outputs of f corresponding to inputs for which u takes on all possible values.

Corollary 2. *Let $f\colon \mathbb{F}_2^n \to \mathbb{F}_2$ and $a, u \in \mathbb{F}_2^n$. We have*

$$\alpha_u = \sum_{0\le a\le u} f(a)\,.$$

Proof. This follows from the second equality in 8 and the fact that $\partial_u f(0) = \alpha_u$, by definition. □

The second important property of the derivative concerns its degree.

Proposition 9. *The degree of the derivative of f with respect to u satisfies*

$$\deg(\partial_u f) \le \deg(f) - \mathrm{HW}(u)\,.$$

Proof. By definition, we have $\partial_u f = \sum_{u\le v} \alpha_v x^{v-u}$. Let w be such that $\alpha_w \neq 0$ and $\deg(\partial_u f) = \mathrm{HW}(w-u)$. Using that $u \le w$ and that x^w is a monomial in f, we find that $\deg(\partial_u f) = \mathrm{HW}(w-u) = \mathrm{HW}(w) - \mathrm{HW}(u) \le \deg(f) - \mathrm{HW}(u)$. □

The coefficient of any monomial x^u with the Hamming weight of u exceeding the degree of the function is 0.

Proposition 10. *If $\mathrm{HW}(u) \ge \deg(f(x))$, then $\partial_u f(x)$ is the coefficient α_u of x^u in f. In particular, if $\mathrm{HW}(u) > \deg(f(x))$, then this coefficient α_u is 0.*

Proof. If $\mathrm{HW}(u) \geq \deg\big(f(x)\big)$, then $\deg\big(\partial_u f(x)\big) \leq 0$. This implies that $\partial_u f(x)$ is a constant, i.e., $\partial_u f(x) = \partial_u f(a)$ for any $a \in \mathbb{F}_2^n$. In particular, this is true for a equal to 0. By definition, it follows that $\partial_u f(0) = \alpha_u$. If $\mathrm{HW}(u) > \deg\big(f(x)\big)$, then $\deg\big(\partial_u f(x)\big) < 0$, which implies that α_u is 0. □

2.6.3 Division properties

The various division properties are generalizations of the integral properties from [33]. In some cases, they allow us to accurately predict the coefficient of a monomial in the ANF of a function.

Definition 12. *Let X be a multisubset of $\mathbb{F}_2^n$ and let K be a subset of $\mathbb{F}_2^n$. We say that X has the K-division property if for all $u \in \mathbb{F}_2^n$ we have that*

$$\sum_{x \in X} x^u = \begin{cases} \textit{unknown} & \textit{if there exists a } k \in K \textit{ such that } k \leq u\,, \\ 0 & \textit{otherwise.} \end{cases}$$

We can trade computation time for a higher resolution.

Definition 13. *Let X be a multisubset of $\mathbb{F}_2^n$ and let K and L be subsets of $\mathbb{F}_2^n$. We say that X has the (K, L)-division property if for all $u \in \mathbb{F}_2^n$ we have that*

$$\sum_{x \in X} x^u = \begin{cases} \textit{unknown} & \textit{if there exists a } k \in K \textit{ such that } k \leq u\,, \\ 1 & \textit{if there exists an } l \in L \textit{ such that } l = u\,, \\ 0 & \textit{otherwise.} \end{cases}$$

Propagation rules. Any function $f\colon \mathbb{F}_2^n \to \mathbb{F}_2^m$ can be modeled as the composition of "copy and expand", "multiply and compress", "add and compress", and "secret key addition" functional blocks. Hence, we give rules for the propagation of the (K, L)-division property through each of these blocks. In this paragraph, the binary operator + denotes addition in $\mathbb{Z}$ and the binary operator $\oplus$ denotes addition in $\mathbb{F}_2$. In other words, to compute the propagation of a division property, we treat bits as integers.

Copy and expand. Consider the propagation through the copy and expand function

$$(x \mapsto (x_0, x_0, x_1, \ldots, x_{n-1}))\colon \mathbb{F}_2^n \to \mathbb{F}_2^{n+1}\,.$$

Suppose that the multiset at the input has the (K, L)-division property. The multiset at the output has the (K', L')-division property, where K' and L' are computed as follows. Write $A \leftarrow a$ for the insertion of the element a into the multiset A. For each $k \in K$, do

$$K' \leftarrow \begin{cases} (0, 0, k_1, \ldots, k_{n-1}) & \text{if } k_0 = 0\,, \\ (1, 0, k_1, \ldots, k_{n-1}), (0, 1, k_1, \ldots, k_{n-1}) & \text{if } k_0 = 1\,. \end{cases}$$

For each $l \in L$, do

$$L' \leftarrow \begin{cases} (0, 0, l_1, \dots, l_{n-1}) & \text{if } l_0 = 0\,, \\ (1, 0, l_1, \dots, l_{n-1}), (0, 1, l_1, \dots, l_{n-1}), (1, 1, l_1, \dots, l_{n-1}) & \text{if } l_0 = 1\,. \end{cases}$$

Multiply and compress. Consider the propagation through the multiply and compress function

$$(x \mapsto (x_0 x_1, x_2, \dots, x_{n-1}))\colon \mathbb{F}_2^n \to \mathbb{F}_2^{n-1}\,.$$

Suppose that the multiset at the input has the (K, L)-division property. The multiset at the output has the (K', L')-division property, where K' and L' are computed as follows. For each $k \in K$, do

$$K' \leftarrow \left(\left\lfloor \frac{k_0 + k_1}{2} \right\rfloor, k_2, \dots, k_{n-1}\right).$$

For each $l \in L$ such that (l_0, l_1) equals $(0, 0)$ or $(1, 1)$, do

$$L' \leftarrow \left(\left\lfloor \frac{l_0 + l_1}{2} \right\rfloor, l_2, \dots, l_{n-1}\right).$$

Add and compress. Consider the propagation through the add and compress function

$$(x \mapsto (x_0 \oplus x_1, x_2, \dots, x_{n-1}))\colon \mathbb{F}_2^n \to \mathbb{F}_2^{n-1}\,.$$

Suppose that the multiset at the input has the (K, L)-division property. The multiset at the output has the (K', L')-division property, where K' and L' are computed as follows, for each $k \in K$ and $l \in L$, respectively. For each $k \in K$ such that (k_0, k_1) equals $(0, 0)$, $(1, 0)$, or $(0, 1)$, do

$$K' \leftarrow (k_0 + k_1, k_2, \dots, k_{n-1})\,.$$

Write $A \rightleftarrows a$ for the insertion of the element a into the multiset A if it is not present and removal otherwise. For each $l \in L$ such that (l_0, l_1) equals $(0, 0)$, $(1, 0)$, or $(0, 1)$, do

$$L' \rightleftarrows (l_0 + l_1, l_2, \dots, l_{n-1})\,.$$

This propagation rule has the so-called *cancellation* property; in effect, we are only keeping those vectors that appear an odd number of times.

Secret key addition. Let $i \in [0, n-1]$. Consider the propagation through the function

$$(x \mapsto (x_0, \dots, x_{i-1}, x_i + s_i, x_{i+1}, \dots, x_{n-1}))\colon \mathbb{F}_2^n \to \mathbb{F}_2^n$$

that adds a secret bit s_i to x_i. For every $l \in L$ with $l_i = 0$, we insert the vector $(l_0, \dots, l_{i-1}, 1, l_{i+1}, \dots, l_{n-1})$ into K. This propagation rule has the so-called *unknown-producing* property.

Algorithms. Suppose that we can write $f\colon \mathbb{F}_2^n \to \mathbb{F}_2^m$ as $f = \bigcirc_{i=0}^{l-1} \mathrm{R}_i$ for some positive integer $l \geq 1$ and functional blocks R_i of the types that were described in the previous paragraph and with compatible domains and codomains. Using the appropriate propagation rule of the previous paragraph, we compute the (K_{i+1}, L_{i+1})-division property at the output of R_i from the (K_i, L_i)-division property at the input of R_i for all $i \in [0, l-1]$. Conceptually, this leads to the following ordered $(l+1)$-partite graph $G = (V, E)$. Put $V_i = \{(i, v) : v \in K_i \cup L_i\}$ for the ith level, where we put the index to make the nodes unique. The set of nodes is equal to $V = \cup_{i=0}^{l} V_i$. The set of edges E is formed by all unordered pairs $\{(i, v), (i+1, w)\}$ $(i \in [0, l-1])$ for which w is generated by v through a propagation rule. We summarize several algorithms for finding sets of paths in, or, more generally, subgraphs of G.

Breadth-first search. The *breadth-first search* (BFS) algorithm traverses the graph level by level by applying the propagation rules to generate the nodes. Because it explores the entire graph, it requires generation of every node, which makes it very expensive. Hence, we mention it for historical reasons and because it is conceptually the easiest algorithm to understand.

Binary integer programming. By combining functional blocks and propagation rules, we may assume that the R_i are the round functions of f. In this context, an l-round *division trail* is a path in the graph that contains one node from each of the levels. We can specify a start node and an end node, typically a standard basis vector. This determines a set of division trails. We can efficiently model this set as a system of linear inequalities in variables that are required to be either 0 or 1. This is called a *binary integer programming* (BIP) model. Various models (for various division properties) exist with different solving times. See, for example, [22] and [30]. At the time of writing a popular solver for BIP models is the one by Gurobi.

Graph pruning. In many cases, we do not need to visit every node in the graph to be able to answer a query like "does the ith output bit of f sum to zero over a given multiset X or not?" Like before, we use the BFS algorithm to traverse each of the l levels. However, when we traverse the jth level, we use a BIP model of the propagation of the K-division property through $l - j$ rounds of f to efficiently identify subgraphs that we do not need to visit. For more details, we refer to [43].

2.6.4 Framework of an integral attack

Integral attacks consist of an offline phase followed by an online phase:

Offline phase. The offline phase is an analysis step where the adversary accesses the polynomial representations of the step functions of a primitive that depends on some secret. They apply rewrite rules to these polynomial representations in order to simplify it, e.g., eliminating variables and lowering the degree. Importantly, the rewrite rules determine an affine input space V. Using combinatorial arguments involving the degree or by propagating an initial division property vector [41],

the adversary is able to determine the vector of coefficients of some target monomial. To be able to mount a successful attack, this vector should either be a constant that does not depend on the secret at all or depend on the secret in a way that leads to a system of equations that is easy to solve, e.g., linear dependence. The outcome of this step is an affine input space V and a target monomial x^u.

Online phase. The online phase is an execution step where the adversary accesses a cryptographic oracle for a fixed secret. They recover the vector of coefficients of the target monomial x^u by summing over the affine input space V that was obtained during the offline phase. The vector of coefficients is then used as a distinguisher or to set up a system of equations in the secret bits that may lead to recovery of the secret.

2.7 Differential cryptanalysis

Suppose that G and H are finite abelian groups and consider a function $f\colon G \to H$ between G and H. Differential cryptanalysis of f is about answering the following question. Given a uniform random element $x \xleftarrow{\$} G$ and a fixed element $\Delta_{\text{in}} \in G$, what is the shape of the distribution of the random element $f(x) - f(x + \Delta_{\text{in}}) \in H$? For example, what is the probability that the random element equals $\Delta_{\text{out}} \in H$? To study this question systematically, we introduce some basic terminology.

2.7.1 Differential probability

The tuple $(\Delta_{\text{in}}, \Delta_{\text{out}}) \in G \times H$ is called a differential over f, Δ_{in} is called an input difference, and Δ_{out} is called an output difference. The main quantity of interest is the probability that a given differential occurs.

Definition 14. *The* differential probability *(*DP*) of* $(\Delta_{in}, \Delta_{out})$ *is defined as*

$$\mathrm{DP}_f(\Delta_{in}, \Delta_{out}) := \Pr[f(x) - f(x + \Delta_{in}) = \Delta_{out}] .$$

The definition provides a concrete method to compute DP_f by counting the number of solutions to the differential equation. This motivates the definition of a solution set.

Definition 15. *The solution set of* $(\Delta_{in}, \Delta_{out})$ *is the set*

$$Z_f(\Delta_{in}, \Delta_{out}) := \{x \in G : f(x) - f(x + \Delta_{in}) = \Delta_{out}\} .$$

Because x is a uniform random element, we readily obtain the following correspondence between DP_f and the cardinality of the solution set.

Proposition 11. *The DP of* $(\Delta_{in}, \Delta_{out})$ *is equal to*

$$\mathrm{DP}_f(\Delta_{in}, \Delta_{out}) = \frac{|Z_f(\Delta_{in}, \Delta_{out})|}{|G|}.$$

Tuples $(x, x + \Delta_{\mathrm{in}})$ with $x \in Z_f(\Delta_{\mathrm{in}}, \Delta_{\mathrm{out}})$ are said to *follow* the differential $(\Delta_{\mathrm{in}}, \Delta_{\mathrm{out}})$. If such tuples exist, then we say that the input difference Δ_{in} is *compatible* with the output difference Δ_{out} over f and call $(\Delta_{\mathrm{in}}, \Delta_{\mathrm{out}})$ a *valid* differential.

2.7.2 Differential trails

Suppose now that f is obtained as the composition of k round functions. That is, we assume that

$$f = \bigcirc_{i=0}^{k-1} \mathrm{R}_i.$$

Here, $\mathrm{R}_i \colon G_i \to G_{i+1}$ is a function between finite abelian groups G_i and G_{i+1}. We write $f[r] := \mathrm{R}_{r-1} \circ \cdots \circ \mathrm{R}_0$ and define $f[0] := \mathrm{id}$ with id the identity function. To study differentials over f, we study the propagation of differences through the R_i.

Definition 16. *A k-round* differential trail *over f is a sequence*

$$Q = (q^{(0)}, q^{(1)}, \ldots, q^{(k)}) \in \prod_{i=0}^{k} G_i$$

that satisfies $\mathrm{DP}_{\mathrm{R}_i}(q^{(i)}, q^{(i+1)}) > 0$ *for* $i = 0, \ldots, k-1$.

Let us further suppose that $\mathrm{R}_i = \iota_i \circ \mathrm{L}_i \circ \mathrm{N}_i$ is the composition of a round constant addition, a linear layer, and a non-linear layer. Sometimes we specify a higher-resolution differential trail as

$$Q := (b_{-1}, a_0, b_0, b_0, \ldots, a_k, b_k, b_k)$$

by giving the intermediate differences between N_i and L_i as well. They are related by the equations $b_i = \mathrm{L}_i(a_i) = q_{i+1}$ for $i = 0, \ldots, k-1$.

We write $\mathrm{DT}(\Delta_{\mathrm{in}}, \Delta_{\mathrm{out}})$ for the set of all differential trails in the differential $(\Delta_{\mathrm{in}}, \Delta_{\mathrm{out}})$. These are the trails with $q^{(0)} = \Delta_{\mathrm{in}}$ and $q^{(k)} = \Delta_{\mathrm{out}}$. We call $(\Delta_{\mathrm{in}}, \Delta_{\mathrm{out}})$ the *enveloping differential* of the trails in $\mathrm{DT}(\Delta_{\mathrm{in}}, \Delta_{\mathrm{out}})$. If $|\mathrm{DT}(\Delta_{\mathrm{in}}, \Delta_{\mathrm{out}})| \geq 2$, then we say that trails *cluster* together in the differential $(\Delta_{\mathrm{in}}, \Delta_{\mathrm{out}})$. By deleting the initial difference Δ_{in} and final difference Δ_{out} of a differential trail we are left with a *differential trail core* A differential trail core obtained in this way is said to be in the differential $(\Delta_{\mathrm{in}}, \Delta_{\mathrm{out}})$. Note that a differential trail core actually defines a set of differential trails with the same inner differences.

Let E_i be the event that corresponds to the predicate $\mathrm{R}_i(f[i](x)) - \mathrm{R}_i(f[i](x) + q^{(i)}) = q^{(i+1)}$. We now define the DP of a differential trail.

Definition 17. *The* DP *of a differential trail is defined as*

$$\mathrm{DP}_f(Q) \coloneqq \prod_{i=0}^{k} \Pr\left[E_i \;\middle|\; \bigcap_{j=0}^{i-1} E_j\right].$$

Each round differential $(q^{(i)}, q^{(i+1)})$ has a solution set $Z_{\mathrm{R}_i}(q^{(i)}, q^{(i+1)})$. Consider the transformed set of points $Z_i \coloneqq f[i]^{-1}(Z_{\mathrm{R}_i}(q^{(i)}, q^{(i+1)}))$ at the input of f. For a tuple $(x, x+q^{(0)})$ to follow the differential trail, it is required that $x \in Z_f(Q) \coloneqq \bigcap_{i=0}^{k-1} Z_i$. The DP of the trail is the fraction of states x that satisfy this equation.

Proposition 12. *The* DP *of a differential trail is equal to*

$$\mathrm{DP}_f(Q) = \frac{|Z_f(Q)|}{|G|}.$$

Definition 18. *The round differentials are said to be* independent *if the corresponding events are, i.e., if*

$$\begin{aligned}\mathrm{DP}_f(Q) &= \prod_{i=0}^{k-1} \Pr[E_i] \\ &= \prod_{i=0}^{k-1} \mathrm{DP}_{\mathrm{R}_i}(q^{(i)}, q^{(i+1)}).\end{aligned}$$

Any given tuple $(x, x + \Delta_{\mathrm{in}})$ follows exactly one differential trail. Hence, the DP of the differential $(\Delta_{\mathrm{in}}, \Delta_{\mathrm{out}})$ is the sum of the DPs of all differential trails with initial difference Δ_{in} and final difference Δ_{out}.

Proposition 13.

$$\mathrm{DP}_f(\Delta_{in}, \Delta_{out}) = \sum_{Q \in DT(\Delta_{in}, \Delta_{out})} \mathrm{DP}_f(Q).$$

Proof.

$$\begin{aligned}\mathrm{DP}_f(\Delta_{\mathrm{in}}, \Delta_{\mathrm{out}}) &= \Pr[f(x) - f(x + \Delta_{\mathrm{in}}) = \Delta_{\mathrm{out}}] \\ &= \sum_{Q \in \mathrm{DT}(\Delta_{\mathrm{in}}, \Delta_{\mathrm{out}})} \Pr\left[\{f(x) - f(x + \Delta_{\mathrm{in}}) = \Delta_{\mathrm{out}}\} \cap \bigcap_{i=0}^{k-1} E_i\right] \\ &= \sum_{Q \in \mathrm{DT}(\Delta_{\mathrm{in}}, \Delta_{\mathrm{out}})} \mathrm{DP}_f(Q).\end{aligned}$$

□

Given any differential $(\Delta_{\mathrm{in}}, \Delta_{\mathrm{out}})$ over a round function R, it is typically easy to compute its DP value. By specifying the intermediate differences we obtain a differential trail $(\Delta_{\mathrm{in}}, b, c, \Delta_{\mathrm{out}})$. Suppose that N is the application of m S-boxes in parallel. Thanks to the linearity of L, we have $c = \mathrm{L}(b)$ and due to the

fact that a difference is invariant under addition of a constant, all valid such differential trails are of the form $(\Delta_{\text{in}}, \mathrm{L}^{-1}(\Delta_{\text{out}}), \Delta_{\text{out}}, \Delta_{\text{out}})$. Therefore, the differential $(\Delta_{\text{in}}, \Delta_{\text{out}})$ contains only a single trail and its DP is the DP of the differential $(\Delta_{\text{in}}, \mathrm{L}^{-1}(\Delta_{\text{out}}))$ over N:

$$\mathrm{DP}_{\mathrm{R}}(\Delta_{\text{in}}, \Delta_{\text{out}}) = \prod_{0 \le j < m} \mathrm{DP}_{\mathrm{S}_j}(\mathrm{Proj}_j(\Delta_{\text{in}}), \mathrm{Proj}_j(\mathrm{L}^{-1}(\Delta_{\text{out}}))) .$$

Hence, the DP of a round differential is the product of the DP values of its S-box differentials.

2.7.3 Restriction weight

The DP of a differential $(\Delta_{\text{in}}, \Delta_{\text{out}})$ depends on the cardinality of its solution set. In the following, we suppose that G and H are vector spaces of dimension n over a finite field $\mathbb{F}_q$. If $Z_f(\Delta_{\text{in}}, \Delta_{\text{out}})$ is an affine space, then the DP is equal to $q^{\dim Z_f(\Delta_{\text{in}}, \Delta_{\text{out}}) - n}$. This inspires the following definition.

Definition 19. *The* restriction weight *of a differential* $(\Delta_{in}, \Delta_{out})$ *that satisfies* $\mathrm{DP}_f(\Delta_{in}, \Delta_{out}) > 0$ *is defined as*

$$\mathrm{w_r}(\Delta_{in}, \Delta_{out}) = -\log_q \mathrm{DP}_f(\Delta_{in}, \Delta_{out}) .$$

The weight represents the number of independent linear equations over $\mathbb{F}_q$ that is necessary to describe the solution set.

A tuple $(x, x + q^{(0)})$ follows a trail $Q = (q^{(0)}, q^{(1)}, \ldots, q^{(k)})$ if and only if $x \in Z_f(Q)$. The solution set of each round differential can be defined by a number of equations that is equal to the weight of this round differential. For a differential trail, we sum the weights of the round differentials.

Definition 20. *The restriction weight of a differential trail* $Q = (q^{(0)}, q^{(1)}, \ldots, q^{(k)})$ *is defined as*

$$\mathrm{w_r}(Q) := \sum_{i=0}^{k-1} \mathrm{w_r}(q^{(i)}, q^{(i+1)}) .$$

We now explain the significance of this definition. If the round differentials are independent in the sense of Definition 18, then we have that $\mathrm{DP}_f(Q) = q^{-\mathrm{w_r}(Q)}$. In general, the approximation can not be good if $\mathrm{w_r}(Q) > n$, since $\mathrm{DP}_f(Q) \ge q^{-n}$ if Q has any tuples following it.

2.8 Linear cryptanalysis

Linear analysis of cryptographic primitives effectively is Fourier analysis on finite abelian groups. As such, the theory is well-understood and this section serves as a recap. The ideas that we present here are based on the works of Daemen [15], Baignères et al. [2], and Daemen and Rijmen [19]. Many of the proofs can be found in the book by Hou [31].

2.8.1 Characters

Let $(G, +)$ be a finite abelian group and let e be the (finite) exponent of G, i.e., the smallest integer n such that $na = 0$ for all $a \in G$.

A *character* of G is a homomorphism from G into the subgroup of $\mathbb{C}^*$ consisting of the eth roots of unity. The set of characters of G is denoted by $\hat{G}$ and it forms a group under the multiplication defined by $(\chi\chi')(a) = \chi(a)\chi'(a)$ for all $a \in G$ and $\chi, \chi' \in \hat{G}$. The groups G and $\hat{G}$ are isomorphic, but this isomorphism is not canonical.

For a fixed isomorphism between G and $\hat{G}$ and for each $a \in G$, we write χ_a for the image of a under this isomorphism. In particular, the character χ_0 that is defined by $\chi_0(a) = 1$ for all $a \in G$ is called the *trivial character* and it is the identity element of the group $\hat{G}$.

Now, let $(G, +, \cdot)$ be the commutative ring that is obtained by introducing a multiplicative structure on G. This is always possible by the fundamental theorem of finite abelian groups. A character $\chi \in \hat{G}$ is called a *generating character* for G if $\chi_a(b) = \chi(ab)$ for all $a, b \in G$. If a commutative ring has a generating character for its additive group, then $\chi_a(b) = \chi(ab) = \chi(ba) = \chi_b(a)$. In the case that G is the direct sum of n copies of a commutative ring R and if R has a generating character, say ϕ, then we obtain a generating character χ for G by setting $\chi(a_1, \dots, a_n) = \phi(a_1) \cdots \phi(a_n)$. It holds that $\chi_a(b) = \chi(ab) = \phi(a^\top b)$, where the multiplication in G is defined component-wise.

As an example, consider G equal to $\mathbb{F}_q$ with $q = p^d$ and put $\omega = e^{2\pi i/p}$. Let $\mathrm{Tr}\colon \mathbb{F}_q \to \mathbb{F}_p$ be the absolute trace function that is defined by $\mathrm{Tr}(x) = \sum_{i=0}^{d-1} x^{p^i}$. This is a linear mapping. Each $u \in \mathbb{F}_q$ defines a generating character χ_u for $\mathbb{F}_q$ that is defined by

$$\chi_u(x) = \omega^{\mathrm{Tr}(ux)}, \qquad x \in \mathbb{F}_q\,.$$

As a second example, consider G equal to $\mathbb{F}_q^n$, which is a direct sum of n copies of $\mathbb{F}_q$. Hence, each $u \in \mathbb{F}_q^n$ gives a generating character χ_u for $\mathbb{F}_q^n$ that is defined by

$$\chi_u(x) = \omega^{\mathrm{Tr}(u^\top x)}, \qquad x \in \mathbb{F}_q^n\,.$$

2.8.2 The Fourier transform

Consider the set $\mathbb{C}^G$ of functions $f\colon G \to \mathbb{C}$. Fix an ordering of the element of G, e.g., $G = \{a_0, \dots, a_{n-1}\}$. We write $v_f = (f(a_0), \dots, f(a_{n-1}))$ for the finite sequence of the output values of f. By identifying a function f with the vector $v_f \in \mathbb{C}^{|G|}$, $\mathbb{C}^G$ can be seen as a finite-dimensional complex inner product space with inner product

$$\langle f, g\rangle = \sum_{a\in G} f(a)\overline{g(a)}, \qquad f, g \in \mathbb{C}^G\,.$$

For any $f \in \mathrm{C}^G$, the inner product induces a norm by setting

$$\|f\| = \langle f, f \rangle^{\frac{1}{2}}.$$

The standard basis of C^G is formed by the set of Dirac delta functions $\{\delta_a \in \mathrm{C}^G : a \in G\}$, which are defined by

$$\delta_a(b) = \begin{cases} 1 & \text{if } a = b, \\ 0 & \text{if } a \neq b. \end{cases}$$

In the context of linear analysis, the solution to the problem of secret key translation lies in changing the basis of C^G to the set of characters of G. For any $a, b \in G$, the corresponding characters satisfy $\langle \chi_a, \chi_b \rangle = |G| \delta_a(b)$. By normalizing the characters, we obtain an orthonormal basis

$$\Phi_G = \{\phi_a : a \in G\},$$

where $\phi_a = |G|^{-\frac{1}{2}} \chi_a$. By projecting f onto Φ_G, we find that

$$f = \sum_{a \in G} \langle f, \phi_a \rangle \phi_a.$$

The operator $\mathcal{F} \colon \mathrm{C}^G \to \mathrm{C}^G$ that is defined by $\mathcal{F}(f)(a) = \langle f, \phi_a \rangle$ for all $a \in G$ is called the *Fourier transform*. By identifying a function f with v_f, the Fourier transform is best described as assigning to f its coordinates in the normalized character basis. The *Plancherel theorem* asserts that the Fourier transform is unitary, i.e., we have

$$\langle \mathcal{F}(f), \mathcal{F}(g) \rangle = \langle f, g \rangle, \qquad f, g \in \mathrm{C}^G.$$

Let us return to the question of how to address the problem of secret key translation. Let $b \in G$. We define the translation operator $T_b \colon \mathrm{C}^G \to \mathrm{C}^G$ by $(T_b f)(a) = f(a + b)$ for all $a \in G$. Moreover, we define the modulation operator $M_b \colon \mathrm{C}^G \to \mathrm{C}^G$ by $(M_b f)(a) = \phi_b(a) f(a)$ for all $a \in G$. The big insight is that translation turns into modulation when changing from the standard basis to the normalized character basis, i.e.,

$$T_b = \mathcal{F}^{-1} \circ M_b \circ \mathcal{F}, \qquad b \in G.$$

Let H be a finite abelian group and let $F \colon G \to H$ be a mapping between G and H. We want a representation of F in C^G. To that end, let χ be any character of H. We take as representation the function $\chi \circ F \in \mathrm{C}^G$.

2.8.3 Correlation

We now specialize to the case that G and H are each equal to the vector space $\mathbb{F}_q^n$ over the finite field $\mathbb{F}_q$.

Let $\alpha\colon \mathbb{F}_q^n \to \mathbb{F}_q^n$ be a transformation of $\mathbb{F}_q^n$. We consider pairs $(u, v) \in \mathbb{F}_q^n \times \mathbb{F}_q^n$ that we call *linear approximations* of α. We refer to u as the *output mask* and to v as the *input mask*. The linear approximation $(0, 0)$ is called *trivial*. The *correlation* of the linear approximation is defined as

$$\mathrm{C}_\alpha(u, v) = q^{-\frac{n}{2}} \mathcal{F}(\chi_u \circ \alpha)(v) \,.$$

We call the masks u and v *compatible* over α if $\mathrm{C}_\alpha(u, v)$ is nonzero. In general, correlations are complex numbers.

2.8.4 Linear trails

Suppose now that α is obtained as the composition of k round functions. That is, we assume that

$$\alpha = \bigcirc_{i=0}^{k-1} \mathrm{R}_i \,.$$

The analysis of a linear approximation of α relies on linear approximations of its rounds. This naturally leads to the notion of a linear trail [14].

Definition 21. *A sequence* $Q = (q^{(0)}, q^{(1)}, \ldots, q^{(k)}) \in (\mathbb{F}_q^n)^{k+1}$ *that satisfies* $\mathrm{C}_{\mathrm{R}_i}(q^{(i)}, q^{(i+1)}) \neq 0$ *for* $0 \leq i \leq k-1$ *is called a* linear trail.

We write $\mathrm{LT}(u, v)$ for the set of all linear trails in the linear approximation (u, v). These are the trails with $q^{(0)} = u$ and $q^{(k)} = v$. We call (u, v) the *enveloping linear approximation* of the trails in $\mathrm{LT}(u, v)$. If $|\mathrm{LT}(u, v)| \geq 2$, then we say that trails *cluster* together in the linear approximation (u, v).

By deleting the initial linear mask u and final linear mask v of a linear trail $(u, q^{(1)}, \ldots, q^{(k-1)}, v)$ we are left with a *linear trail core*. A linear trail core obtained in this way is said to be in the linear approximation (u, v). Note that a linear trail core actually defines a set of linear trails with the same inner linear masks.

Definition 22. *The* correlation contribution *of a linear trail Q over α equals*

$$\mathrm{C}_\alpha(Q) = \prod_{i=0}^{k-1} \mathrm{C}_{\mathrm{R}_i}(q^{(i)}, q^{(i+1)}) \,.$$

From the theory of correlation matrices [14], it follows that

$$\mathrm{C}_\alpha(u, v) = \sum_{Q \in \mathrm{LT}(u,v)} \mathrm{C}_\alpha(Q) \,.$$

Given any linear approximation (u, v) over a round function R, it is easy to compute its correlation. By specifying the intermediate linear masks we obtain a linear trail (u, b, c, v). Thanks to the linearity of L, we have $b = \mathrm{L}^\top(c)$ and due to the fact that a linear mask is invariant under addition of a constant, all valid

such linear trails are of the form $(u, \mathrm{L}^\top(v), v, v)$. Hence the linear approximation (u, v) contains only a single trail and its correlation contribution is the correlation of the linear approximation $(u, \mathrm{L}^\top(v))$ over the S-box layer, where the round constant addition affects the sign:

$$\mathrm{C_R}(u, v) = \chi_v(\iota(0)) \prod_{0 \le j < m} \mathrm{C_{S_j}}(\mathrm{Proj}_j(u), \mathrm{Proj}_j(\mathrm{L}^\top(v))) .$$

2.8.5 Linear potential and weight

Definition 23. *The* linear potential *(*LP*) of a linear approximation (u, v) is a real number and related to the correlation by*

$$\mathrm{LP}_\alpha(u, v) = \mathrm{C}_\alpha(u, v)\overline{\mathrm{C}_\alpha(u, v)} .$$

Analogous to the differential cryptanalysis case, we define a weight metric.

Definition 24. *If u and v are compatible over α, then we can define the* correlation weight *of the linear approximation (u, v) as*

$$\mathrm{w}_\alpha(u, v) := -\log_q(\mathrm{LP}_\alpha(u, v)) .$$

Definition 25. *The correlation weight of a linear trail $Q = (q^{(0)}, q^{(1)}, \ldots, q^{(k)})$ is defined as*

$$\mathrm{w_c}(Q) := \sum_{i=0}^{k-1} \mathrm{w_c}(q^{(i)}, q^{(i+1)}) .$$

3 Research question

The central research question of this thesis is:

> "Can we increase our understanding of how to design an efficient function family defined over an arbitrary finite field that is conjectured to be PRF-secure?"

The meaning of the word efficiency depends on the computational context and the relevant performance measure. In hardware implementations, typical measures include latency and throughput, whereas in the context of garbled circuits, multiplicative depth is a more appropriate measure.

This thesis focuses on function families built on cryptographic permutations, with an emphasis on studying the interaction between their components through the lenses of differential, linear, and integral cryptanalysis.

In chapter 4, we discuss how aligned designs have been favored due to their inherent structure, which facilitates combinatorial reasoning about trail bounds. However, it is precisely this structure that leads to various clustering effects. In contrast, unaligned designs avoid such clustering. However, the lack of structure in unaligned designs makes manual trail bound analysis impractical, necessitating the use of computer programs. Given these considerations, I prefer unaligned designs, as I believe structure to be the foundation of any kind of successful cryptanalysis.

In chapter 6, chapter 7, and chapter 8, the nonlinear layer is based on the multiplication, either between two field elements or as a squaring operation. This operation not only exhibits ideal differential and linear properties but also remains field-agnostic, making it a versatile building block for cryptographic designs.

Rather than analyzing the cryptographic permutation in isolation, we state a security claim for the overall primitive. This allows us to reduce the number of rounds in the permutation, improving efficiency without compromising the targeted notion of security. An example of this approach is given in chapter 6.

Finally, chapter 5 demonstrates that combining strong individual components does not automatically result in a secure construction. This observation underscores the importance of understanding how cryptographic building blocks interact, highlighting the need for careful integration of components to ensure security.

References

1. M. Artin. *Algebra*. Birkhäuser, 1998.
2. T. Baignères, J. Stern, and S. Vaudenay. "Linear Cryptanalysis of Non Binary Ciphers". In: *Selected Areas in Cryptography*. Ed. by C. Adams, A. Miri, and M. Wiener. Springer Berlin Heidelberg, Heidelberg, Germany, 2007, pp. 184–211. DOI: 10.1007/978-3-540-77360-3_13. URL: https://doi.org/10.1007/978-3-540-77360-3_13.
3. G. V. Bard. *Algebraic Cryptanalysis*. 1st. Springer Publishing Company, Incorporated, 2009. ISBN: 0387887563.
4. M. Bardet, J. Faugère, and B. Salvy. "On the complexity of the F5 Gröbner basis algorithm". *J. Symb. Comput.* 70, 2015, pp. 49–70.
5. M. Bellare, A. Desai, E. Jokipii, and P. Rogaway. "A Concrete Security Treatment of Symmetric Encryption". In: *38th Annual Symposium on Foundations of Computer Science, FOCS '97, Miami Beach, Florida, USA, October 19-22, 1997*. IEEE Computer Society, 1997, pp. 394–403. DOI: 10.1109/SFCS.1997.646128. URL: https://doi.org/10.1109/SFCS.1997.646128.
6. M. Bellare and P. Rogaway. "Random Oracles are Practical: A Paradigm for Designing Efficient Protocols". In: *CCS '93, Proceedings of the 1st ACM Conference on Computer and Communications Security, Fairfax, Virginia, USA, November 3-5, 1993*. Ed. by D. E. Denning, R. Pyle, R. Ganesan, R. S. Sandhu, and V. Ashby. ACM, 1993, pp. 62–73. DOI: 10.1145/168588.168596. URL: https://doi.org/10.1145/168588.168596.
7. D. J. Bernstein. "How to Stretch Random Functions: The Security of Protected Counter Sums". *J. Cryptol.* 12:3, 1999, pp. 185–192. DOI: 10.1007/S001459900051. URL: https://doi.org/10.1007/s001459900051.
8. G. Bertoni, J. Daemen, S. Hoffert, M. Peeters, G. Van Assche, and R. Van Keer. "Farfalle: parallel permutation-based cryptography". *IACR Trans. Symmetric Cryptol.* 2017:4, 2017, pp. 1–38.
9. K. Bhargavan and G. Leurent. "On the Practical (In-)Security of 64-bit Block Ciphers: Collision Attacks on HTTP over TLS and OpenVPN". In: *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, Vienna, Austria, October 24-28, 2016*. Ed. by E. R. Weippl, S. Katzenbeisser, C. Kruegel, A. C. Myers, and S. Halevi. ACM, 2016, pp. 456–467. DOI: 10.1145/2976749.2978423. URL: https://doi.org/10.1145/2976749.2978423.
10. E. Biham and A. Shamir. "Differential Cryptanalysis of DES-like Cryptosystems". In: *Advances in Cryptology - CRYPTO '90, 10th Annual International Cryptology Conference, Santa Barbara, California, USA, August 11-15, 1990, Proceedings*. Ed. by A. Menezes and S. A. Vanstone. Vol. 537. Lecture Notes in Computer Science. Springer, 1990, pp. 2–21. DOI: 10.1007/3-540-38424-3_1. URL: https://doi.org/10.1007/3-540-38424-3%5C_1.
11. D. Boneh and V. Shoup. *A Graduate Course in Applied Cryptography*. 2024. URL: https://crypto.stanford.edu/~dabo/cryptobook/draft_0_2.pdf.

12. S. Burris and H. P. Sankappanavar. *A Course in Universal Algebra*. Springer, New York, 1981.

13. D. A. Cox, J. Little, and D. O'Shea. *Ideals, Varieties, and Algorithms*. Fourth. Undergraduate Texts in Mathematics. Springer, 2015. ISBN: 978-3-319-16720-6.

14. J. Daemen. *Cipher and hash function design, strategies based on linear and differential cryptanalysis, PhD Thesis*. K.U.Leuven, 1995.

15. J. Daemen, R. Govaerts, and J. Vandewalle. "Correlation Matrices". In: *Fast Software Encryption: Second International Workshop. Leuven, Belgium, 14-16 December 1994, Proceedings*. Ed. by B. Preneel. Vol. 1008. Lecture Notes in Computer Science. Springer, 1994, pp. 275–285. DOI: 10.1007/3-540-60590-8_21. URL: https://doi.org/10.1007/3-540-60590-8%5C_21.

16. J. Daemen, S. Hoffert, G. Van Assche, and R. Van Keer. "The design of Xoodoo and Xoofff". *IACR Trans. Symmetric Cryptol.* 2018:4, 2018, pp. 1–38. DOI: 10.13154/tosc.v2018.i4.1-38.

17. J. Daemen, L. R. Knudsen, and V. Rijmen. "The Block Cipher Square". In: *FSE*. Vol. 1267. LNCS. 1997, pp. 149–165.

18. J. Daemen, B. Mennink, and G. V. Assche. "Full-State Keyed Duplex with Built-In Multi-user Support". In: *Advances in Cryptology - ASIACRYPT 2017 - 23rd International Conference on the Theory and Applications of Cryptology and Information Security, Hong Kong, China, December 3-7, 2017, Proceedings, Part II*. Ed. by T. Takagi and T. Peyrin. Vol. 10625. Lecture Notes in Computer Science. Springer, 2017, pp. 606–637.

19. J. Daemen and V. Rijmen. "Correlation Analysis in GF(2^n)". In: *The Design of Rijndael: The Advanced Encryption Standard (AES)*. Springer Berlin Heidelberg, Heidelberg, Germany, 2020, pp. 181–194. DOI: 10.1007/978-3-662-60769-5_12. URL: https://doi.org/10.1007/978-3-662-60769-5_12.

20. J. Daemen and V. Rijmen. "Probability distributions of correlation and differentials in block ciphers". *J. Math. Cryptol.* 1:3, 2007, pp. 221–242. DOI: 10.1515/JMC.2007.011. URL: https://doi.org/10.1515/JMC.2007.011.

21. *Definition of CRYPTOGRAPHY — merriam-webster.com*. https://www.merriam-webster.com/dictionary/cryptography. [Accessed 17-08-2024].

22. P. Derbez and B. Lambin. "Fast MILP Models for Division Property". *IACR Trans. Symmetric Cryptol.* 2022:2, 2022, pp. 289–321. DOI: 10.46586/TOSC.V2022.I2.289-321.

23. W. Diffie and M. E. Hellman. "New directions in cryptography". *IEEE Trans. Inf. Theory* 22:6, 1976, pp. 644–654. DOI: 10.1109/TIT.1976.1055638. URL: https://doi.org/10.1109/TIT.1976.1055638.

24. M. J. Dworkin. *SP 800-38A 2001 edition. Recommendation for Block Cipher Modes of Operation: Methods and Techniques*. Technical report. Gaithersburg, MD, USA, 2001.

25. S. Even and Y. Mansour. "A Construction of a Cipher from a Single Pseudorandom Permutation". *J. Cryptol.* 10:3, 1997, pp. 151–162. DOI: 10.1007/S001459900025. URL: https://doi.org/10.1007/s001459900025.

26. J.-C. Faugère. "A new efficient algorithm for computing Gröbner bases without reduction to zero F5". In: *ISSAC*. ACM, 2002, pp. 75–83.

27. J. Faugère, P. M. Gianni, D. Lazard, and T. Mora. "Efficient Computation of Zero-Dimensional Gröbner Bases by Change of Ordering". *J. Symb. Comput.* 16:4, 1993, pp. 329–344.

28. W. Feller. *An Introduction to Probability Theory and Its Applications*. Vol. 1. Wiley, 1968. ISBN: 0471257087. URL: http://www.amazon.ca/exec/obidos/redirect?tag=citeulike04-20%7B%5C&%7Dpath=ASIN/0471257087.

29. G. Genovese. "Improving the algorithms of Berlekamp and Niederreiter for factoring polynomials over finite fields". *J. Symb. Comput.* 42:1-2, 2007, pp. 159–177.

30. Y. Hao, G. Leander, W. Meier, Y. Todo, and Q. Wang. "Modeling for Three-Subset Division Property without Unknown Subset". *J. Cryptol.* 34:3, 2021, p. 22.

31. X.-d. Hou. *Lectures on Finite Fields*. American Mathematical Society, 2018.

32. A. Kerckhoffs. "La cryptographie militaire". *Journal des Sciences Militaires*, 1883, pp. 161–191.

33. L. R. Knudsen and D. A. Wagner. "Integral Cryptanalysis". In: *FSE*. Vol. 2365. LNCS. 2002, pp. 112–127.

34. D. Kuijsters. "Coding Theory: A Gröbner Basis Approach". MSc thesis. Eindhoven University of Technology, 2017.

35. X. Lai. "Higher Order Derivatives and Differential Cryptanalysis". In: *Communications and Cryptography: Two Sides of One Tapestry*. Springer US, 1994, pp. 227–233.

36. M. Luby and C. Rackoff. "How to Construct Pseudorandom Permutations from Pseudorandom Functions". *SIAM J. Comput.* 17:2, 1988, pp. 373–386. DOI: 10.1137/0217022. URL: https://doi.org/10.1137/0217022.

37. M. Matsui. "Linear Cryptanalysis Method for DES Cipher". In: *Advances in Cryptology - EUROCRYPT '93, Proceedings*. Ed. by T. Helleseth. DOI: 10.1007/3-540-48285-7_33.

38. D. Micciancio and M. Walter. "On the Bit Security of Cryptographic Primitives". In: *Advances in Cryptology - EUROCRYPT 2018 - 37th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Tel Aviv, Israel, April 29 - May 3, 2018 Proceedings, Part I*. Ed. by J. B. Nielsen and V. Rijmen. Vol. 10820. Lecture Notes in Computer Science. Springer, 2018, pp. 3–28. DOI: 10.1007/978-3-319-78381-9_1. URL: https://doi.org/10.1007/978-3-319-78381-9%5C_1.

39. K. H. Rosen. *Discrete Mathematics and Its Applications: And Its Applications*. McGraw-Hill Higher Education, 2006. ISBN: 0072880082.

40. R. P. Stanley. *Enumerative Combinatorics*. Vol. 1. Cambridge Studies in Advanced Mathematics 49. Cambridge University Press, Cambridge, NY, 2012. ISBN: 9781107015425 1107015421 9781107602625 1107602629.

41. Y. Todo and M. Morii. "Bit-Based Division Property and Application to Simon Family". *IACR Cryptol. ePrint Arch.*, 2016, p. 285.

42. G. S. Vernam. "Cipher Printing Telegraph Systems for Secret Wire and Radio Telegraphic Communications". *Journal American Institute of Electrical Engineers* XLV:??, 1926, pp. 109–115.

43. S. Wang, B. Hu, J. Guan, K. Zhang, and T. Shi. "MILP-aided Method of Searching Division Property Using Three Subsets and Applications". In: *Advances in Cryptology - ASIACRYPT 2019 - 25th International Conference on the Theory and Application of Cryptology and Information Security, Kobe, Japan, December 8-12, 2019, Proceedings, Part III*. Ed. by S. D. Galbraith and S. Moriai. Vol. 11923. Lecture Notes in Computer Science. Springer, 2019, pp. 398–427.

Part II

Research chapters

This part contains the scientific contributions of the author of this thesis. The publications are included in a format that closely resembles their original presentation in the respective journals or conference proceedings. Only minor editorial changes, such as formatting adjustments, have been made. Substantive content has not been altered; any errors present in the original publications have been retained to reflect the state of the work at the time of publication.

4 Thinking Outside the Superbox

Nicolas Bordes[1], Joan Daemen[2], Daniël Kuijsters[2], Gilles Van Assche[3]

1 – Université Grenoble Alpes, France
2 – Radboud University, The Netherlands
3 – STMicroelectronics, Belgium

My contributions. This chapter is based on work that was accepted at Crypto 2021. I contributed significantly to its content, including the software for generating data and histograms, the attempt to formalize the notion of alignment, and the writing of the text. The exceptions are subsection 4.6.4 and section 4.7. Some shortcomings in our formalization have been addressed by Lambin et al. in [27].

Abstract. The design of a block cipher or cryptographic permutation can be approached in many different ways. One such approach, popularized by AES, consists in grouping the bits along the S-box boundaries, e.g., in bytes, and in consistently processing them in these groups. This aligned approach leads to hierarchical structures like superboxes that make it possible to reason about the differential and linear propagation properties using combinatorial arguments. In contrast, an unaligned approach avoids any such grouping in the design of transformations. However, without hierarchical structure, sophisticated computer programs are required to investigate the differential and linear propagation properties of the primitive. In this paper, we formalize this notion of alignment and study four primitives that are exponents of different design strategies. We propose a way to analyze the interactions between the linear and the nonlinear layers w.r.t. the differential and linear propagation, and we use it to systematically compare the four primitives using non-trivial computer experiments. We show that alignment naturally leads to different forms of clustering, e.g., of active bits in boxes, of two-round trails in activity patterns, and of trails in differentials and linear approximations.

4.1 Introduction

Modern block ciphers and cryptographic permutations consist of the iteration of a round function. In many cases this round function consists of a layer of nonlinear S-boxes, a mixing layer, a shuffle layer (AKA a bit transposition or bit permutation), and the addition of a round key (in block ciphers) or constant (in cryptographic permutations).

Many papers investigate S-boxes and try to find a good compromise between implementation cost and propagation properties or provide a classification of all invertible S-boxes of a given width, see, e.g., [28,

35]. Similarly, there is a rich literature on certain types of mixing layers. In particular, there have been many papers written about finding maximum-distance separable (MDS) mappings or near-MDS mappings with minimum implementation cost according to some metric, see, e.g., [29, 38]. Building a good cipher starts with taking a good S-box and mixing layer and the rich cryptographic literature on these components provides us with ample choice. However, how these building blocks are combined in a round function and the resulting propagation properties has received much less systematic attention.

A standard way for designing a good round function from an S-box and an MDS mapping is the one followed in the Advanced Encryption Standard (AES) [33] and is known as the *wide trail strategy* [14, 21]. This strategy gives criteria for the shuffle layer and comes with easy-to-verify bounds for the differential probability (DP) of differential trails (also known as characteristics) and the linear potential (LP) of linear trails. These bounds and its simplicity have made it one of the most applied design strategies, and AES has inspired a plethora of primitive designs, including lightweight ones. By adopting 4-bit S-boxes instead of 8-bit ones and modern lightweight MDS layers in a smart structure, multiple lightweight ciphers have been constructed. Many lessons were learned and this line of design has culminated in the block cipher of the NIST lightweight competition candidate Saturnin [13], a truly modern version of AES.

Naturally, there are alternative design approaches. A popular design approach is the one underlying the 64-bit lightweight block cipher Present [10]. Its round function has no MDS layer and simply consists of an S-box layer, a bit shuffle, and a key addition. It gets its diffusion from the combination of a smart choice of the bit shuffle and specific propagation criteria from its well-chosen S-box and doing many rounds. The Present line of design has also been refined in the form of the Gift (64- and 128-bit) block ciphers [2] and the cryptographic permutations of the Spongent lightweight hash function [9] that is standardized in [1].

Another distinctive design approach is that of the cryptographic permutation of the SHA-3 standard [34], Keccak-f. Unlike Present, its round function does have a mixing layer, and it actually has all ingredients that AES has. Specifically, in their rationale, the designers also refer to the wide trail design strategy [7]. However, this wide-trail flavor does not appear to come with the simple bounds as in the case of AES, and designers have to resort to tedious and time-consuming programming efforts to obtain similar bounds. This is related to the fact that AES operates on *bytes* and Keccak-f on *bits*. The Keccak-f designers have discussed the difference between these two design approaches in [18]. In that paper, they have coined the term *alignment* to characterize this difference and supported it with some propagation experiments on Keccak-f. The Keccak-f line of design has also been refined and led to the 384-bit permutation that is used in Xoodyak [15], namely Xoodoo [16], a truly modern version of Keccak-f.

This treatment is not exhaustive and other distinctive design strategies exist. Some of them do not even use S-boxes or mixing layers, but they are based on alternating Additions with Rotations and XOR (ARX) such as Salsa [5], or they iterate very simple round functions many times such as Simon [3].

In this paper we systematically analyze the impact of alignment on the differential and linear propagation properties of ciphers. We show that certain design choices regarding how the S-box and mixing

layers are combined have a profound impact on the propagation properties. We identify and name a number of effects that are relevant in this context. Furthermore, we believe that this makes it possible to give a meaningful and non-ambiguous definition of the term alignment.

To illustrate this, we study the four primitives Rijndael-256 [20], Saturnin, Spongent-384, and Xoodoo. They have comparable width and all have a nonlinear layer consisting of equally-sized S-boxes that have the lowest known maximum DP and LP for their dimensions, see Section 4.2. They represent the three different design strategies, where we include both Rijndael-256 and Saturnin to illustrate the progress made in the last twenty years. We investigate their difference propagation and correlation properties, where for multiple rounds we adopt a *fixed-key* perspective. This, combined with the choice of relatively wide primitives, is geared towards their usage in permutation-based cryptography, but most findings are also relevant for the key-alternating block cipher case.

4.1.1 Outline and contributions

After discussing notation and conventions, we review the notions of differential and linear cryptanalysis in Section 4.2. In Section 4.3 we show how the nonlinear layer defines a so-called *box partition*, and we present a non-ambiguous definition of alignment. In Section 4.4 we present our four ciphers from the perspective of alignment and compare the costs of their round functions. Surprisingly, Spongent, despite being specified at bit level like Keccak-f, turns out to be aligned.

In Section 4.5 we recall the notions of bit and box weight as a measure of the mixing power of a linear layer. We report on this mixing power by means of *histograms* of states by their weight before and after the linear layer, rather than the usual *branch number* criterion. For all ciphers we observe a decay in mixing power from bit to box weight and describe and name the effect that causes this: *huddling*. This effect is more pronounced in aligned ciphers. This translates directly to the two-round differential and linear trail weight distributions, and we list them for all four ciphers. For the two most competitive proposals, we include histograms for three-round trails and a comparison for four rounds. Remarkably, despite the fact that Saturnin has a more expensive S-box layer and a mixing layer with better bit-level mixing power, Xoodoo has better differential and linear trail histograms for more than two rounds.

In Section 4.6, we show that trails that cluster necessarily share the same activity pattern, and we introduce the *cluster histogram* as a quantitative tool for the relation between the linear layer and the clustering of two-round trails in ciphers. We see that there is more clustering in the aligned than in the unaligned ciphers. We present the cluster histogram of the four primitives and, for three of them, we also analyze their two-round trail weight histograms. We conclude with a discussion on the clustering of trails in two and three rounds, and show that, at least up to weight 50, differentials over three rounds of Xoodoo admit only one trail, hence they do not cluster.

Finally, in Section 4.7 we study the independence of round differentials in trails. We show that, again at least up to weight 50, three-round differentials of Xoodoo are independent.

The generation of our histograms was non-trivial and the computation methods could be considered a contribution in themselves. See Section A after the paper. Software is available at `https://github.com/ongetekend/ThinkingOutsideTheSuperbox` under the CC0 license (public domain).

4.1.2 Notation and conventions

In this paper, we use the following conventions and notation. We write $\mathbb{Z}_{\geq 0}$ for the nonnegative integers and $\mathbb{Z}_{>0}$ for the positive integers. We write k with $k \in \mathbb{Z}_{\geq 0}$ for nonnegative integer variables. In other words, k is used as a placeholder for any nonnegative integer value.

Whenever we use indices, they always begin at 0. We define $[0,\, k-1] = \{i \in \mathbb{Z}_{\geq 0} : 0 \leq i \leq k-1\}$. Given a set S and an equivalence relation $\sim$ on S, we write $[a]_\sim$ for the equivalence class of $a \in S$. We denote the cardinality of S by $|S|$.

We study permutations $f\colon \mathbb{F}_2^b \to \mathbb{F}_2^b$. Any block cipher is transformed into a permutation by fixing the key, e.g., we fix all of its bits to 0.

We use the term *state* for a vector of b bits. It is either a vector that the permutation is applied to, a difference, or a linear mask (See Section 4.2). Given a state $a \in \mathbb{F}_2^b$, we refer to its ith component as a_i. In this paper, we consider index sets $\mathrm{B}_i \subseteq [0,\, b-1]$ that form an *ordered* partition. We write $P_i(a)\colon \mathbb{F}_2^b \to \mathbb{F}_2^{|\mathrm{B}_i|}$ for the *projection* onto the bits of a indexed by B_i.

We write e_i^k for the ith standard basis vector in $\mathbb{F}_2^k$, i.e., for $j \in [0,\, k-1]$ we have that $e_{ij}^k = 1$ if $i = j$ and 0 otherwise. We write + for vector addition in $\mathbb{F}_2^k$.

Permutations are typically built by composing a number of lightweight *round functions*, i.e., $f = \mathrm{R}_{r-1} \circ \cdots \circ \mathrm{R}_1 \circ \mathrm{R}_0$ for some $r \in \mathbb{Z}_{>0}$. We write $f[r] = \mathrm{R}_{r-1} \circ \cdots \circ \mathrm{R}_0$ and define $f[0] = \mathrm{id}$ with id the identity function. A round function is composed of *step functions*, i.e., $\mathrm{R}_i = \iota_i \circ \mathrm{L}_i \circ \mathrm{N}_i$, where N_i is a nonlinear map, L_i is a linear map, and ι_i is addition of a round constant. Apart from the round constant addition, these round functions are often, but not always, identical. For this reason, we will often simply write N or L, without reference to an index if the context allows for this, and we call N the nonlinear layer of f and L the linear layer of f. We write n for the number of S-boxes of N and denote their size by m. In this context, we suppose that $\mathrm{B}_j = \{jm, \ldots, (j+1)m - 1\}$.

Permutations of the index space are written as $\tau\colon [0,\, b-1] \to [0,\, b-1]$. By *shuffle (layer)*, we mean a linear transformation $\pi\colon \mathbb{F}_2^b \to \mathbb{F}_2^b$ given by $\pi(a) = \mathrm{P}_\tau\, a$, where P_τ is the permutation matrix associated with some τ, i.e., obtained by permuting the columns of the $(b \times b)$ identity matrix according to τ.

Given a linear transformation $\mathrm{L}\colon \mathbb{F}_2^b \to \mathbb{F}_2^b$, there exists a matrix $\mathrm{M} \in \mathbb{F}_2^{b \times b}$ such that $\mathrm{L}(a) = \mathrm{M}\, a$. We define its transpose $\mathrm{L}^\top\colon \mathbb{F}_2^b \to \mathbb{F}_2^b$ by $\mathrm{L}^\top(a) = \mathrm{M}^\top\, a$ and we denote the inverse of $\mathrm{L}^\top$, when it exists, by $\mathrm{L}^{-\top}$.

4.2 Differential and linear cryptanalysis

A major motivation behind the tools developed in this paper is better understanding of the interplay between the linear and nonlinear layer in relation to differential and linear cryptanalysis. We want to be

able to use the associated language freely when discussing these tools. Therefore, in this section, we go over the basic notions to make sure they are on hand when needed.

4.2.1 Differential cryptanalysis

Differential cryptanalysis [8] is a chosen-plaintext attack that exploits the non-uniformity of the distribution of differences at the output of a permutation when it is applied to pairs of inputs with a fixed difference. We call an ordered pair of an input and output difference $(\Delta_{\text{in}}, \Delta_{\text{out}}) \in (\mathbb{F}_2^b)^2$ a differential.

Definition 26. *Let $f\colon \mathbb{F}_2^b \to \mathbb{F}_2^b$ be a permutation and define $U_f(\Delta_{in}, \Delta_{out}) = \{x \in \mathbb{F}_2^b : f(x) + f(x + \Delta_{in}) = \Delta_{out}\}$. We call $U_f(\Delta_{in}, \Delta_{out})$ the solution set of the differential $(\Delta_{in}, \Delta_{out})$.*

Definition 27. *The* differential probability *(*DP*) of a differential $(\Delta_{in}, \Delta_{out})$ over the permutation $f\colon \mathbb{F}_2^b \to \mathbb{F}_2^b$ is defined as $\mathrm{DP}_f(\Delta_{in}, \Delta_{out}) = \frac{|U_f(\Delta_{in}, \Delta_{out})|}{2^b}$.*

If there exists an ordered pair $(x, x+\Delta_{\text{in}})$ with $x \in U_f(\Delta_{\text{in}}, \Delta_{\text{out}})$, then it is said to follow the differential $(\Delta_{\text{in}}, \Delta_{\text{out}})$. In this case, we say that the input difference Δ_{in} is *compatible* with the output difference Δ_{out} through f and call $(\Delta_{\text{in}}, \Delta_{\text{out}})$ a *valid* differential.

Definition 28. *A sequence $Q = (q^{(0)}, q^{(1)}, \dots, q^{(k)}) \in \left(\mathbb{F}_2^b\right)^{k+1}$ that satisfies $\mathrm{DP}_{\mathrm{R}_i}(q^{(i)}, q^{(i+1)}) > 0$ for $0 \le i \le k-1$ is called a k-round* differential trail*.*

Sometimes we specify a trail as $Q = (b_{-1}, a_0, b_0, \dots, a_k, b_k)$ by giving the intermediate differences between N_i and L_i as well, where $b_i = \mathrm{L}_i(a_i) = q_{i+1}$. We write $\mathrm{DT}(\Delta_{\text{in}}, \Delta_{\text{out}})$ for the set of all differential trails in the differential $(\Delta_{\text{in}}, \Delta_{\text{out}})$, so with $q^{(0)} = \Delta_{\text{in}}$ and $q^{(k)} = \Delta_{\text{out}}$. We call $(\Delta_{\text{in}}, \Delta_{\text{out}})$ the *enveloping differential* of the trails in $\mathrm{DT}(\Delta_{\text{in}}, \Delta_{\text{out}})$. If $|\mathrm{DT}(\Delta_{\text{in}}, \Delta_{\text{out}})| > 1$, then we say that trails *cluster* together in the differential $(\Delta_{\text{in}}, \Delta_{\text{out}})$.

By deleting the initial difference Δ_{in} and final difference Δ_{out} of a differential trail

$$(\Delta_{\text{in}}, q^{(1)}, \dots, q^{(k-1)}, \Delta_{\text{out}})$$

we are left with a *differential trail core*. A differential trail core obtained in this way is said to be in the differential $(\Delta_{\text{in}}, \Delta_{\text{out}})$. Note that a differential trail core actually defines a set of differential trails with the same inner differences.

We now define the DP of a differential trail. Each round differential $(q^{(i)}, q^{(i+1)})$ has a solution set $U_{\mathrm{R}_i}(q^{(i)}, q^{(i+1)})$. Consider the transformed set of points $U_i = f[i]^{-1}(U_{\mathrm{R}_i}(q^{(i)}, q^{(i+1)}))$ at the input of f. For an ordered pair $(x, x + q^{(0)})$ to follow the differential trail, it is required that $x \in U_f(Q) = \bigcap_{i=0}^{k-1} U_i$. The fraction of states x that satisfy this equation is the DP of the trail.

Definition 29. *The* DP *of a differential trail is defined as $\mathrm{DP}_f(Q) = \frac{|U_f(Q)|}{2^b}$.*

Definition 30. *The round differentials are said to be* independent *if*

$$\mathrm{DP}_f(Q) = \prod_{i=0}^{k-1} \mathrm{DP}_{\mathrm{R}_i}(q^{(i)}, q^{(i+1)}) .$$

Any given ordered pair $(x, x+\Delta_{\text{in}})$ follows exactly one differential trail. Hence, the DP of the differential $(\Delta_{\text{in}}, \Delta_{\text{out}})$ is the sum of the DPs of all differential trails with initial difference Δ_{in} and final difference Δ_{out}.

$$\mathrm{DP}_f(\Delta_{\text{in}}, \Delta_{\text{out}}) = \sum_{Q \in \mathrm{DT}(\Delta_{\text{in}}, \Delta_{\text{out}})} \mathrm{DP}_f(Q)\,.$$

Given any differential $(\Delta_{\text{in}}, \Delta_{\text{out}})$ over a round function R, it is easy to compute its DP value. By specifying the intermediate differences we obtain a differential trail $(\Delta_{\text{in}}, b, c, \Delta_{\text{out}})$. Thanks to the linearity of L, we have $c = \mathrm{L}(b)$ and due to the fact that a difference is invariant under addition of a constant, all valid such differential trails are of the form $(\Delta_{\text{in}}, \mathrm{L}^{-1}(\Delta_{\text{out}}), \Delta_{\text{out}}, \Delta_{\text{out}})$. Therefore, the differential $(\Delta_{\text{in}}, \Delta_{\text{out}})$ contains only a single trail and its DP is the DP of the differential $(\Delta_{\text{in}}, \mathrm{L}^{-1}(\Delta_{\text{out}}))$ over the S-box layer:

$$\mathrm{DP}_{\mathrm{R}}(\Delta_{\text{in}}, \Delta_{\text{out}}) = \prod_{0 \le j < n} \mathrm{DP}_{\mathrm{S}_j}(P_j(\Delta_{\text{in}}), P_j(\mathrm{L}^{-1}(\Delta_{\text{out}})))\,.$$

Hence, the DP of a round differential is the product of the DP values of its S-box differentials.

Definition 31. *The* restriction weight *of a differential* $(\Delta_{in}, \Delta_{out})$ *that satisfies* $\mathrm{DP}_f(\Delta_{in}, \Delta_{out}) > 0$ *is defined as* $\mathrm{w_r}(\Delta_{in}, \Delta_{out}) = -\log_2 \mathrm{DP}_f(\Delta_{in}, \Delta_{out})$.

For a differential trail, we sum the weights of the round differentials.

Definition 32. *The restriction weight of a differential trail* $Q = (q^{(0)}, q^{(1)}, \ldots, q^{(k)})$ *is defined as*

$$\mathrm{w_r}(Q) = \sum_{i=0}^{k-1} \mathrm{w_r}(q^{(i)}, q^{(i+1)})\,.$$

If the round differentials are independent in the sense of Definition 30, then we have that $\mathrm{DP}_f(Q) = 2^{-\mathrm{w_r}(Q)}$.

4.2.2 Linear cryptanalysis

Linear cryptanalysis [30] is a known-plaintext attack. It exploits large correlations (in absolute value) between linear combinations of input bits and linear combinations of output bits of a permutation.

Definition 33. *The (signed)* correlation *between the linear mask* $u \in \mathbb{F}_2^b$ *at the input and the linear mask* $v \in \mathbb{F}_2^b$ *at the output of a function* $f\colon \mathbb{F}_2^b \to \mathbb{F}_2^b$ *is defined as*

$$\mathrm{C}_f(u, v) = \frac{1}{2^b} \sum_{x \in \mathbb{F}_2^b} (-1)^{u^\top x + v^\top f(x)}\,.$$

If $\mathrm{C}_f(u, v) \neq 0$, then we say that u is compatible with v. We call the ordered pair of linear masks (u, v) a *linear approximation*. We note that in the literature (e.g., in the linear cryptanalysis attack by Matsui [30]) the term linear approximation has several meanings. It should not be confused with what we call a linear trail.

Definition 34. *A sequence* $Q = (q^{(0)}, q^{(1)}, \ldots, q^{(k)}) \in \left(\mathbb{F}_2^b\right)^{k+1}$ *that satisfies* $\mathrm{C}_{\mathrm{R}_i}(q^{(i)}, q^{(i+1)}) \neq 0$ *for* $0 \leq i \leq k-1$ *is called a* linear trail.

We write $\mathrm{LT}(u, v)$ for the set of all linear trails in the linear approximation (u, v), so with $q^{(0)} = u$ and $q^{(k)} = v$. We call (u, v) the *enveloping linear approximation* of the trails in $\mathrm{LT}(u, v)$. If $|\mathrm{LT}(u, v)| > 1$, then we say that trails *cluster* together in the linear approximation (u, v).

By deleting the initial linear mask u and final linear mask v of a linear trail $(u, q^{(1)}, \ldots, q^{(k-1)}, v)$ we are left with a *linear trail core*. A linear trail core obtained in this way is said to be in the linear approximation (u, v). Note that a linear trail core actually defines a set of linear trails with the same inner linear masks.

Definition 35. *The* correlation contribution *of a linear trail* Q *over* f *equals*

$$\mathrm{C}_f(Q) = \prod_{i=0}^{k-1} \mathrm{C}_{\mathrm{R}_i}(q^{(i)}, q^{(i+1)}) .$$

From the theory of correlation matrices [14], it follows that

$$\mathrm{C}_f(u, v) = \sum_{Q \in \mathrm{LT}(u,v)} \mathrm{C}_f(Q) .$$

Given any linear approximation (u, v) over a round function R, it is easy to compute its correlation. By specifying the intermediate linear masks we obtain a linear trail (u, b, c, v). Thanks to the linearity of L, we have $b = \mathrm{L}^\top(c)$ and due to the fact that a linear mask is invariant under addition of a constant, all valid such linear trails are of the form $(u, \mathrm{L}^\top(v), v, v)$. Hence the linear approximation (u, v) contains only a single trail and its correlation contribution is the correlation of the linear approximation $(u, \mathrm{L}^\top(v))$ over the S-box layer, where the round constant addition affects the sign:

$$\mathrm{C}_{\mathrm{R}}(u, v) = (-1)^{v^\top \iota(0)} \prod_{0 \leq j < n} \mathrm{C}_{\mathrm{S}_j}(P_j(u), P_j(\mathrm{L}^\top(v))) .$$

Definition 36. *The* linear potential *(*LP*) of a linear approximation* (u, v) *is defined as* $\mathrm{LP}_f(u, v) = \mathrm{C}_f(u, v)^2$.

Analogous to the differential cryptanalysis case, we define a weight metric.

Definition 37. *The* correlation weight *of a linear approximation* (u, v) *with* $\mathrm{LP}_f(u, v) \neq 0$ *is given by* $\mathrm{w_c}(u, v) = -\log_2 \mathrm{LP}_f(u, v)$.

Definition 38. *The correlation weight of a linear trail* $Q = (q^{(0)}, q^{(1)}, \ldots, q^{(k)})$ *is defined as*

$$\mathrm{w_c}(Q) = \sum_{i=0}^{k-1} \mathrm{w_c}(q^{(i)}, q^{(i+1)}) .$$

4.3 Box partitioning and alignment

In this section, we consider the partition of the index space defined by the nonlinear layer N. The *alignment* properties of the other step functions with respect to this partition have an important impact on the propagation properties of the round function.

The nonlinear layer N consists of the parallel application of n S-boxes of size m to disjoint parts of the state, indexed by B_i. Formally, this means that we can write N as $\mathrm{S}_0 \times \cdots \times \mathrm{S}_{n-1}$ and that it is characterized by

$$P_i \circ (\mathrm{S}_0 \times \cdots \times \mathrm{S}_{n-1}) = \mathrm{S}_i \circ P_i \text{ for } 0 \leq i \leq n-1\,.$$

Hence, N defines a unique *ordered* partition $\Pi_\mathrm{N} = (\mathrm{B}_0, \ldots, \mathrm{B}_{n-1})$ of the index space $[0,\ b-1]$. We call Π_N the *box partition* defined by N and the B_i N-boxes. If there is no ambiguity, we call the box partition Π and its members boxes.

Besides the box partition, it is clearly possible to define other partitions of the index space as well. We call a partition *non-trivial* if it has at least two members. Between any two partitions of the index space there may be a relation that we denote as *refinement*.

Definition 39. *We call* Π *a* refinement *of* Π' *and write* $\Pi \leq \Pi'$ *if for every* $(i, \mathrm{B}_i) \in \Pi$ *there exists a* $(j, \mathrm{B}'_j) \in \Pi'$ *such that* $\mathrm{B}_i \subseteq \mathrm{B}'_j$.

Let Π be a partition of the index space consisting of k boxes, each of size l. We call a shuffle layer a Π-shuffle if the associated permutation matrix can be partitioned into k identity matrices of dimension $(l \times l)$. If this is the case, then bit index permutation can be specified as a box index permutation.

Definition 40. *We call* $\phi\colon \mathbb{F}_2^b \to \mathbb{F}_2^b$ aligned *to* Π *if we can decompose it as*

$$\phi_0 \times \cdots \times \phi_{k-1}\colon \bigtimes_{i=0}^{k-1} \mathbb{F}_2^l \to \bigtimes_{i=0}^{k-1} \mathbb{F}_2^l\,,$$

In this case, we call the ϕ_i *box functions*.

Definition 41. *Given a round function that is composed of the parallel application* N *of equally-sized S-boxes, a linear layer* L*, and the addition* ι *of a round constant, we say it is* aligned *if it is possible to decompose the linear layer* L *as* $\mathrm{L} = \pi \circ \mathrm{M}$ *in such a way that*

- π *is a* Π_N*-shuffle;*
- M *is aligned to a non-trivial partition* Π_M *that satisfies* $\Pi_\mathrm{N} \leq \Pi_\mathrm{M}$.

We assume that the split between the linear and nonlinear layer is chosen so as to maximize the number of S-boxes in N.

Note that ι does not play a role in the alignment properties. If all of the round functions of a primitive are aligned, then we call the primitive aligned. If the primitive is *not* aligned, then we call it *unaligned*.

Any aligned primitive has a *superbox* structure [36], that is helpful when investigating distributions and bounds on the DP of two-round differentials and the LP of two-round trails. We explain what this means. Consider a two-round structure

$$\pi \circ \mathrm{M} \circ \mathrm{N} \circ \pi \circ \mathrm{M} \circ \mathrm{N} \ .$$

The final two linear steps π and M have no effect on the distributions, so we can simplify this expression to $\mathrm{N} \circ \pi \circ \mathrm{M} \circ \mathrm{N}$. Clearly, $\mathrm{N} \circ \pi = \pi \circ \mathrm{N}'$, with $\mathrm{N}' := \pi^{-1} \circ \mathrm{N} \circ \pi$. Hence, this is equivalent to $\pi \circ \mathrm{N}' \circ \mathrm{M} \circ \mathrm{N}$. Discarding the shuffle layer at the end gives $\mathrm{N}' \circ \mathrm{M} \circ \mathrm{N}$. Since $\Pi_{\mathrm{N}'} = \Pi_{\mathrm{N}} \leq \Pi_{\mathrm{M}}$, we can view this as the parallel application of a number of superboxes. We call this a *superbox layer*. In a sequence of two rounds, $\mathrm{N}' \circ \mathrm{M} \circ \mathrm{N}$ is a (composite) nonlinear layer and $\pi \circ \mathrm{M} \circ \pi$ is a (composite) linear layer. If the latter is aligned to a non-trivial partition Π such that $\Pi_{\mathrm{M}} \leq \Pi$, then we call this two-round structure aligned to Π_{M}.

4.4 The ciphers we investigate

In this section we describe the round functions of the ciphers we investigate in this paper, their alignment properties, and compare their implementation cost.

4.4.1 Rijndael

Rijndael [20] is a block cipher family supporting all block and key lengths of $b = 32k$ bits, with $4 \leq k \leq 8$, i.e., ranging from 128 up to and including 256 bits. The case $b = 128$ is of great importance as Rijndael with that block length is the ubiquitous AES [33]. In this paper we investigate Rijndael-256, the instance with $b = 256$, a width closer to those of the other ciphers we investigate. In the remainder of this paper we will write Rijndael for Rijndael-256.

The Rijndael round function consists of four steps: a nonlinear layer SubBytes, a box shuffle ShiftRows, a mixing layer MixColumns, and round key addition AddRoundKey. As its name suggest, $\Pi_{\mathtt{SubBytes}}$ partitions the state in bytes and ShiftRows is a $\Pi_{\mathtt{SubBytes}}$-shuffle. The mixing layer, MixColumns, is aligned to a non-trivial partition $\Pi_{\mathtt{MixColumns}}$ that corresponds to the 8 *columns*, each containing 4 bytes, and we have $\Pi_{\mathtt{SubBytes}} \leq \Pi_{\mathtt{MixColumns}}$. It follows that Rijndael is aligned. Figure 4.1 shows Rijndael-128 that is is easier to draw due to its dimensions, but the alignment properties for Rijndael-256 are the same.

4.4.2 Saturnin

The Saturnin [13] block cipher has a 256-bit key and block length. The state has several representations: three-dimensional, two-dimensional, and flat. In three dimensions, the 256-bit state is represented as a $4 \times 4 \times 4$ cube of 4-bit *nibbles*. Nibbles in the cube are indexed by triples (x, y, z). A *slice* is a subset of the nibbles with z constant. A *sheet* is a subset of the nibbles with x constant. A *column* is a subset of the nibbles with x and z constant.

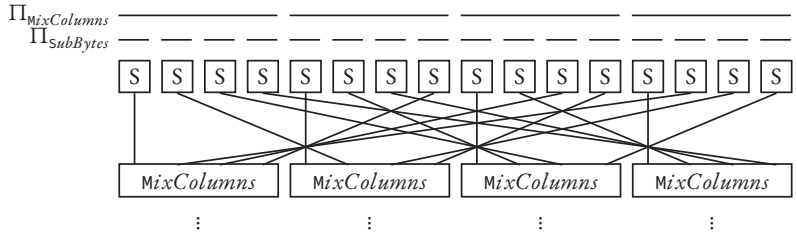


Figure 4.1: Alignment properties of Rijndael.

The Saturnin permutation is composed of a number of so-called super-rounds and a super-round consists of two consecutive rounds with indices $2r$ and $2r + 1$. Round $2r$ is composed as $\mathrm{MC} \circ \mathrm{S}$, where MC is a mixing layer and S is a nonlinear layer. There are two different rounds with odd indices. Round $4r + 1$ is composed as follows: $\mathrm{RC} \circ \mathrm{RK} \circ \mathrm{SR}_{\mathrm{slice}}^{-1} \circ \mathrm{MC} \circ \mathrm{SR}_{\mathrm{slice}} \circ \mathrm{S}$. Round $4r + 3$ consists of $\mathrm{RC} \circ \mathrm{RK} \circ \mathrm{SR}_{\mathrm{sheet}}^{-1} \circ \mathrm{MC} \circ \mathrm{SR}_{\mathrm{sheet}} \circ \mathrm{S}$. Here, RC denotes addition of a round constant, RK denotes addition of a round key, and $\mathrm{SR}_{\mathrm{slice}}$ and $\mathrm{SR}_{\mathrm{sheet}}$ shuffle nibbles. The partition Π_{S} divides the state into 64 nibbles. The shuffles $\mathrm{SR}_{\mathrm{slice}}$ and $\mathrm{SR}_{\mathrm{sheet}}$ are Π_{S}-shuffles. The mixing layer MC is aligned to a non-trivial partition Π_{MC} that divides the state into 16 columns, each consisting of 4 nibbles, and that satisfies $\Pi_{\mathrm{S}} \leq \Pi_{\mathrm{MC}}$. It follows that Saturnin is aligned. In a super-round we identify the sequence $\mathrm{S} \circ \mathrm{MC} \circ \mathrm{S}$ as a superbox layer with partition Π_{MC} and the linear layer of such a round is $\mathrm{SR}_{\mathrm{slice}}^{-1} \circ \mathrm{MC} \circ \mathrm{SR}_{\mathrm{slice}}$. This is a mixing layer that is aligned to a non-trivial partition Π_{slice} that divides the state into 4 slices, each containing 4 columns, and we have $\Pi_{\mathrm{MC}} \leq \Pi_{\mathrm{slice}}$. Similarly, for the other type of super-round, the mixing layer is aligned to a non-trivial partition Π_{sheet} that divides the state into 4 sheets, and we have $\Pi_{\mathrm{MC}} \leq \Pi_{\mathrm{sheet}}$. It follows that the super-rounds of Saturnin are aligned and hence have their own superboxes. These have width 64 bits and we call them *hyperboxes*. Figure 4.2 shows the alignment properties of the steps.

4.4.3 Spongent

Spongent [9] is a sponge-based hash function family that uses a Present-like permutation. The permutation is defined for any b that is a multiple of 4. In this paper, we only consider the case $b = 384$, to match the state size of the largest of the other permutations that we investigate, Xoodoo. The round function of Spongent consists of three steps: a round constant addition ιCounter, a 4-bit S-box layer sBoxLayer, and a bit shuffle pLayer.

The index permutation of the bit shuffle pLayer is:

$$\mathrm{pLayer}(j) = \begin{cases} 96j \bmod 383, & \text{if } j \in [0, 382] \\ 383, & \text{if } j = 383 \end{cases}$$

As indicated by the Spongent designers in [9], we can decompose it into a mixing layer, followed by a box shuffle:

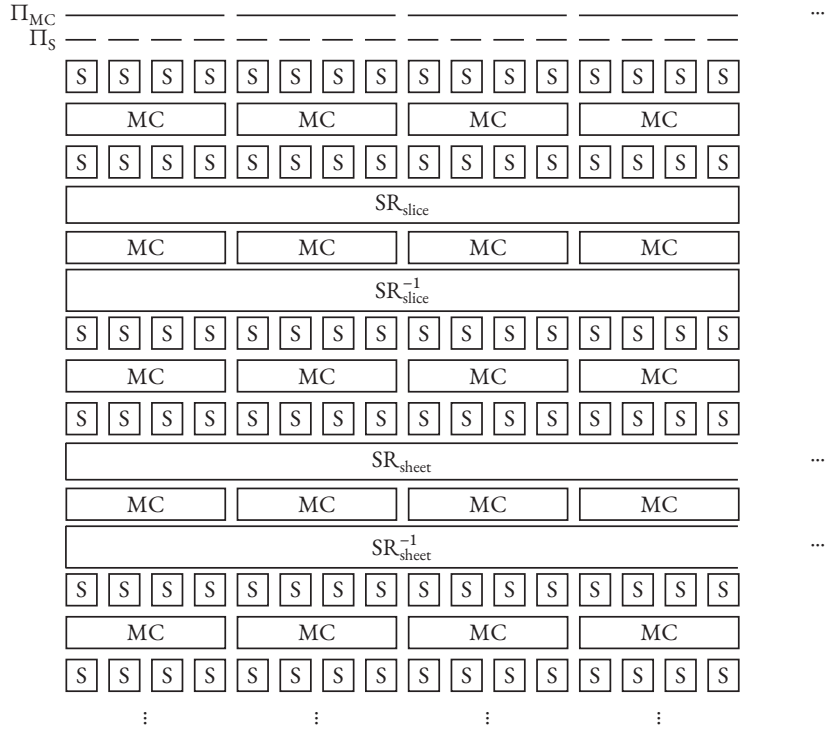


Figure 4.2: Alignment properties of Saturnin.

1. S*pongentMixLayer* applies the same mixing function S*pongentMix* in parallel to the 24 *subgroups* (following the terminology of [9]). It is a bit shuffle associated with the index permutation $\tau_{\text{subgroup}} \colon [0, 15] \to [0, 15]$:

$$\tau_{\text{subgroup}}(j) = \begin{cases} 4j \bmod 15, & \text{if } j \in [0, 14] \\ 15, & \text{if } j = 15 \end{cases}$$

2. S*pongentBoxShuffle* is a box shuffle that is associated with the box index permutation

$$\tau_{\text{box}} \colon [0, 95] \to [0, 95]$$

defined by:

$$\tau_{\text{box}}(j) = \left\lfloor \frac{j}{4} \right\rfloor + 24(j \bmod 4) .$$

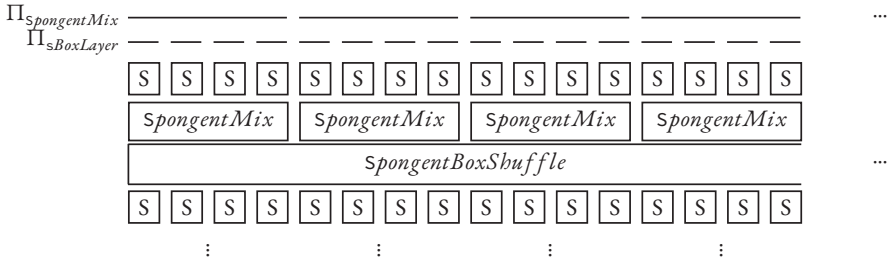


Figure 4.3: Alignment properties of Spongent.

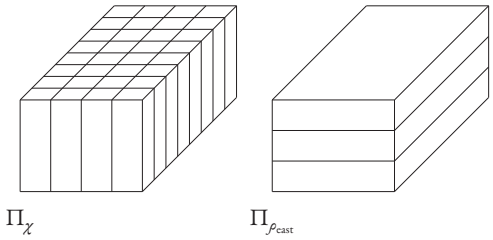


Figure 4.4: Alignment properties of Xoodoo.

The sBoxLayer defines a box partition $\Pi_{\text{sBoxLayer}}$ corresponding to the 96 4-bit boxes. The box shuffle $\text{SpongentBoxShuffle}$ is a $\Pi_{\text{sBoxLayer}}$-shuffle. The bit shuffle SpongentMixLayer is aligned to a non-trivial partition $\Pi_{\text{SpongentMixLayer}}$ that divides the state into 96 16-bit subgroups, each grouping four consecutive boxes, and we have $\Pi_{\text{sBoxLayer}} \leq \Pi_{\text{SpongentMixLayer}}$. It follows that Spongent is aligned. Figure 4.3 shows these steps and their alignment properties.

4.4.4 Xoodoo

Xoodoo [16] is a permutation with $b = 384$. The state consists of 3 equally sized horizontal *planes*, each one consisting of 4 parallel 32-bit *lanes*. Alternatively, the state can be seen as a set of 128 *columns* of 3 bits, arranged in a 4×32 array.

The round function of Xoodoo consists of the following five steps: a mixing layer θ, a bit shuffle ρ_{east}, round constant addition ι, a nonlinear layer χ, and a bit shuffle ρ_{west}. The χ step applies the same 3-bit S-box to the columns of the state. The nonlinear layer χ defines a box partition Π_{χ} that corresponds to the 128 columns. The bit shuffles ρ_{east} and ρ_{west} perform translations of planes and are not aligned to Π_{χ}. The mixing layer θ defines no non-trivial box partition at all. Due to the properties of the ρ steps and θ it is impossible to split the linear layer in a column shuffle and a mixing layer that is aligned to a partition that Π_{χ} is a refinement of. In other words, Xoodoo is unaligned. See Section E for a more formal proof. Figure 4.4 shows the alignment properties of the steps.

4.4.5 Round cost

In this section, we compare the implementation complexity of the round functions of the four ciphers. This depends on the platform and the requirements. Platforms may range from low-end 8-bit CPUs to multi-core high-end workstation CPUs, FPGAs, and even dedicated hardware. Requirements include throughput, latency, usage of resources such as power and energy consumption, area in hardware, and RAM/ROM usage in software. Moreover, protection against fault attacks and/or side channel attacks may be required.

In our comparison of the round functions we let their three layers guide us: the S-box layer, the mixing layer (if any), and the shuffle layer. We also discuss the presence of key addition in block ciphers and its relative cost.

S-box layer

Given that our ciphers have invertible S-boxes with lowest known maximum DP and LP values that can be achieved for their width, their implementation cost increases with width.

We report on the implementations with minimum number of binary XOR, binary AND/OR, and unary NOT operations that we found in the literature. For Spongent we found no such numbers. We have also determined a minimal sum-of-products (SOP) form in Boolean algebra of the S-boxes using the Espresso algorithm [31] for two-level logic optimization. For Rijndael, finding the minimal SOP was infeasible. We refer to Section B for the SOP expressions. Using De Morgan's laws, the SOP form can be implemented by two layers of nand gates. Table 4.1 lists the number of nand gates per bit for each of the S-boxes.

We can see in Table 4.1 that the cost of the Saturnin and Spongent S-boxes is comparable. The cost of the Xoodoo S-box is roughly half of that, but is only 3 bits wide instead of 4. The Rijndael S-box is a roughly a factor 10 more costly than that of Saturnin and Spongent, a very high price for its better max DP/LP value. These numbers give an indication for the size of a hardware circuit and the number of cycles in bit-sliced software implementations. The number of and/or operations is related to the cost of masking countermeasures.

Mixing layer

Spongent has no mixing layer, so there is no cost. Xoodoo-θ requires 2 binary xor operations per bit, while Saturnin's MC can be implemented with 2.25 binary xor operations per bit [13]. The circuit depth for these computations is in both cases 4 xor gates. Despite the difference in design philosophy, their computational costs are almost the same.

A simple implementation of Rijndael's MixColumns takes 3.875 binary xor operations per bit and has a circuit depth of 3 xor gates. This was reduced to $97/32 \approx 3$ additions per bit [25] at the expense of a higher circuit depth. Despite the fact that both MixColumns and Saturnin's MC implement an MDS mapping operating on 5 boxes, their costs diverge. The main difference between the two is that MixColumns operates on bytes while MC operates on nibbles. However, this is not the reason for the

higher cost per bit of MixColumns. The reason is that there have been significant advances in building efficient MDS mappings and MC reaps the benefits of that.

Shuffle layer

Rijndael, Spongent, and Xoodoo consist of the iteration of a single round function. In a hardware architecture that implements the full round in combinational logic, a bit shuffle consists of wiring between gates. Saturnin has three different rounds, so this is more complex in a hardware architecture in which a single round is implemented in combinational logic. However, in a combinational block that implements a sequence of four rounds, the shuffle operations do correspond to wiring.

We compare software implementation on a particular platform: the ARM Cortex-M4 processor. We choose this because it is a popular lightweight platform for benchmarks and for three of our ciphers there is assembly code available. On this platform, it is difficult to assess the cost of the shuffle layer in isolation due to the *barrel shifter*. This feature of the ARM architecture allows applying (cyclic) shift operations to one of the two operands in arithmetic and bitwise Boolean instructions at no additional cost. To compare, we measure the number of cycles of the entire round function, revealing the marginal cost of the shuffle layer. Table 4.2 lists the performance of the round functions of our four ciphers expressed in number of cycles per byte as measured on a Cortex-M4 processor. In addition, it includes references to the bit-sliced implementations that we have used in order to measure the cycle counts. In Rijndael and Saturnin we removed any operations related to the key addition to make a fair comparison possible and in Saturnin we measured the number of cycles for 4 rounds and divided that by 4. We have not included Spongent because we do not have access to any (optimized) assembly code. However, considering that it was designed with hardware in mind, we do not believe it is competitive in software.

4.5 Huddling

In this section, we describe a phenomenon that we call *huddling*. We present the bit and box weight histograms as natural extensions of the bit and box branch numbers, respectively. Using these histograms, we analyze the huddling properties of the ciphers described in Section 4.4. We see that these properties are more pronounced in ciphers that are aligned. Finally, we look at the relation between huddling and the distribution of trail weights.

4.5.1 Definitions of bit weight, box weight and their histograms

The weight of a two-round trail $(q_{\text{in}}, a, b, q_{\text{out}})$ over $\mathrm{N} \circ \mathrm{L} \circ \mathrm{N}$ can be bounded from below by the sum of the number of active boxes at the input and output of L. This number is fully determined by a as $b = \mathrm{L}(a)$ in differential trails and $a = \mathrm{L}^{\mathsf{T}}(b)$ in linear trails. The distribution of states a according to this number determines the *mixing power* of the linear layer with respect to Π_{N}.

First, we formally define what it means for a box to be active. To this end, we define an *indicator function* $1_i \colon \mathbb{F}_2^b \to \mathbb{F}_2$ with respect to a box partition Π by $1_i(a) = 0$ if $P_i(a) = 0$ and $1_i(a) = 1$ otherwise.

We call the box B_i *active* in the difference or linear mask $a \in \mathbb{F}_2^b$ if $1_i(a) = 1$ and *passive* otherwise. The natural metric associated with box activity is the *box weight* of a, defined by $\mathrm{w}_\Pi(a) = |\{i \in [0, n-1] : 1_i(a) \neq 0\}|$. Clearly, a box is active in a difference or linear mask if at least one of the bits in that box is non-zero. We call the bit *i active* in a if $a_i = 1$ and *passive* otherwise. The number of active bits is given by the *bit weight* of a, i.e., $\mathrm{w}_2(a) = |\{i \in [0, b-1] : a_i \neq 0\}|$. The *activity pattern* of a is defined by $\eta_\Pi(a) = \sum_{i=0}^{n-1} 1_{\mathrm{B}_i}(a) e_i^n$. It is the vector whose ith component is one if box B_i is active and zero otherwise.

In order to quantify the mixing power of a linear transformation L, we consider the weight distribution of $(a, \mathrm{L}(a))$ over all differences or linear masks $a \in \mathbb{F}_2^b$ and embed it in a histogram. This is a well-known concept in coding theory, where weight distributions are embedded in so-called weight enumerator polynomials that classify the code [23].

Definition 42. *The* weight histogram *of a linear transformation* $\mathrm{L} \colon \mathbb{F}_2^b \to \mathbb{F}_2^b$ *is a function* $\mathcal{N}_{\cdot,\mathrm{L}} \colon \mathbb{Z}_{\geq 0} \to \mathbb{Z}_{\geq 0}$ *given by*

$$\mathcal{N}_{\cdot,\mathrm{L}}(k) = |\{a \in \mathbb{F}_2^b : \mathrm{w}_\cdot(a) + \mathrm{w}_\cdot(\mathrm{L}(a)) = k\}| .$$

The cumulative version on the same domain and codomain is given by

$$\mathcal{C}_{\cdot,\mathrm{L}}(k) = \sum_{l \leq k} \mathcal{N}_{\mathrm{L}}(l) .$$

Here, $\cdot$ *denotes either* 2 *or* Π.

The *tail* of the histogram consists of the left-most values that correspond to low weight.

If the primitive is aligned, then π is a box shuffle and this implies that the box weight histograms of $\mathrm{L} = \mathrm{M} \circ \pi$ and M are the same. The superbox structure of an aligned primitive makes it possible to use a divide-and-conquer approach to compute the weight histograms. Indeed, let $S(\mathrm{w}) = \{v \in \mathbb{Z}_{\geq 0}^s : \sum_{i=0}^{s-1} v_i = \mathrm{w}\}$ with s the number of superboxes. Then we can compute the weight histograms of M by *convolving* the weight histograms of its box functions:

$$\mathcal{N}_{\cdot,\mathrm{M}}(\mathrm{w}) = \sum_{v \in S(\mathrm{w})} \prod_{i=0}^{s-1} \mathcal{N}_{\cdot,\mathrm{M}_i}(v_i) . \tag{4.1}$$

We note that the *differential branch number* [14] is simply the smallest non-zero entry of this histogram, i.e., $\min\{\mathrm{w} > 0 : \mathcal{N}_{\cdot,\mathrm{L}}(\mathrm{w}) > 0\}$. The linear branch number is the smallest non-zero entry in the corresponding histogram of $\mathrm{L}^\top$ and can be different from its differential counterpart. This is not the case for the mappings in this paper and we will omit the qualifier in the remainder. A higher branch number typically implies higher mixing power. However, the weight histogram is more informative than just the branch number. The number of differences or linear masks meeting the branch number is valuable information as well. In general, the weight histogram allows a more nuanced comparison of mixing layers than the branch number.

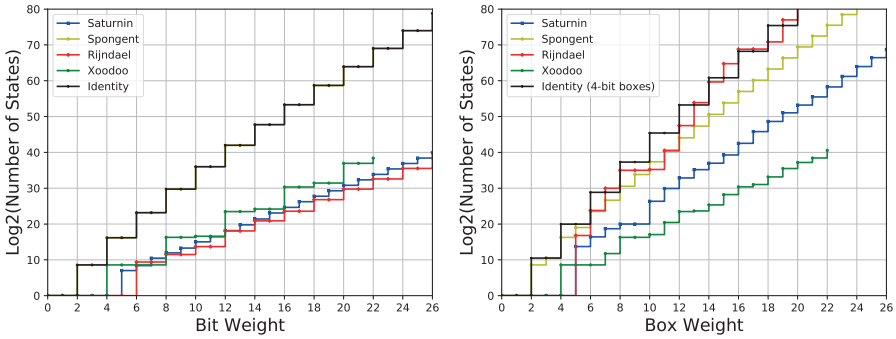


Figure 4.5: Cumulative bit weight and box weight histograms.

The box weight histogram is the relevant histogram in the context of the wide trail design strategy [21]. A linear layer that systematically has lower values in the tail of its box weight histogram than the other does typically has fewer two-round trails with low weight, given equal nonlinear layers.

4.5.2 Bit and box weight histograms

We discuss the cumulative bit and box weight histograms for the linear layers of our four ciphers, given in Figure 4.5. We include the histogram for the identity function, assuming 4-bit S-boxes for the box weight to allow for comparison with Spongent and Saturnin.

The bit weight histogram for Spongent coincides with that of the identity permutation. This is because its linear layer is a bit shuffle. As the identity permutation maps inputs to identical outputs, it has only non-zero entries for even bit weights. Its bit branch number is 2. In conclusion, its mixing power is the lowest possible.

The bit branch number of the mixing layer of Rijndael, MixColumns, is 6, that of Saturnin-MC is 5, and that of Xoodoo-θ is 4.

Similar to Spongent, the bit weight histograms of Rijndael and Xoodoo have only non-zero entries at even bit weights. This is because both Xoodoo-θ and Rijndael-MixColumns can be modeled as $a \mapsto (\mathrm{I} + \mathrm{M})a$ for some matrix $\mathrm{M} \in \mathbb{F}_2^{b \times b}$ with the property that the bit weight of $\mathrm{M}\,a$ is even for all $a \in \mathbb{F}_2^b$. Saturnin-MC cannot be modeled in that way and does have non-zero entries at odd bit weights.

The bit weight histograms of Rijndael and Saturnin are very close and that of Xoodoo is somewhat higher. The ranking per bit weight histogram reflects the computational resources invested in the mixing layer: Rijndael uses 3.5 additions per bit, Saturnin 2.25, Xoodoo 2, and Spongent 0.

In the box weight histograms we see the following. For Spongent the box branch number is 2, the same as the bit branch number. However, the box weight histogram of Spongent has a lower tail than the identity permutation. What it shows is the mixing power of S*pongentMixLayer* in our factorization of p*Layer*, operating on 4-box superboxes.

The box branch number of the linear layers of Rijndael, MixColumns, and of Saturnin-MC are both 5, while for Xoodoo it is 4.

The discrepancy between the bit and box weight histogram brings us to the notion of *bit huddling*: many active bits huddle together in few active boxes. We say that the bit huddling in a linear layer is *high* if the concentration is high and we say that the bit huddling is *low* otherwise.

Huddling has an effect on the contribution of states a to the histogram, i.e., by definition we have that $w_\Pi(a) + w_\Pi(L(a)) \leq w_2(a) + w_2(L(a))$. In words, from bit to box weight, huddling moves states to the left in the histogram, thereby raising the tail. Huddling therefore results in the decay of mixing power at box level as compared to bit level. In the absence of huddling, the bit and box weight histogram would be equal. However, huddling cannot be avoided altogether as states do exist with multiple active bits in a box (note that $m \geq 2$).

We see Rijndael has *high* bit huddling. In moving from bit weights to box weights, the branch number decreases from 6 to 5 and the tail rises from being the lowest of the four to the highest. This is a direct consequence of the large width of the Rijndael S-boxes, namely 8, and the byte alignment. Indeed, MixColumns only mixes bits within the 32-bit columns. We call this the *superbox huddling effect*. Of course, there is a reason for these large S-boxes: they have low maximum DP/LP values. They were part of a design approach assuming table-lookup implementations where the main impact of the S-box size is the size of the lookup tables. Unfortunately table-lookups are expensive in dedicated hardware and on modern CPUs lookup tables are kept in cache making such implementations susceptible to cache-timing attacks [4].

Saturnin, with its Rijndael-like structure also exhibits the superbox huddling effect, though less pronounced than Rijndael. From bits to boxes the branch number does not decrease and the tail rises less than for Rijndael. Clearly, its smaller S-box size, namely 4, allows for less bit huddling. Due to its alignment, Spongent exhibits the superbox huddling effect, but less so than Saturnin. The reason for this is the already high tail in the bit weight histogram, due to the absence of bit-level diffusion in the mixing layer.

Finally, Xoodoo has the *lowest* bit huddling of the four primitives studied. This is the consequence of two design choices: having very small S-boxes (3-bit) and the absence of alignment, avoiding the superbox huddling effect altogether.

4.5.3 Two-round trail weight histograms

We define the trail weight histogram analogous to Definition 42 with the change that

$$\mathcal{N}_\cdot(k) = \left|\left\{\text{trails } Q : w_\cdot(Q) = k\right\}\right|,$$

where $\cdot$ is either r for differential trails or c for linear trails. Like for the other diagrams, the lower the tail, the lower the number of states with small weights, the better.

Figure 4.6 reports on the distribution of the weight of two-round differential and linear trails of our four ciphers. To compute the trail weight histograms of the aligned ciphers, we convolved the histograms

of the superbox structures (See Equation 4.1). The distribution of the linear trails for Rijndael is an approximation that was obtained by first taking the integer part of the correlation weights of its S-box to allow for integer arithmetic. The other distributions are exact.

While Rijndael performed the worst with respect to the box weight metric, we see that it performs the best with respect to the trail weights. The reasons are the low maximum DP/LP value of its S-box and its high branch number. However, as seen in Section 4.4.5, one pays a price in terms of the implementation cost. The relative ranking of the other ciphers does not change in moving from box weight to trail weights. Still, Xoodoo loses some terrain due to its more lightweight S-box layer.

Despite the difference in design approach, Xoodoo and Saturnin have quite similar two-round trail weight histograms. It is therefore interesting how the trail weight histograms compare for three and four rounds.

4.5.4 Three-round trail weight histograms

We have computed the three-round differential and linear trail weight histograms for Saturnin and Xoodoo and give them in Figure 4.7. We did not do it for Rijndael due to the prohibitively high cost of its round function and neither for Spongent due to its non-competitive bounds for multiple-round trails as reported in [9]. Hence, we focus on Saturnin and Xoodoo as exponents of the aligned and unaligned wide-trail design approaches. Computing the three-round Saturnin trail histograms turned out to be very computationally intensive for higher weights (see Subsection A.3 for more details) and we were forced to stop at weight 36. Still, the diagrams show the big difference in histograms between Saturnin and Xoodoo.

Despite the fact that the box branch number of Xoodoo is 4 and that of Saturnin is 5, we see that for three-round trails, Xoodoo performs much better than Saturnin. In particular, Xoodoo has no trails with weight below 36, whereas Saturnin has about 2^{43} linear trails with weight below 36, starting from weight 18. Moreover, it has about 2^{47} differential trails with weight below 36, starting from weight 19. This confirms the idea that branch number alone does not paint the whole picture and that these histograms prove to be very useful in comparing the different design approaches.

4.5.5 Four rounds and beyond

We did not conduct experiments for four or more rounds, but can make use of available information. According to [15], there exist no differential or linear trails over four rounds of Xoodoo with weight below 74. In contrast, Saturnin has roughly 2^{82} four-round differential trails with 25 active S-boxes and it has more than $2^{94.5}$ such linear trails. See Section C for a derivation of this estimate. Since each S-box has a weight of 2 or 3, this implies many four-round differential trails with weights in the range $[50,\ 75]$. The linear trails have weights in the range $[50,\ 100]$ due to the fact that active S-boxes have weight 2 or 4. Naturally, in both cases there are also trails with 26, 27, ... active S-boxes and their number grows quickly with the box weight due to the additional degrees of freedom in building them. It follows that the trend

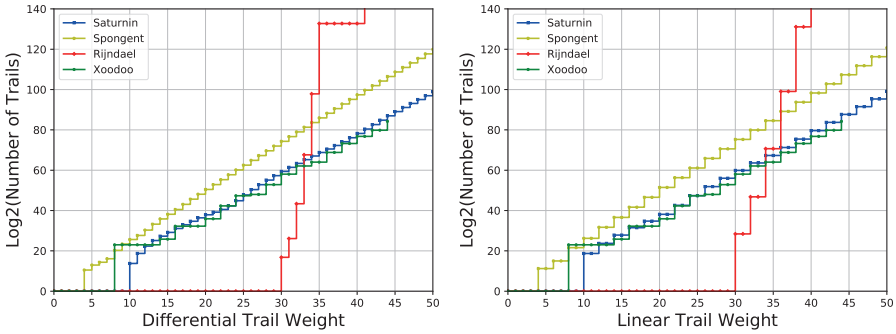


Figure 4.6: Two rounds: cumulative differential and linear trail weight histograms.

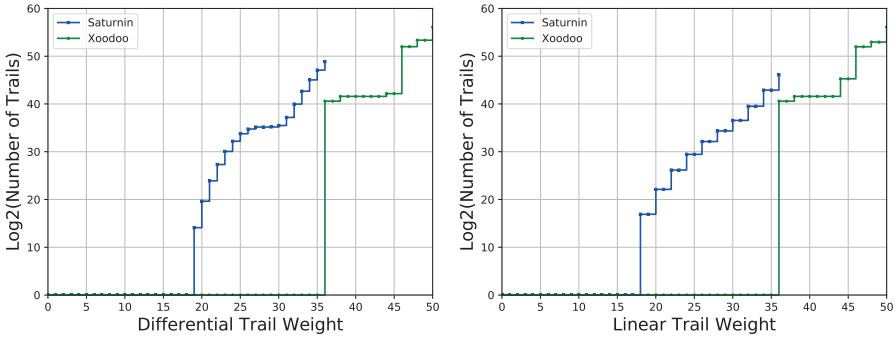


Figure 4.7: Three rounds: cumulative differential and linear trail weight histograms.

we see in three-round trails persists for four-round trails: unaligned Xoodoo has a significantly lower tail than aligned Saturnin, despite its lighter round function and lower branch number.

For trails over five rounds and more we report on the known lower bounds on weight in Table 4.6 in Section D. We see that up to 6 rounds Xoodoo remains ahead of Saturnin. For higher weights the trail scan programs in Xoodoo reach their computational limit and Saturnin overtakes Xoodoo. Advances in trail scanning are likely to improve the bounds for Xoodoo while for Saturnin the currently known bounds are much more tight. For the whole range Rijndael is well ahead and Spongent is invisible with its weight of 28 for 6 rounds.

4.6 Clustering

In this section, we investigate clustering of differential trails and of linear trails. The occurrence of such clustering in two-round differentials and linear approximations requires certain conditions to be satisfied. In particular, we define an equivalence relation of states with respect to a linear layer and an S-box partition that partitions the state space in candidate two-round trail cores and the size of its equivalence

classes upper bounds the amount of possible trail clustering. This is the so-called cluster partition. We present the partitions of our four ciphers by means of their cluster histograms. For all four ciphers, we report on two-round trail clustering and for Xoodoo in particular we look at the three-round case. With its unaligned structure, we found little clustering in Xoodoo. However, the effects of clustering are apparent in the aligned primitives Rijndael, Saturnin, and Spongent, with them being most noticeable in Rijndael.

4.6.1 The cluster histogram

To define the cluster histogram we need to define two equivalence classes.

Definition 43. *Two states are box-activity equivalent if they have the same activity pattern with respect to a box partition* Π*:*

$$a \sim a' \text{ if and only if } r_\Pi(a) = r_\Pi(a') .$$

We denote the set of states that are box-activity equivalent with a by $[a]_\sim$ *and call it the* box-activity class *of a.*

Box-activity equivalence has an application in the relation between trail cores and differentials and linear approximations.

Proposition 14. *Two trail cores* $(a_0, b_0 \dots, a_{r-2}, b_{r-2})$ *and* $(a_0^*, b_0^* \dots a_{r-2}^*, b_{r-2}^*)$ *over a function*

$$f = \mathrm{N}_{r-1} \circ \mathrm{L}_{r-2} \circ \mathrm{N}_{r-2} \circ \cdots \circ \mathrm{L}_0 \circ \mathrm{N}_0$$

that are in the same differential (or linear approximation) satisfy $a_0 \sim a_0^*$ *and* $b_{r-2} \sim b_{r-2}^*$*.*

Proof. Let $(\Delta_{\text{in}}, \Delta_{\text{out}})$ be the differential over f that the trail cores are in. Since N_0 and N_{r-2} preserve activity patterns, we have that $\Delta_{\text{in}} \sim a_0$, and $\Delta_{\text{in}} \sim a_0^*$, and $\Delta_{\text{out}} \sim b_{r-2}$, and $\Delta_{\text{out}} \sim b_{r-2}^*$. From the symmetry and transitivity of $\sim$ it follows that $a_0 \sim a_0^*$ and $b_{r-2} \sim b_{r-2}^*$. □

Considering the case $r = 2$ in Proposition 14 immediately gives rise to a refinement of box-activity equivalence.

Definition 44. *Two states are cluster-equivalent with respect to a linear mapping* $\mathrm{L} : \mathbb{F}_2^b \to \mathbb{F}_2^b$ *and a box partition* Π *if they are box-activity equivalent before* L *and after it (See Figure 4.8):*

$$a \approx a' \text{ if and only if } a \sim a' \text{ and } \mathrm{L}(a) \sim \mathrm{L}(a') .$$

We denote the set of states that are cluster-equivalent with a by $[a]_\approx$ *and call it the* cluster class *of a. The partition of* $\mathbb{F}_2^b$ *according to these cluster classes is called the* cluster partition*.*

Corollary 3. *If two two-round trail cores* $(a, \mathrm{L}(a))$ *and* $(a^*, \mathrm{L}(a^*))$ *over* $f = \mathrm{N} \circ \mathrm{L} \circ \mathrm{N}$ *are in the same differential, then* $a \approx a^*$*.*

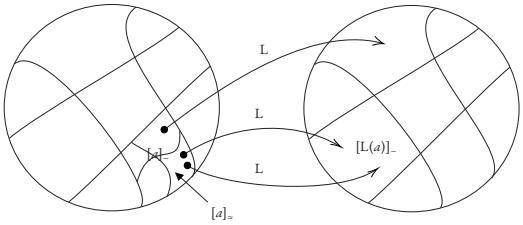


Figure 4.8: Partitions of $\mathbb{F}_2^b$ defined by $\sim$ and $\approx$.

Proof. If we apply Proposition 14 to the case $r = 2$, we have $a \sim a^*$ and $\mathrm{L}(a) \sim \mathrm{L}(a^*)$. It follows that $a \approx a^*$. □

Corollary 3 shows that the defining differences of any two-round trail cores that cluster together are in the same cluster class. It follows that if these cluster classes are small, then there is little clustering.

For all $a' \in [a]_{\approx}$ the box weight $\mathrm{w}_\Pi(a') + \mathrm{w}_\Pi(\mathrm{L}(a'))$ is the same. We denote this weight by $\widetilde{\mathrm{w}}([a]_{\approx})$.

Definition 45. *Let* $\mathrm{L} : \mathbb{F}_2^b \to \mathbb{F}_2^b$ *be a linear transformation. Let* $\approx$ *be the equivalence relation given in Definition 44. The* cluster histogram $\mathrm{N}_{\Pi,\mathrm{L}} : \mathbb{Z}_{\geq 0} \times \mathbb{Z}_{\geq 0} \to \mathbb{Z}_{\geq 0}$ *of* L *with respect to the box partition* Π *is given by*

$$\mathrm{N}_{\Pi,\mathrm{L}}(k, c) = \left|\{[a]_{\approx} \in \mathbb{F}_2^b/{\approx} : \widetilde{\mathrm{w}}([a]_{\approx}) = k \wedge |[a]_{\approx}| = c\}\right|.$$

For a fixed box weight, the cluster histogram shows the distribution of the sizes of the cluster classes with that box weight. Ideally, for small box weights, the cluster classes are all very small. Large cluster classes of small weight may lead to two-round trails with a large DP or LP.

4.6.2 The cluster histograms of our ciphers

Next, we present the cluster histograms of the superboxes of Rijndael, Saturnin, and Spongent and of the Saturnin hyperbox. Moreover, we present a partial cluster histogram of Xoodoo. The results for Rijndael and Saturnin are found in Table 4.3, for Spongent in Table 4.4, and for Xoodoo in Table 4.5. In these tables, C denotes the cardinality of a cluster class and N denotes the number of cluster classes with that cardinality. For instance, an expression such as (32×1) (36×7) means that there are 32 cluster classes of cardinality 1 and 36 classes of cardinality 7. Looking at $\widetilde{\mathrm{w}} = 8$ across the three tables, we see that Rijndael, Saturnin, and Spongent have only a single cluster class containing all the states with $\mathrm{w}_\Pi(a) + \mathrm{w}_\Pi(\mathrm{L}(a)) = 8$. In contrast, for Xoodoo, each state a sits in its own cluster class. This means that $\mathrm{L}(a)$ is in a different box activity class than $\mathrm{L}(b)$ for any $b \in [a]_{\sim}$ and $b \neq a$.

Thanks to the fact that the mixing layers of Rijndael and Saturnin have the MDS property, the entries of their cluster histograms are combinatorial expressions of m, the box size, and n, the number of boxes. We describe these methods in detail in Subsection A.2.

Table 4.4 gives the cluster histogram of SPONGENT's superbox. For weights above 4 we see large cluster equivalence classes.

Now, consider the cluster histogram of XOODOO in Table 4.5. We see that up to and including box weight 13, we have $|[a]_{\approx}| = 1$. For box weight 14, 15, and 16, we see that $|[a]_{\approx}| \leq 2$. Due to its unaligned structure, it is less likely that equal activity patterns are propagated to equal activity patterns. Therefore, many cluster classes contain only a single state.

4.6.3 TWO-ROUND TRAIL CLUSTERING

Two-round trail clustering in the keyed RIJNDAEL superbox was investigated in [22]. In that paper the *expected* DP values of trails and differentials are studied, where expected means averaged over all keys. We see considerable clustering in differentials with 5 active S-boxes. For these, the maximum expected DP of differentials is more than a factor 3 higher than the maximum expected DP of 2-round trails, with differentials containing up to 75 trails. For more active S-boxes the number of trails per differential is much higher and hence clustering is worse, but their individual contributions to the expected DP are much smaller and all differentials have expected DP very close to 2^{-32}. For fixed keys or in an unkeyed superbox these differentials and trails have a DP that is a multiple of 2^{-31}. For trails this effect was studied in [19].

In this section we report on our experiments on the other three of our ciphers where we compare two-round differentials with differential trails and linear approximations with linear trails. Figure 4.9 shows the number of differentials and differential trails up to a given weight of the SATURNIN and the SPONGENT superboxes. In both cases, we see that for low weight the histograms are close and as the weight grows, these histograms diverge. For SATURNIN there are roughly 50 times more differentials with weight 15 or less than differential trails with weight 15 or less. For SPONGENT this ratio is roughly 20. This divergence is due to two reasons: clustering and what we call *clipping*. Due to the large number of differential trails and the limited width of the superbox, the trails cluster. This effect is especially strong for trails with almost all S-boxes active and would give rise to many differentials with DP close to 2^{-16} as the superbox has width 16. What we observe is a majority of differentials with DP equal to 2^{-15}. This is the result of the fact that any differential over a superbox has an even number of ordered pairs and hence the minimum DP is 2^{-15}, yielding weight 15. We call this effect clipping: the weight of differentials cannot be strictly greater than 15. A trail over a k-bit superbox with weight $w > k - 1$ cannot have a $DP = 2^{-w}$ as this would imply a fractional number of pairs. This effect has been studied in AES and we refer to Section 4.7 for a discussion.

Figure 4.10 shows the weight histograms for two-round differentials and linear approximations. The full-state correlation weight histogram of SATURNIN was obtained from that of any of its columns by first rounding the correlation weights to the nearest integer to make integer arithmetic possible. The full-state correlation weight histogram of SPONGENT was obtained in a similar manner. The remainder of the histograms is exact. Table 4.5 shows that in XOODOO almost all differentials contain only a single trail. This means that the clustering is negligible. Therefore, there is no difference between Figures 4.6 and 4.10

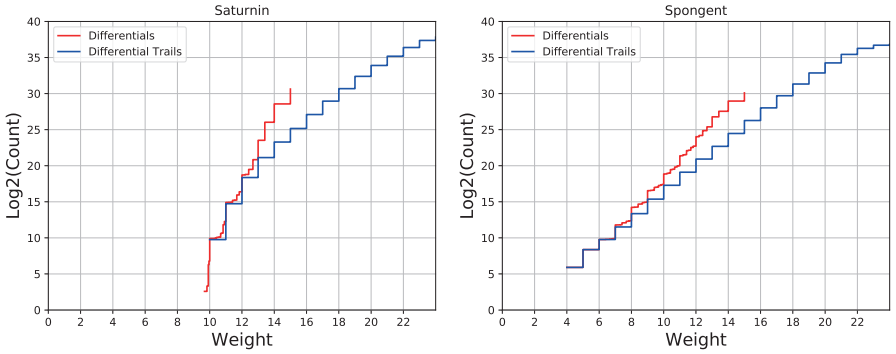


Figure 4.9: Differentials and differential trails in the superboxes of Saturnin and Spongent.

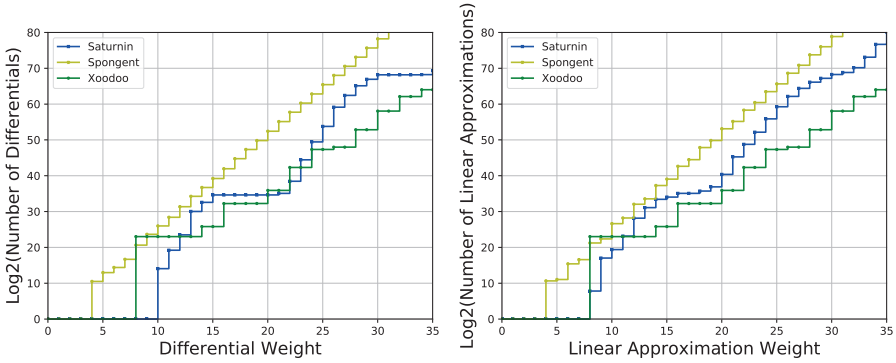


Figure 4.10: Two rounds: cumulative restriction and correlation weight histograms.

for Xoodoo. For Saturnin the clustering is the most striking. For linear trails we observe a similar effect. For Spongent the clustering is less outspoken due to the fact that the trail weight histogram is quite bad to start with.

The effect of clustering in four-round (or two super-round) Saturnin is interesting. Four-round Saturnin consists of the parallel application of four 64-bit hyperboxes. The consequence is that for a fixed key, the roughly $2^{127} \cdot 4$ differentials that are active in a single hyperbox and have non-zero DP, all have weight below 63. When computing expected DP values averaging the DP over all round keys, this is closer to 64.

The cluster classes also determine the applicability of the very powerful *truncated differential attacks* [24]. These attacks exploit sets of differentials that share the same box activity pattern in their input difference and the same box activity pattern in their output difference. Despite the fact that the individual trails in these truncated differentials may have very low DP, the joint probability can be significant due to the massive numbers. For two-round differentials the cluster classes are exactly the trail cores in a given truncated differential. In Table 4.3 we see that the cluster classes for the Rijndael superbox and

Saturnin hyperbox are very large. This clustering leads to powerful distinguishers for e.g., 4-round AES and 8-round Saturnin. The latter can be modeled as 4 hyperboxes followed by an MDS mixing layer followed by 4 hyperboxes and an input difference with a single active hyperbox will have 4 active hyperboxes after 8 rounds, with probability 1. In contrast, if the cluster classes are small, as in the case of the unaligned Xoodoo permutation, it is very unlikely that truncated differential attacks would have an advantage over ordinary differential attacks.

4.6.4 Three-round trail clustering in Xoodoo

Recall that for Xoodoo, no 4-round trails exist with weight below 74 and Table 4.5 showed that trail clustering in two-round differentials in Xoodoo is negligible, as expected because of its unaligned design. We investigate the conjecture that it is also the case for three rounds.

First, we present a generic technique to find all trails that have an enveloping differential compatible with a given three-round trail core. We apply the technique to Xoodoo, for which it is very efficient.

Given the trail core $(a_1^*, b_1^*, a_2^*, b_2^*)$, Proposition 14 shows that we can restrict ourselves to those tuples (a_1, b_1, a_2, b_2) with $a_1 \sim a_1^*$ and $b_2 \sim b_2^*$. The difference a_1^* defines a vector space A' of all the states in which a box is passive whenever it is passive in a_1^*. If $a_1 \in [a_1^*]_\sim$, then $a_1 \in A'$. Similarly, b_2^* defines a vector space B'. If $b_2 \in [b_2^*]_\sim$, then $b_2 \in B'$. The vector space $B = \mathrm{L}(A')$ contains the candidate values for b_1. Similarly, the vector space $A = \mathrm{L}^{-1}(B')$ contains candidate values for a_2. Because it preserves activity patterns, N restricts the set of candidate values to those satisfying $b_1 \sim a_2$. Hence, we can limit the search to those $x \in B$ and $y \in A$ with $x \sim y$.

To find all valid trails of the form $(\Delta_{\text{in}}, a_1, b_1, a_2, b_2, \Delta_{\text{out}})$, we first reduce the size of the space of all trail cores (a_1, b_1, a_2, b_2) using a necessary condition. When this space is small enough, we exhaustively search for a valid trail.

We write $\overline{B}$ for a basis of B and $\overline{A}$ for a basis of A. To reduce the dimension of the spaces, we will apply an algorithm directly on their bases. First, we need the notion of *isolated active bit*.

Definition 46. *A bit i of $b \in \overline{B}$ is said to be an isolated active bit if $b_i = 1$ and $b'_i = 0$ for all $b' \in \overline{B} \setminus \{b\}$.*

A basis vector having an isolated active bit determines the box activity of any linear combination that includes it.

Proposition 15. *If $b \in \overline{B}$ has an isolated active bit in position i, then any vector in the affine space $b + \mathrm{span}(\overline{B} \setminus \{b\})$ has the corresponding box activated.*

Proof. If b has an isolated active bit in position i, then the ith bit of any vector in $b + \mathrm{span}(\overline{B} \setminus \{b\})$ is active. As a result, the box containing this bit is active. □

Similar to how an isolated active bit always activates the corresponding box, a box is never activated if no basis vector activates it.

Proposition 16. *If the ith box is passive in every vector of $\overline{A}$, then the ith box is passive in all vectors of A. We say that box i is passive in $\overline{A}$.*

We define a condition that makes it possible to remove a basis vector from the basis without excluding potentially valid trails.

Condition 1. *We say that a basis vector $b \in \overline{B}$ satisfies the* reduction condition *if and only if it has an isolated active bit in a box that is passive in $\overline{A}$. The same is true when swapping the role of $\overline{B}$ and $\overline{A}$.*

The following lemma shows that the reduction condition is sufficient to reduce the dimension of the vector space we consider.

Proposition 17. *If a basis vector $b \in \overline{B}$ satisfies Condition 1, then all valid differences before the* N *in the middle are in span$(\overline{B} \setminus \{b\})$. The same is true when swapping the role of $\overline{B}$ and $\overline{A}$.*

Proof. As a consequence of Proposition 15 and Proposition 16, a valid difference before the nonlinear layer cannot be constructed from $b^{(i)}$ because it would contradict the fact that the activity pattern is preserved through the nonlinear layer. □

The algorithm now consists in repeatedly removing basis vectors from $\overline{B}$ and $\overline{A}$ that satisfy Condition 1 until this is no longer possible. This can be done efficiently by searching for pivots for a Gaussian elimination among indices of vectors from $\overline{A'}$ (respectively $\overline{B'}$) that correspond to never activated boxes in $\overline{B'}$ (respectively $\overline{A'}$). Indeed, these pivots can be used to row-reduce the corresponding basis along them, thus revealing an isolated active bit.

If the algorithm sufficiently decreased the dimensions, then we can exhaustively test all pairs $(b_1, a_2) \in B \times A$ (after reduction) according to the following criteria:

- (b_1, a_2) is a valid differential over N;
- There exists a Δ_{in} such that both $(\Delta_{\text{in}}, a_1^*)$ and $(\Delta_{\text{in}}, a_1)$ are valid differentials over N;
- There exists a Δ_{out} such that both $(b_2^*, \Delta_{\text{out}})$ and $(b_2, \Delta_{\text{out}})$ are valid differentials over N.

Applying our method to all three-round trail cores of Xoodoo up to weight 50 [17] shows that there exists no cluster for all these trails.

4.7 Dependence of round differentials

In this section we study the dependence of round differentials in the sense of Definition 30 in Section 4.2.1. It has been found in [19] that the vast majority of trails over the Rijndael superbox have dependent round differentials. We will investigate this for differential trails over three-round Xoodoo. We expect that the dependence effects observed in Rijndael disappear in an unaligned cipher. Hence, we now investigate this for differential trails over three-round Xoodoo.

4.7.1 Masks for differentials over nonlinear components

We note $V_{\mathrm{N}}(\Delta_{\mathrm{in}}, \Delta_{\mathrm{out}})$ the set of output states that follow the differential $(\Delta_{\mathrm{in}}, \Delta_{\mathrm{out}})$ over N, i.e. $V_N(\Delta_{\mathrm{in}}, \Delta_{\mathrm{out}}) = \mathrm{N}(U_{\mathrm{N}}(\Delta_{\mathrm{in}}, \Delta_{\mathrm{out}}))$. From [19], we have that $U_{\mathrm{N}}(\Delta_{\mathrm{in}}, \Delta_{\mathrm{out}})$ and $V_{\mathrm{N}}(\Delta_{\mathrm{in}}, \Delta_{\mathrm{out}})$ are affine if

$$|U_{\mathrm{S}_i}(P_i(\Delta_{\mathrm{in}}), P_i(\Delta_{\mathrm{out}}))| \leq 4$$

for each S-box. Since this assumption holds for our four ciphers, both $U_{\mathrm{N}}(\Delta_{\mathrm{in}}, \Delta_{\mathrm{out}})$ and $V_{\mathrm{N}}(\Delta_{\mathrm{in}}, \Delta_{\mathrm{out}})$ are affine and can be described by a system of affine equations on the bits of the state x. Each affine equation can be written as $u^{\top}x + c$ with u a b-bit vector called mask and c a bit.

Given a three-round differential trail $Q = (\Delta_{\mathrm{in}}, a_1, b_1, a_2, b_2, \Delta_{\mathrm{out}})$, one can define four sets of masks:

- A_1, the masks that come from $V_{\mathrm{N}}(\Delta_{\mathrm{in}}, a_1)$;
- B_1, the masks that come from $U_{\mathrm{N}}(b_1, a_2)$;
- A_2, the masks that come from $V_{\mathrm{N}}(b_1, a_2)$;
- B_2, the masks that come from $U_{\mathrm{N}}(b_2, \Delta_{\mathrm{out}})$.

These masks are said to be all *independent* if

$$|U_{\mathrm{N}\circ\mathrm{L}\circ\mathrm{N}\circ\mathrm{L}\circ\mathrm{N}}(Q)| = 2^{b-(|A_1|+|B_1|+|B_2|)} = 2^{b-(|A_1|+|A_2|+|B_2|)}.$$

which is, per Definition 30, equivalent to the independence of round differentials.

We first present an efficient generic method for determining whether three-round trail masks are independent. Then we apply this method to Xoodoo. Since L is linear, A_1 can be linearly propagated through it to obtain a set of masks A_1' at the input of the second nonlinear layer. Similarly, we can propagate B_2 through the inverse linear layer to obtain a set of masks B_2' at the output of the second nonlinear layer.

4.7.2 Independence of masks over a nonlinear layer

B_1 and A_1' form sets of masks at the input of the second nonlinear layer. If the rank of $C_1 = B_1 \cup A_1'$ is the sum of the ranks of B_1 and A_1', then C_1 contains independent masks. The same strategy can be used to test for dependence of masks in $C_2 = A_2 \cup B_2'$.

As for the independence of masks of the complete trail, we need to check for dependence between C_1 and B_2' or between A_1' and C_2. We will apply an algorithm similar to the one we used in Section 4.6.4 to reduce bases. However, here we use it to reduce the cardinalities of the mask sets.

The following lemma makes this possible.

Proposition 18. *Let C_1 and B_2' be two sets of masks before and after an S-box layer. If a mask u in C_1 satisfies Condition 1, then the number of states that satisfy the equations associated with the masks in both*

$C_1 \setminus \{u\}$ and B'_2 is exactly two times the number of states before removing u. The same is true by swapping the role of C_1 and B'_2.

Proof. Since u satisfies Condition 1, let i be the index of the isolated bit, j be the index of the corresponding S-Box and k the number of masks in B'_2. No mask in B'_2 is putting a constraint on any of the m bits of the jth S-Box, thus the 2^{b-k} solutions can be seen as 2^{b-k-m} groups of 2^m different states that only differ in the m bits of the jth S-box. Since the S-box is invertible, the application of the inverse of the nonlinear layer to a whole group of 2^m vectors results in a group of 2^m different states that, again, only differ on the value of the jth S-box.

We can further divide those 2^{b-k-m} groups each into 2^{m-1} subgroups of 2 different states that only differ in the value of the ith bit. By definition on an isolated bit, either both or none of the two states inside a subgroup satisfy all equations associated with the masks in $C_1 \setminus \{u\}$. Finally, inside a subgroup exactly one of the two states will satisfy the equation associated with mask u. Thus, the number of solutions by removing u is multiplied by exactly two. □

We first check for linear dependence inside C_1 by computing its associated rank. Then, we recursively check if some mask in either C_1 or B'_2 satisfies Condition 1 and if it is the case we remove them from the sets of masks.

There are three possible outcomes when applying this process to a three-round differential trail:

- If C_1 is not full rank, we can conclude that masks in B_1 and A'_1 are dependent;
- Else, if either set is empty, Proposition 18 applied at each step guarantees us that the number of states satisfying the equations associated with the masks in both C_1 and B'_2 is equal to $2^{b-(|C_1|+|B'_2|)}$, that is to say the masks are independent;
- If none of the two conditions above are met, we cannot directly conclude about (in)dependence between remaining masks but we can apply the same method to A_1 and C_2 and hope for a better outcome.

4.7.3 Application to Xoodoo

This process is used to check for independence in differential trails over three rounds of Xoodoo. It has been applied to the same differential trails as processed in Section 4.6.4. In all cases, the masks, and thus round differentials, were found to be independent. This was not obtained by sampling, but instead by counting the number of solutions, hence this independence is exact in the sense of Definition 30. As a result, the DP of each such trail is the product of the DP values of its round differentials, which implies that $\mathrm{DP}(Q) = 2^{-\mathrm{w_r}(Q)}$.

4.8 Conclusion

We put forward alignment as a crucial property that characterizes the interactions between linear and nonlinear layers w.r.t. the differential and linear propagation properties. We conducted experiments on

four S-box based primitives that otherwise represent different design approaches. We precisely defined what it means for a primitive to be aligned and showed that Rijndael, Saturnin, and Spongent are aligned, whereas Xoodoo is unaligned. Through these examples, we highlighted and analyzed different effects of alignment on the propagation properties.

Acknowledgements. We thank Bart Mennink for helpful comments. Moreover, we would like to thank the anonymous reviewers of an earlier version of this paper for their useful feedback. Joan Daemen and Daniël Kuijsters are supported by the European Research Council under the ERC advanced grant agreement under grant ERC-2017-ADG Nr. 788980 ESCADA. This work is partially supported by the French National Research Agency in the framework of the *Investissements d'avenir* programme (ANR-15-IDEX-02).

A Histogram computations

In this section, we describe methods for computing histograms. First, we describe a general method to obtain the histogram of an aligned function from the histograms of its box functions. Second, we describe some methods to obtain the cluster histograms of the ciphers described in Section 4.4.

Almost all of our computations were done to full precision. The only exception is the case of computing the correlation weights for Rijndael, Saturnin, and Spongent. In this case, we took the integer part of the intermediate results and computed on those numbers.

As a rule of thumb, the most interesting part of any histogram is its left tail, i.e., the part containing the distribution of the low weights. As a consequence of this, we are satisfied if we are able to compute a partial histogram that includes this tail. We see in the case of Rijndael and Xoodoo that this is frequently all we can hope to expect.

A.1 Convolution

Let $\mathrm{L} = \mathrm{L}_0 \times \cdots \times \mathrm{L}_{n-1} : \mathbb{F}_2^b \to \mathbb{F}_2^b$ be a function composed of box functions with respect to an ordered partition Π. Computing a histogram of L exhaustively is often computationally infeasible. However, the histograms of the box functions L_i typically are feasible to compute thanks to the small box width. Given the histograms of the L_i, it is possible to combine them to get the histogram of L itself. This combining operation is a form of convolution and is therefore denoted as $*$. The idea is best illustrated by an example.

Example 4. *Consider Rijndael and suppose we wish to compute the convolution of the restriction weight histograms of two of its S-boxes for a fixed output difference. We represent the histograms in two-column notation, where we list the restriction weights in the first column, and for each one its image, i.e., the number of input differences with the given weight, in the second column. To make the representation finite, a necessity for performing computations on these objects, we restrict ourselves to those weights for which the image is non-zero.*

$$\begin{bmatrix} 0 & 1 \\ 6 & 1 \\ 7 & 126 \end{bmatrix} * \begin{bmatrix} 0 & 1 \\ 6 & 1 \\ 7 & 126 \end{bmatrix} = \begin{bmatrix} 0+0 & 1 \cdot 1 \\ 0+6 & 1 \cdot 1 \\ 0+7 & 1 \cdot 126 \\ 6+0 & 1 \cdot 1 \\ 6+6 & 1 \cdot 1 \\ 6+7 & 1 \cdot 126 \\ 7+0 & 126 \cdot 1 \\ 7+6 & 126 \cdot 1 \\ 7+7 & 126 \cdot 126 \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ 6 & 2 \\ 7 & 252 \\ 12 & 1 \\ 13 & 252 \\ 14 & 15876 \end{bmatrix}$$

To the end of abstracting the notion showcased in the example, suppose that there exist t associative binary operations $\oplus_j : \mathbb{Z}_{\geq 0} \times \mathbb{Z}_{\geq 0} \to \mathbb{Z}_{\geq 0}$ for $0 \leq j \leq t-1$. For our purposes, these operations are just + (regular addition) or · (regular multiplication) and we restrict ourselves to either $t = 2$ or $t = 3$. Furthermore, suppose that $T^i \subseteq \mathbb{Z}_{\geq 0}^t$ is an encoding of some histogram of L_i for $0 \leq i \leq n-1$. We think of T^i as an $s \times t$ matrix for some $s \in \mathbb{Z}_{\geq 0}$ and index its rows as T^i_j and its entries as T^i_{jk}. We define the binary convolution operator

$$* : \mathbb{Z}_{\geq 0}^t \times \mathbb{Z}_{\geq 0}^t \to \mathbb{Z}_{\geq 0}^t$$

that combines sequences as

$$(l_0, \dots, l_{t-1}) * (m_0, \dots, m_{t-1}) = (l_0 \oplus_0 m_0, \dots, l_{t-1} \oplus_{t-1} m_{t-1})$$

We note that $*$ is associative, since the $\oplus_j$ are. Using this operator, it is possible to combine sequences from different histograms in a sensible way. Any sequence in the histogram of L is related to the sequences in the histograms of the L_i in the following way:

$$(n_0, \dots, n_{t-1}) = \mathop{\Large *}_{\substack{0 \leq i \leq n-1 \\ j \in C_i(n_0, \dots, n_{t-1})}} T^i_j$$

where

$$C_i(n_0, \dots, n_{t-1}) = \{j \in \mathbb{Z}_{\geq 0} : \bigoplus_{i=0}^{n-1}{}_l \, T^i_{jl} = n_l \text{ for } 0 \leq l \leq t-1\}$$

A.2 Mixing layers based on MDS codes

We show how to compute the box weight histogram and the cluster histogram of a mixing layer L that is based on MDS codes. In the cipher built on top of L, we suppose that there is an S-box layer N. Let

the ordered partition Π_0 be defined by N. Moreover, we suppose that the size of the boxes of Π_0 is m. Consider

$$\mathrm{L} = \mathrm{L}_0 \times \cdots \times \mathrm{L}_{s-1} : \bigtimes_{i=0}^{s-1} \mathbb{F}_2^{km} \to \bigtimes_{i=0}^{s-1} \mathbb{F}_2^{km}.$$

Now, L defines an ordered partition Π_1. Indeed, each of the s boxes of Π_1 is the union of k boxes of Π_0 of size m. It follows that $\Pi_0 \leq \Pi_1$, N is box-aligned with respect to both Π_0 and Π_1, and L is box-aligned with respect to Π_1.

Definition 47. *A function $f : \mathbb{F}_{2^m}^k \to \mathbb{F}_{2^m}^k$ is called an MDS function if the set $\{(x, f(x)) : x \in \mathbb{F}_{2^m}^k\} \subseteq \mathbb{F}_{2^m}^{2k} \cong \mathbb{F}_2^{2km}$ is an MDS code over $\mathbb{F}_{2^m}$ of minimum distance d.*

The minimum distance d is precisely the box branch number. Henceforth, we make the assumption that the box functions L_i, with $0 \leq i \leq s-1$, are MDS functions, i.e., that have branch number $d = k+1$.

First, Proposition 19 shows how to compute the box weight histogram of L_i with respect to the subset of Π_0 consisting of the boxes indexed by $ik, \ldots, (i+1)k - 1$ (they form a partition of the input space of L_i).

Proposition 19. *Let C be an $[2k, k, k+1]$ MDS code over $\mathbb{F}_q$. The weight distribution of C is given by $A_0 = 1$, $A_w = 0$ for $1 \leq w < k+1$, and*

$$A_w = \binom{2k}{w} \sum_{i=0}^{w-k-1} (-1)^i \binom{w}{i} (q^{w-k-i} - 1)$$

for $k+1 \leq w \leq 2k$.

Proof. This is a specific case of Proposition 7.4.1 in [23]. □

As an example, in both MixColumns of Rijndael and MC of Saturnin we have $k = 4$. Using convolution, it is now possible to obtain the box weight histogram of L from those of the L_i.

Next, we show how to compute the L_i-box histogram. We put $C_{m,k}(\mathrm{w}) = |[a]_{\simeq}|$ for any $a \in \mathbb{F}_2^{km}$ with $\mathrm{w}_\Pi(a) + \mathrm{w}_\Pi(\mathrm{L}(a)) = \mathrm{w}$. In other words, $C_{m,k}(\mathrm{w})$ does not depend on the box activity pattern of a, but only on its box weight. A box activity pattern can be chosen in $\binom{2n}{k}$ Before stating the main result, we prove some lemmas.

Proposition 20. *We have $C_{m,k}(k+1) = 2^m - 1$.*

Proof. Since L_i is an MDS function, there exists a $k \times k$ matrix M such that $H = (M I_k)$ is a parity-check matrix of an MDS code of dimension k. Let $S \subseteq [0, 2k-1]$ with $|S| = k$ be a subset of the column index space. The columns of H indexed by S form a $k \times k$ sub-matrix. Since H defines an MDS code, this sub-matrix is invertible. This means that the columns of the reduced row echelon form of H indexed by S form the identity matrix I_k. By permuting the columns of the reduced row echelon form, we obtain a parity-check matrix of an equivalent code, $H' = (M' I_k)$. This defines a linear map $\mathrm{M} : \mathbb{F}_{2^m}^k \to \mathbb{F}_{2^m}^k$ given

by $\mathrm{M}(a) = M'^{\top} a$. Now, let $a \in \mathbb{F}_2^{km}$ with $\mathrm{w}_\Pi(a) + \mathrm{w}_\Pi(\mathrm{L}(a)) = k + 1$. Pick the subset S in such a way that it contains the index of one active box of $(a, \mathrm{L}(a))$ and such that the other indices correspond to $k - 1$ passive boxes. The other k active boxes are completely determined by this single active box through M. Hence, we have only $2^m - 1$ degrees of freedom. □

Proposition 21. *We have* $C_{m,k}(k + 2) = (2^m - 1)(2^m - 1 - k)$.

Proof. Let $a \in \mathbb{F}_2^{km}$ with $\mathrm{w}_\Pi(a) + \mathrm{w}_\Pi(\mathrm{L}(a)) = k + 2$. By the same argument as given in Proposition 20, we pick S in such a way that it contains the indices of two active boxes of a. There are $(2^m - 1)^2$ ways of choosing the vector a such that it is active in two boxes. Then M determines the other k boxes. Clearly $k + 1 \leq \mathrm{w}_\Pi(a) + \mathrm{w}_\Pi(\mathrm{M}(a)) \leq k + 2$ (as there are only k boxes at the output of M). We subtract the number of vectors that lead to a box weight of $k + 1$ According to Proposition 20, this number is $2^m - 1$ for a fixed position of an active box. We can choose this position in k ways. Hence, in total, we need to subtract $k(2^m - 1)$ inputs. The result readily follows. □

Proposition 22 states the main result. The L_i-box histogram follows directly from it.

Proposition 22. *For* $k + 1 \leq w \leq 2k$*, the following recurrence relation holds:*

$$C_{m,k}(\mathrm{w}) = (2^m - 1)^{w-k} - \sum_{1 \leq i \leq w-k-1} \binom{k}{i} C_{m,k}(\mathrm{w} - i)$$

Moreover, $C_{m,k}(0) = 1$.

Proof. Let $a \in \mathbb{F}_2^{km}$ with $\mathrm{w}_\Pi(a) + \mathrm{w}_\Pi(\mathrm{L}(a)) = \mathrm{w}$. By the same argument as given in Proposition 20, pick S such that it contains the indices of $w - k$ active boxes. There are $(2^m - 1)^{w-k}$ ways of choosing the vector a such that it is active in $w - k$ boxes. It follows that $k + 1 \leq \mathrm{w}_\Pi((a, \mathrm{M}(a))) \leq w$. We subtract the number of vectors that lead to a box weight of $k + i$ for $1 \leq i \leq w - k - 1$ and obtain the result. □

Again, using convolution, it is possible to obtain the cluster histogram from the L_i-box histograms.

A.3 Exhaustive search

Cluster histogram up to given box weight

Let $\mathrm{L} : \mathbb{F}_2^b \to \mathbb{F}_2^b$ be a linear transformation. Suppose that we want to determine the cluster histogram, but that it is infeasible to construct the whole histogram because L is not box-aligned nor does it have any other properties that make it easy to do so. In this case, it is still possible to construct the cluster histogram up to a given box weight. To this end, suppose that we have an algorithm similar to the Trail Search described in [32] that generates a list of vectors up to a given weight.

For a given difference $a \in \mathbb{F}_2^b$, consider the vector $(a, \mathrm{L}(a)) \in \mathbb{F}_2^{2b}$ with $\mathrm{w}_\Pi(a) + \mathrm{w}_\Pi(\mathrm{L}(a)) = \mathrm{w}$. We wish to compute $|[a]_\sim|$. Consider the vector space spanned by the basis vectors that have the same box activity pattern as a:

$$V(a) = \left\langle \bigcup_{\substack{0 \le i \le n-1 \\ \pi_\Pi(a)_i \neq 0}} \{e_{im}^b, e_{im+1}^b, \ldots, e_{(i+1)m-1}^b\} \right\rangle$$

We compute a basis for $\mathrm{L}(V(a)) \cap V(\mathrm{L}(a))$ using Zassenhaus algorithm [26]. Then

$$c = |[a]_\sim| = |\{v \in \mathrm{L}(V(a)) \cap V(\mathrm{L}(a)) : v \in [\mathrm{L}(a)]_\sim \wedge \mathrm{L}^{-1}(v) \in [a]_\sim\}|$$

and we increment $N_{\Pi,\mathrm{L}}(\mathrm{w}, c)$ by one. Once we have considered all vectors of box weight w, the values of $N_{\Pi,\mathrm{L}}(\mathrm{w}, \cdot)$ are exact.

Three-round trail search Saturnin

During the first three rounds of Saturnin, all step functions are applied, in parallel, to disjoint slices. Since we are interested in the tail of the differential or linear trail weight histogram, we may limit our search for trails to a single slice. The cipher applies sixteen S-boxes to a slice and we write Π for the corresponding partition. Clearly, an activity pattern with respect to Π can be encoded as a vector of sixteen bits. We represent this vector as the following 4×4 binary matrix:

$$\begin{bmatrix} \pi_\Pi(a)_3 & \pi_\Pi(a)_2 & \pi_\Pi(a)_1 & \pi_\Pi(a)_0 \\ \pi_\Pi(a)_6 & \pi_\Pi(a)_5 & \pi_\Pi(a)_4 & \pi_\Pi(a)_7 \\ \pi_\Pi(a)_9 & \pi_\Pi(a)_8 & \pi_\Pi(a)_{11} & \pi_\Pi(a)_{10} \\ \pi_\Pi(a)_{12} & \pi_\Pi(a)_{15} & \pi_\Pi(a)_{14} & \pi_\Pi(a)_{13} \end{bmatrix}$$

In this representation, each matrix row corresponds to the activity pattern of differences or linear masks at the input of a single MC in the first mixing layer. Similarly, a matrix column corresponds to the activity pattern of differences or linear masks at the output of a single MC in the second mixing layer. This allows us to compute candidate activity patterns for which the weight of any differential or linear trail that contains a difference or linear mask contained in that pattern does not exceed an upper bound on the differential or linear trail weight that we set beforehand. Indeed, we generate all possible 4×4 binary matrices, encoding all possible activity patterns. Each matrix row and each matrix column has a bit weight, which corresponds to the box weight of the differences and linear masks at the input or output of a single MC. Since MC defines an MDS code of minimum distance 5, the differences or linear masks associated with a matrix row of bit weight w contribute at least $5 - \mathrm{w}$ to the differential or linear trail weight. A similar argument can be given for the columns. This gives a lower bound on the actual differential or linear trail weight of any trail comprising a difference or linear mask contained in that activity pattern. We determine whether this lower bound is smaller than or equal to the upper bound that we set. The

result is a collection of candidate activity patterns, the lower bound of which does not exceed our fixed upper bound. To the end of determining the actual trail weights, we first switch back to the following sequential representation as this makes it easier to apply the step functions:

$$\begin{bmatrix} r_{\Pi}(a)_0 & r_{\Pi}(a)_1 & r_{\Pi}(a)_2 & r_{\Pi}(a)_3 \\ r_{\Pi}(a)_4 & r_{\Pi}(a)_5 & r_{\Pi}(a)_6 & r_{\Pi}(a)_7 \\ r_{\Pi}(a)_8 & r_{\Pi}(a)_9 & r_{\Pi}(a)_{10} & r_{\Pi}(a)_{11} \\ r_{\Pi}(a)_{12} & r_{\Pi}(a)_{13} & r_{\Pi}(a)_{14} & r_{\Pi}(a)_{15} \end{bmatrix}$$

Using convolution and exhaustive search within the candidate activity patterns, we are able to find all the differential and linear trails within that slice up to a given weight.

B Minimal sum-of-product forms

We have used the Espresso algorithm to get a minimal sum-of-products (SOP) form of the three ciphers below. Note that addition denotes 'or', multiplication denotes 'and', and an overline denotes negation.

Xoodoo:

$$\begin{aligned} Y_0 &= X_0X_1 + X_0\overline{X_1X_2} + \overline{X_0X_1}X_2 \\ Y_1 &= X_1X_2 + X_0\overline{X_1X_2} + \overline{X_0}X_1\overline{X_2} \\ Y_2 &= X_0X_2 + \overline{X_0X_1}X_2 + \overline{X_0}X_1\overline{X_2} \end{aligned}$$

Saturnin:

$$\begin{aligned} Y_0 &= X_0\overline{X_1X_2X_3} + X_0\overline{X_2}X_3 + \overline{X_0}X_1\overline{X_2X_3} + X_1X_2X_3 + \overline{X_1}X_2\overline{X_3} \\ Y_1 &= X_0\overline{X_1}X_2 + \overline{X_0X_1X_2}X_3 + \overline{X_1}X_2\overline{X_3} + \overline{X_0}X_1 \\ Y_2 &= \overline{X_0}X_1X_2X_3 + \overline{X_0}X_2\overline{X_3} + \overline{X_0}X_1\overline{X_2X_3} + X_0X_1 \\ Y_3 &= X_0\overline{X_1X_2X_3} + \overline{X_0}X_1\overline{X_2X_3} + X_2X_3 + X_1X_2 \end{aligned}$$

Spongent:

$$\begin{aligned} Y_0 &= X_0X_1\overline{X_2}X_3 + X_0\overline{X_1X_2}X_3 + X_0X_1\overline{X_2X_3} + \overline{X_0}X_1X_2X_3 + \overline{X_1}X_2\overline{X_3} + \overline{X_0X_1X_2} \\ Y_1 &= X_0X_1\overline{X_2}X_3 + X_0\overline{X_1}X_2X_3 + \overline{X_0}X_1X_2 + X_0\overline{X_1X_2X_3} + \overline{X_0X_1X_2} + X_1X_2X_3 \\ Y_2 &= X_0\overline{X_1X_2}X_3 + \overline{X_0X_1}X_2\overline{X_3} + X_0X_1X_2\overline{X_3} + \overline{X_0X_2}X_3 + X_1X_2X_3 + X_0\overline{X_1X_2X_3} \\ Y_3 &= X_0X_1\overline{X_2X_3} + \overline{X_0}X_1X_2X_3 + X_0\overline{X_1}X_2X_3 + \overline{X_0X_1}X_2\overline{X_3} + X_0X_1X_2\overline{X_3} + \overline{X_0X_2}X_3 \\ &\quad + X_0\overline{X_1X_2X_3} \end{aligned}$$

From De Morgan's laws, it follows that $XY + ZW = \overline{\overline{XY} \cdot \overline{ZW}}$. In other words, the circuits of depth two that can be derived from the SOPs above, consisting of a layer of (possibly multi-input) and gates, followed by a layer of (possibly multi-input) or gates, can be converted into a circuit of depth two in which each layer consists of (possibly multi-input) nand gates.

C Estimating the number of trails with 25 active S-boxes in 4-round Saturnin

A trail over 4 rounds of Saturnin is a trail in a hyperbox that has superboxes as S-boxes. This trail is active in 5 superboxes and has 5 active boxes in each superbox.

There are $\binom{8}{5} = 56$ ways to select the 5 active superboxes from the 8 superboxes.

In the central mixing layer there can be $x \in \{1, 2, 3, 4\}$ MC instances active.

In each active MC, we have one degree of freedom as the choice of a single difference among the 5 active ones fixes the four others. This gives 15^x choices for the middle differences.

Each active superbox now has x active boxes at its input (or output), and shall have $5 - x$ active boxes at its output (or input). Consider the case $x = 1$. For a given choice of the middle difference, the difference at the input (or output) of a single S-box is fixed. For differential trails, the number of compatible output differences depends on the concrete output difference but ranges from 6 to 8 with an average of exactly 7. We think it is reasonable in this estimation to approximate this by exactly 7. For linear trails, the number of input masks compatible with a given output mask is always 10.

So given a choice of the intermediate difference, there are 7^5 differential trail cores and 10^5 linear trail cores.

This gives in total $56 \times 15 \times 7^5$ differential trail cores and $56 \times 15 \times 10^5$ Each of these trail cores has in total 20 active S-boxes in the outer S-box layers. Every difference at the inside of such an active S-box has 7 compatible differences at the outside. It follows that there are in total $56 \times 15 \times 7^5 \times 7^{20} \approx 2^{80}$ such differential trails per hyperbox and as there are 4 hyperboxes, this totals to 2^{82}.

Every mask at the inside of such an active S-box has 10 compatible masks at the outside. It follows that there are in total $56 \times 15 \times 10^5 \times 10^{20} \leq 2^{92.5}$ such linear trails per hyperbox and as there are 4 hyperboxes, this totals to $2^{94.5}$.

These are only the trails with a single active MC in the middle mixing layer. We do not count the other classes as their analysis is more involved and the final total number is much less. Hence this class dominates the total number.

D Known trail bounds for up to 12 rounds

In Table 4.6 we list the trail bounds for our four ciphers for up to 12 rounds. For AES the numbers are based on the existence of periodic trails with period 4 where the profile of the number of active S-boxes is (1, 4, 16, 4) and the fact that the minimum weight for the AES S-box is 6. For Saturnin the numbers

are based on the existence of periodic trails with period 8 where the profile of the number of active S-boxes is (1, 4, 16, 4, 16, 64, 16, 4) and the fact that the minimum weight for the Saturnin S-box is 2.

E Why Xoodoo is not aligned

In this appendix, we provide a computer-assisted proof that Xoodoo is not aligned.

Let us assume that we can factor the linear layer of Xoodoo into $L = \pi \circ M$ with M operating on non-trivial superboxes. We can identify the input bits of M that lie in the same superbox with the two following rules:

1. The output bits of L in the same box (column) depend on input bits from the same superbox;
2. Any two output bits that depend on the same input bit must also depend on input bits from the same superbox.

Therefore, we construct a bipartite graph with the 128 output boxes on one side and the 384 input bits on the other side, with edges connecting an output box to the input bits that it depends on. We explicitly constructed this graph (see Figure 4.11) and checked that it is connected. This contradicts the assumption that M operates on non-trivial superboxes.

```
def buildGraph():
    G = Graph()
    for x in range(4):
        for z in range(32):
            G.add_vertex("out-{0}-{1}".format(x, z))
            for y in range(3):
                G.add_vertex("in-{0}-{1}-{2}".format(x, y, z))
    for x in range(4):
        for z in range(32):
            out = "out-{0}-{1}".format(x, z)
            G.add_edge(out, "in-{0}-{1}-{2}".format(x, 0, z))
            G.add_edge(out, "in-{0}-{1}-{2}".format((x+3)%4, 0, (z+27)%32))
            G.add_edge(out, "in-{0}-{1}-{2}".format((x+3)%4, 0, (z+18)%32))
            G.add_edge(out, "in-{0}-{1}-{2}".format((x+3)%4, 1, (z+26)%32))
            G.add_edge(out, "in-{0}-{1}-{2}".format((x+3)%4, 1, (z+17)%32))
            G.add_edge(out, "in-{0}-{1}-{2}".format((x+1)%4, 2, (z+19)%32))
            G.add_edge(out, "in-{0}-{1}-{2}".format((x+1)%4, 2, (z+10)%32))

            G.add_edge(out, "in-{0}-{1}-{2}".format((x+3)%4, 1, (z+31)%32))
            G.add_edge(out, "in-{0}-{1}-{2}".format((x+2)%4, 0, (z+27)%32))
            G.add_edge(out, "in-{0}-{1}-{2}".format((x+2)%4, 0, (z+18)%32))
            G.add_edge(out, "in-{0}-{1}-{2}".format((x+2)%4, 1, (z+26)%32))
            G.add_edge(out, "in-{0}-{1}-{2}".format((x+2)%4, 1, (z+17)%32))
            G.add_edge(out, "in-{0}-{1}-{2}".format((x+0)%4, 2, (z+19)%32))
            G.add_edge(out, "in-{0}-{1}-{2}".format((x+0)%4, 2, (z+10)%32))

            G.add_edge(out, "in-{0}-{1}-{2}".format((x+2)%4, 2, (z+13)%32))
            G.add_edge(out, "in-{0}-{1}-{2}".format((x+3)%4, 0, (z+16)%32))
            G.add_edge(out, "in-{0}-{1}-{2}".format((x+3)%4, 0, (z+ 7)%32))
            G.add_edge(out, "in-{0}-{1}-{2}".format((x+3)%4, 1, (z+15)%32))
            G.add_edge(out, "in-{0}-{1}-{2}".format((x+3)%4, 1, (z+ 6)%32))
            G.add_edge(out, "in-{0}-{1}-{2}".format((x+1)%4, 2, (z+ 8)%32))
            G.add_edge(out, "in-{0}-{1}-{2}".format((x+1)%4, 2, (z+31)%32))
    return G

G = buildGraph()
G.is_connected()
```

Figure 4.11: Sage code to construct the graph detailed in the text and to check its connectivity.

Table 4.1: S-box computational cost comparison.

cipher	max DP/LP	ref.	operations in $\mathbb{F}_2$ # operations xor	and/or	not	2-layer nand circuit # nand gates per # inputs 2-in	3-in	4-in	5-in	6-in	7-in	totals gates	inp.
Rijndael	2^{-6}	[11, 37]	81	32	4				?				
Saturnin	2^{-2}	[13]	6	6	-	4	5	6	1	-	-	16	52
Spongent	2^{-2}			?		-	6	8	-	3	1	18	75
Xoodoo	2^{-2}	[16]	3	3	3	3	6	-	-	-	-	9	24

Table 4.2: The cost of a round in cycles per byte on the ARM Cortex-M4.

Cipher	# cycles/byte
Rijndael [39]	10.0
Saturnin [12]	2.7
Spongent	?
Xoodoo [6]	1.1

Table 4.3: The cluster histograms of Rijndael and Saturnin.

	$N \times C_{m,n}$		
$\widetilde{w}$	Rijndael superbox $m = 8, n = 4$	Saturnin superbox $m = 4, n = 4$	Saturnin hyperbox $m = 16, n = 4$
5	(56×255)	(56×15)	(56×65535)
6	$(28 \times 64005))$	(28×165)	(28×4294574085)
7	(8×16323825)	(8×2625)	$(8 \times 281444913315825)$
8	(1×4162570275)	(1×39075)	$(1 \times 18444923941512806 75)$

Table 4.4: The cluster histogram of SpongentMix of Spongent.

$\widetilde{w}$	$N \times C$
2	(16×1)
3	(48×1)
4	(32×1) (36×7)
5	(8×1) (48×25)
6	(12×79) (16×265)
7	(8×2161)
8	(1×41503)

Table 4.5: Partial cluster histogram (up to translation equivalence) of Xoodoo.

$\widetilde{w}$	$N \times C$
4	(3×1)
7	(24×1)
8	(600×1)
9	(2×1)
10	(442×1)
11	(10062×1)
12	(80218×1)
13	(11676×1)
14	(228531×1) (3×2)
15	(2107864×1) (90×2)
16	(8447176×1) (702×2)
⋮	⋮

Table 4.6: Known lower bounds for weights of differential trails.

		number of rounds								
cipher	source	1	2	3	4	5	6	7	8	12
Spongent	[9]	2	4	8	12	≥ 20	28	-	-	≥ 72
AES	[20]	6	30	54	150	≥ 156	≥ 180	≥ 204	≥ 300	≥ 450
Saturnin	[13]	2	10	19	≥ 50	≥ 58	≥ 90	≥ 122	250	≥ 300
Xoodoo	[15]	2	8	36	[74, 80]	≥ 94	≥ 104	≥ 110	≥ 148	≥ 222

References

1. I. 29192-5:2016. *Information technology — Security techniques — Lightweight cryptography — Part 5: Hash-functions*. 1st ed. International Organization for Standardization, Vernier, Geneva, Switzerland, 2016. URL: https://www.iso.org/standard/67173.html.
2. S. Banik, S. K. Pandey, T. Peyrin, Y. Sasaki, S. M. Sim, and Y. Todo. "GIFT: A Small Present - Towards Reaching the Limit of Lightweight Encryption". In: *Cryptographic Hardware and Embedded Systems - CHES 2017, Proceedings*. Ed. by W. Fischer and N. Homma. Vol. 10529. Lecture Notes in Computer Science. Springer, 2017, pp. 321–345. DOI: 10.1007/978-3-319-66787-4_16.
3. R. Beaulieu, D. Shors, J. Smith, S. Treatman-Clark, B. Weeks, and L. Wingers. "The SIMON and SPECK Families of Lightweight Block Ciphers". *IACR Cryptol. ePrint Arch.* 2013, 2013, p. 404.
4. D. J. Bernstein. *Cache-timing attacks on AES*. Technical report. 2005.
5. D. J. Bernstein. "The Salsa20 Family of Stream Ciphers". In: *New Stream Cipher Designs - The eSTREAM Finalists*. Ed. by M. J. B. Robshaw and O. Billet. Vol. 4986. Lecture Notes in Computer Science. Springer, 2008, pp. 84–97. DOI: 10.1007/978-3-540-68351-3_8.
6. G. Bertoni, J. Daemen, S. Hoffert, M. Peeters, G. Van Assche, and R. Van Keer. *Extended Keccak Code Package*. https://github.com/XKCP/XKCP. Accessed: 02 October 2020.
7. G. Bertoni, J. Daemen, M. Peeters, and G. Van Assche. *The Keccak reference*. 2011.
8. E. Biham and A. Shamir. "Differential Cryptanalysis of DES-like Cryptosystems". In: *Advances in Cryptology - CRYPTO '90, 10th Annual International Cryptology Conference, Santa Barbara, California, USA, August 11-15, 1990, Proceedings*. Ed. by A. Menezes and S. A. Vanstone. Vol. 537. Lecture Notes in Computer Science. Springer, 1990, pp. 2–21. DOI: 10.1007/3-540-38424-3_1. URL: https://doi.org/10.1007/3-540-38424-3%5C_1.
9. A. Bogdanov, M. Knezevic, G. Leander, D. Toz, K. Varici, and I. Verbauwhede. "SPONGENT: The Design Space of Lightweight Cryptographic Hashing". *IACR Cryptol. ePrint Arch.* 2011, 2011, p. 697.
10. A. Bogdanov, L. R. Knudsen, G. Leander, C. Paar, A. Poschmann, M. J. B. Robshaw, Y. Seurin, and C. Vikkelsoe. "PRESENT: An Ultra-Lightweight Block Cipher". In: *Cryptographic Hardware and Embedded Systems - CHES 2007, Proceedings*. Ed. by P. Paillier and I. Verbauwhede. Vol. 4727. Lecture Notes in Computer Science. Springer, 2007, pp. 450–466. DOI: 10.1007/978-3-540-74735-2_31.
11. J. Boyar and R. Peralta. "A New Combinational Logic Minimization Technique with Applications to Cryptology". In: *Experimental Algorithms, 9th International Symposium, SEA 2010, Proceedings*. Ed. by P. Festa. Vol. 6049. Lecture Notes in Computer Science. Springer, 2010, pp. 178–189. DOI: 10.1007/978-3-642-13193-6_16.

12. A. Canteaut, S. Duval, G. Leurent, M. Naya-Plasencia, L. Perrin, T. Pornin, and A. Schrottenloher. *Saturnin Implementations*. https://project.inria.fr/saturnin/files/2019/05/saturnin.zip. Accessed: 01 October 2020.

13. A. Canteaut, S. Duval, G. Leurent, M. Naya-Plasencia, L. Perrin, T. Pornin, and A. Schrottenloher. "Saturnin: a suite of lightweight symmetric algorithms for post-quantum security". *IACR Transactions on Symmetric Cryptology* 2020:S1, 2020, pp. 160–207. DOI: 10.13154/tosc.v2020.iS1.160-207.

14. J. Daemen. *Cipher and hash function design, strategies based on linear and differential cryptanalysis, PhD Thesis*. K.U.Leuven, 1995.

15. J. Daemen, S. Hoffert, M. Peeters, G. Van Assche, and R. Van Keer. "Xoodyak, a lightweight cryptographic scheme". *IACR Trans. Symmetric Cryptol.* 2020:S1, 2020, pp. 60–87. DOI: 10.13154/tosc.v2020.iS1.60-87.

16. J. Daemen, S. Hoffert, G. Van Assche, and R. Van Keer. "The design of Xoodoo and Xoofff". *IACR Trans. Symmetric Cryptol.* 2018:4, 2018, pp. 1–38. DOI: 10.13154/tosc.v2018.i4.1-38.

17. J. Daemen, S. Hoffert, G. Van Assche, and R. Van Keer. *XooTools*. https://github.com/KeccakTeam/Xoodoo/tree/master/XooTools. Accessed: 02 October 2020. 2018.

18. J. Daemen, M. Peeters, G. Van Assche, and G. Bertoni. *On alignment in Keccak*. Note. 2011.

19. J. Daemen and V. Rijmen. "Plateau characteristics". *IET Information Security* 1:1, 2007, pp. 11–17. DOI: 10.1049/iet-ifs:20060099.

20. J. Daemen and V. Rijmen. *The Design of Rijndael - The Advanced Encryption Standard (AES), Second Edition*. Information Security and Cryptography. Springer, 2020. ISBN: 978-3-662-60768-8. DOI: 10.1007/978-3-662-60769-5.

21. J. Daemen and V. Rijmen. "The Wide Trail Design Strategy". In: *Cryptography and Coding, Proceedings*. Ed. by B. Honary. 2001. DOI: 10.1007/3-540-45325-3_20.

22. J. Daemen and V. Rijmen. "Understanding Two-Round Differentials in AES". In: *Security and Cryptography for Networks, 5th International Conference, SCN 2006, Proceedings*. Ed. by R. D. Prisco and M. Yung. Vol. 4116. Lecture Notes in Computer Science. Springer, 2006, pp. 78–94. DOI: 10.1007/11832072_6.

23. W. C. Huffman and V. Pless. *Fundamentals of Error-Correcting Codes*. Cambridge University Press, 2003. ISBN: 978-0-51180707-7. DOI: 10.1017/CBO9780511807077.

24. L. R. Knudsen. "Truncated and Higher Order Differentials". In: *FSE 1994*. Ed. by B. Preneel. Vol. 1008. Lecture Notes in Computer Science. Springer, 1994, pp. 196–211. DOI: 10.1007/3-540-60590-8_16. URL: https://doi.org/10.1007/3-540-60590-8%5C_16.

25. T. Kranz, G. Leander, K. Stoffelen, and F. Wiemer. "Shorter Linear Straight-Line Programs for MDS Matrices". *IACR Trans. Symmetric Cryptol.* 2017:4, 2017, pp. 188–211. DOI: 10.13154/tosc.v2017.i4.188-211.

26. Künzer, Martin, Tentler, and Wahrheit. *Zassenhaus-Algorithmus*. `https://mo.mathematik.uni-stuttgart.de/inhalt/beispiel/beispiel1105/`. Accessed: 30 June 2020.

27. B. Lambin, G. Leander, and P. Neumann. "Pitfalls and Shortcomings for Decompositions and Alignment". In: *Advances in Cryptology - EUROCRYPT 2023 - 42nd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Lyon, France, April 23-27, 2023, Proceedings, Part IV*. Ed. by C. Hazay and M. Stam. Vol. 14007. Lecture Notes in Computer Science. Springer, 2023, pp. 318–347. DOI: `10.1007/978-3-031-30634-1\_11`. URL: `https://doi.org/10.1007/978-3-031-30634-1%5C_11`.

28. G. Leander and A. Poschmann. "On the Classification of 4 Bit S-Boxes". In: *Arithmetic of Finite Fields, First International Workshop, WAIFI 2007, Proceedings*. Ed. by C. Carlet and B. Sunar. Vol. 4547. Lecture Notes in Computer Science. Springer, 2007, pp. 159–176. DOI: `10.1007/978-3-540-73074-3\_13`.

29. C. Li and Q. Wang. "Design of Lightweight Linear Diffusion Layers from Near-MDS Matrices". *IACR Trans. Symmetric Cryptol.* 2017:1, 2017, pp. 129–155. DOI: `10.13154/tosc.v2017.i1.129-155`.

30. M. Matsui. "Linear Cryptanalysis Method for DES Cipher". In: *Advances in Cryptology - EUROCRYPT '93, Proceedings*. Ed. by T. Helleseth. DOI: `10.1007/3-540-48285-7\_33`.

31. P. C. McGeer, J. V. Sanghavi, R. K. Brayton, and A. L. Sangiovanni-Vincentelli. "ESPRESSO-SIGNATURE: a new exact minimizer for logic functions". *IEEE Trans. Very Large Scale Integr. Syst.* 1:4, 1993, pp. 432–440. DOI: `10.1109/92.250190`.

32. S. Mella, J. Daemen, and G. Van Assche. "New techniques for trail bounds and application to differential trails in Keccak". *IACR Trans. Symmetric Cryptol.* 2017:1, 2017, pp. 329–357. DOI: `10.13154/tosc.v2017.i1.329-357`.

33. NIST. *Federal Information Processing Standard 197, Advanced Encryption Standard (AES)*. 2001.

34. NIST. *Federal Information Processing Standard 202, SHA-3 Standard: Permutation-Based Hash and Extendable-Output Functions*. 2015.

35. K. Nyberg. "Differentially Uniform Mappings for Cryptography". In: *Advances in Cryptology - EUROCRYPT '93, Proceedings*. Ed. by T. Helleseth. Vol. 765. Lecture Notes in Computer Science. Springer, 1993, pp. 55–64. DOI: `10.1007/3-540-48285-7\_6`.

36. S. Park, S. H. Sung, S. Chee, E. Yoon, and J. Lim. "On the Security of Rijndael-Like Structures against Differential and Linear Cryptanalysis". In: *Advances in Cryptology - ASIACRYPT 2002, Proceedings*. Ed. by Y. Zheng. Vol. 2501. Lecture Notes in Computer Science. Springer, 2002, pp. 176–191. DOI: `10.1007/3-540-36178-2\_11`.

37. P. Schwabe and K. Stoffelen. “All the AES You Need on Cortex-M3 and M4”. In: *Selected Areas in Cryptography - SAC 2016 - 23rd International Conference, Revised Selected Papers*. Ed. by R. Avanzi and H. M. Heys. Vol. 10532. Lecture Notes in Computer Science. Springer, 2016, pp. 180–194. DOI: 10.1007/978-3-319-69453-5_10.

38. M. R. M. Shamsabad and S. M. Dehnavi. “Dynamic MDS diffusion layers with efficient software implementation”. *Int. J. Appl. Cryptogr.* 4:1, 2020, pp. 36–44. DOI: 10.1504/IJACT.2020.107164.

39. K. Stoffelen. *AES Implementations*. https://github.com/Ko-/aes-armcortexm. Accessed: 01 October 2020.

5 Weak Subtweakeys in SKINNY

Daniël Kuijsters[1], Denise Verbakel[1], Joan Daemen[1]

1 – Radboud University, The Netherlands

My contributions. This chapter is based on work accepted at Indocrypt 2022. As part of her bachelor's thesis, Denise extended the software from the previous chapter to compute data and histograms for the SKINNY cipher. The results she obtained were unexpected, prompting us to investigate and seek an explanation, which we documented. I was responsible for the entirety of this chapter, including reimplementing the software to verify the results, adding additional features, and writing the text.

Abstract. Lightweight cryptography is characterized by the need for low implementation cost, while still providing sufficient security. This requires careful analysis of building blocks and their composition.

SKINNY is an ISO/IEC standardized family of tweakable block ciphers and a reduced-round variant of it is used in the NIST lightweight cryptography standardization process finalist Romulus. We present non-trivial linear approximations of two-round SKINNY that have correlation one or minus one and that hold for a large fraction of all round tweakeys. Moreover, we show how these could have been avoided.

5.1 Introduction

In 2018, NIST initiated a process for the standardization of *lightweight cryptography* [13], i.e., cryptography that is suitable for use in constrained environments. A typical cryptographic primitive is built by composing a relatively simple round function with itself a number of times. To choose this number of rounds, a trade-off is made between the security margin and the performance.

One of the finalists in this standardization process is the Romulus [7] scheme for authenticated encryption with associated data. This scheme is based on a reduced-round variant of the lightweight tweakable block cipher SKINNY [1].

Two of the most important techniques for the analysis of symmetric primitives are differential [2] and linear cryptanalysis [11]. To reason about the security against these attacks, the designers of SKINNY have computed lower bounds on the number of *active* S-boxes in linear and differential trails. However, at the end of Section 4.1 of [1] they write:

> The above bounds are for single characteristic, thus it will be interesting to take a look at differentials and linear hulls. Being a rather complex task, we leave this as future work.

Building on the work of [3], [14] investigated clustering of two-round trails in SKINNY and in this paper we report and explain its most striking finding.

By examination of two rounds, we argue why it is sensible to look at the substructure that consists of a double S-box with a subtweakey addition in between. We study this double S-box structure both from an algebraic point of view and a statistical point of view. We found that for some subtweakeys there are non-trivial *perfect* linear approximations, i.e., that have correlation one or minus one. We present them in this paper together with their constituent linear trails. For both the version of SKINNY that uses the 4-bit S-box and the version that uses the 8-bit S-box, we present one non-trivial perfect linear approximation of the double S-box structure that holds for $1/4$ of all subtweakeys and four non-trivial perfect linear approximations that each hold for $1/16$ of all subtweakeys. In total, $1/4$ of the subtweakeys is *weak*, i.e., it has an associated non-trivial perfect linear approximation. The linear approximations of the double S-box structure can be extended to linear approximations of the full two rounds of SKINNY. From the fact that the double S-box structure appears in four different locations, it follows that $1 - (3/4)^4 \approx 68\%$ of the round tweakeys is weak, i.e., two rounds have a non-trivial perfect linear approximation.

Despite requiring more resources to compute, this shows that for many round tweakeys two rounds are weaker than a single round. Moreover, this also shows that the bounds on the squared correlations of linear approximations that are based on counting the number of active S-boxes in linear trails may not be readily assumed.

We conclude by showing how this undesired property could have easily been avoided by composing the S-box with a permutation of its output bits, which has a negligible impact on the implementation cost.

5.1.1 Outline and contributions

In Section 5.2 we remind the reader of the parts of the SKINNY block cipher specification that are relevant to our analysis. We argue why it is reasonable to study the double S-box structure and explore its algebraic properties. Section 5.3 serves as a reminder for the reader of the relevant statistical analysis tools of linear cryptanalysis. Section 5.4 presents our findings from the study of the linear trails of the double S-box structure. We show how the problem could have been avoided in Section 5.5. Finally, we state the main message behind our findings in Section 5.6.

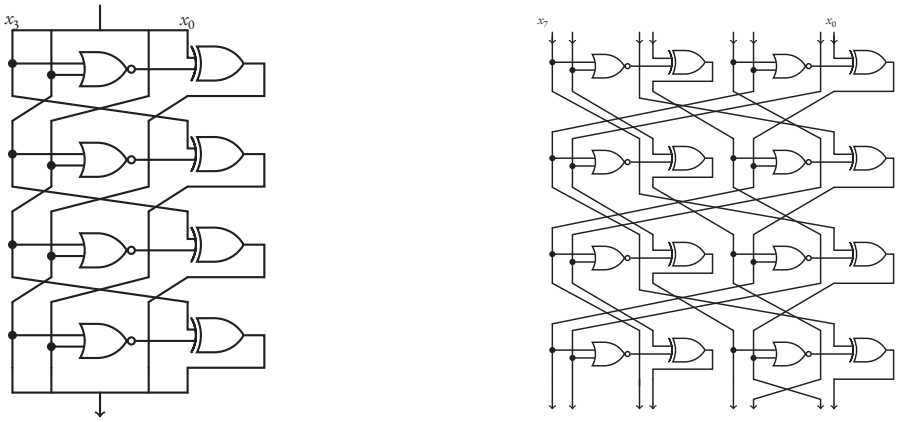
5.2 The SKINNY family of block ciphers

SKINNY [1] is a family of tweakable block ciphers. A member of the SKINNY family is denoted by SKINNY-b-t, where b denotes the block size and t denotes the size of the tweakey [9]. The block size b is equal to 64 bits or 128 bits. The tweakey t is b, $2b$, or $3b$ bits.

The AES-like [6] data path of the SKINNY block cipher is the repeated application of a round function on a representation of the state as a four by four array of m-bit vectors, where m is either four or eight.

Pairs (i, j) comprising a row index i and column index j with $0 \leq i, j \leq 3$ are used to index into the state array. For example, $(0, 0)$ refers to the entry in the top left and $(3, 3)$ to the entry in the bottom right. The m-bit entries $x^{(i,j)}$ are of the form $(x_{m-1}^{(i,j)}, \ldots, x_0^{(i,j)})$.

The round function consists of the following steps in sequence: SubCells, AddConstants, AddRoundTweakey, ShiftRows, and MixColumns.



(a) 4-bit S-box S_4. (b) 8-bit S-box S_8.

Figure 5.1: Circuit-level representation of S_4 and S_8. (Figure adapted from [8].)

Figure 5.1 shows the circuit-level view of the S-boxes that are used in the SubCells step of SKINNY.

The block matrix that is used in the MixColumns step is equal to

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

where 0 denotes the zero matrix of size $m \times m$ and 1 denotes the identity matrix of size m. Each of the four columns of the state is multiplied by M in parallel.

The composition of two rounds is depicted in Figure 5.2.

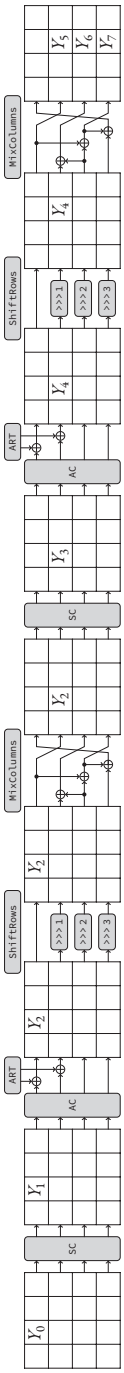


Figure 5.2: Two-round SKINNY. (Figure adapted from [8].)

Consider the entry of the state at position $(0,1)$ in Figure 5.2. It is of the form $Y_0 = x^{(0,1)}$. This expression propagates through the step functions of two rounds and leads to the following intermediate expressions:

$$\begin{aligned} Y_1 &= \mathrm{S}_m(x^{(0,1)}) \\ Y_2 &= \mathrm{S}_m(x^{(0,1)}) + k^{(0)} \\ Y_3 &= \mathrm{S}_m(\mathrm{S}_m(x^{(0,1)}) + k^{(0)}) \\ Y_4 &= \mathrm{S}_m(\mathrm{S}_m(x^{(0,1)}) + k^{(0)}) + k^{(1)} \\ Y_5 + Y_6 + Y_7 &= \mathrm{S}_m(\mathrm{S}_m(x^{(0,1)}) + k^{(0)}) + k^{(1)}\,, \end{aligned}$$

Here, $k^{(0)}$ and $k^{(1)}$ are subtweakeys, which are linear expressions in the cipher key and tweak bits (assuming that the tweakey does not consist entirely of cipher key bits). These linear expressions depend on the round number, but they are known to the attacker. The tweak can be chosen by the attacker and the cipher key is unknown to the attacker. By choosing the tweak, the attacker can attain all values of $k^{(0)}$ and $k^{(1)}$ for a given cipher key.

The final expression shows that the sum of certain triples of state entries at the output of the second round is equal to the application of two S-boxes and subtweakey additions to a single entry of the input to the first round. The second subtweakey addition does not have an important influence on the statistical properties of this expression, so we remove it and turn our attention to the properties of the function

$$\mathrm{D}_{m,k} = \mathrm{S}_m \circ T_{m,k} \circ \mathrm{S}_m\,,$$

where $T_{m,k}$ is defined by $x \mapsto x + k$ for $x \in \mathbb{F}_2^m$. We will refer to $\mathrm{D}_{m,k}$ as the *double S-box structure*.

For reasons of simplicity, we study SKINNY-64-t, i.e., the version with 4-bit S-boxes. However, our results can be extended to the case of 8-bit S-boxes as well.

By concatenating two copies of the 4-bit S-box circuit with a subtweakey addition layer in between we obtain the circuit-level view of $\mathrm{D}_{4,k}$ that is depicted in Figure 5.3. Consider the input x_1. It passes through an XOR gate, the subtweakey addition layer, and finally through a second XOR gate before being routed to the third component of the output of $\mathrm{D}_{4,k}$. If $k_3 = k_2 = 0$, then the XOR gates cancel each other out and the third component of $\mathrm{D}_{4,k}$ is equal to $x_1 + k_0$. This observation does not depend on the value of k_1.

Let us now derive this same result in an algebraic way. Of course, we could compute the algebraic expression for $\mathrm{D}_{4,k}$ directly, but it is more insightful to study the S-box and its inverse.

The 4-bit S-box is of the form

$$\mathrm{S}_4 = \mathrm{N}_4 \circ \mathrm{L}_4 \circ \mathrm{N}_4 \circ \mathrm{L}_4 \circ \mathrm{N}_4 \circ \mathrm{L}_4 \circ \mathrm{N}_4$$

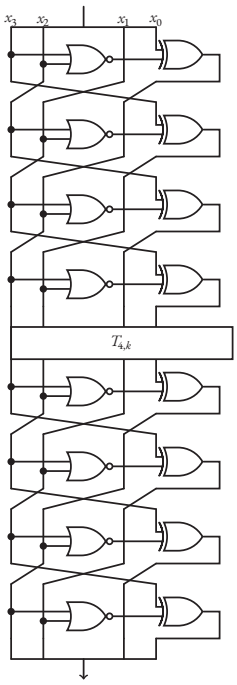


Figure 5.3: Circuit-level representation of $\mathrm{D}_{4,k}$. (Figure adapted from [8].)

where

$$\mathrm{N}_4(x_3, x_2, x_1, x_0) = (x_3, x_2, x_1, x_2x_3 + x_0 + x_2 + x_3 + 1) \quad \text{and}$$
$$\mathrm{L}_4(x_3, x_2, x_1, x_0) = (x_2, x_1, x_0, x_3)\,.$$

It follows that $\mathrm{S}_4 = (\mathrm{S}_4^{(3)}, \mathrm{S}_4^{(2)}, \mathrm{S}_4^{(1)}, \mathrm{S}_4^{(0)})$ where

$$\begin{aligned}
\mathrm{S}_4^{(3)} &= x_2x_3 + x_0 + x_2 + x_3 + 1\\
\mathrm{S}_4^{(2)} &= x_1x_2 + x_1 + x_2 + x_3 + 1\\
\mathrm{S}_4^{(1)} &= x_1x_2x_3 + x_0x_1 + x_1x_2 + x_1x_3 + x_2x_3 + x_0 + x_3\\
\mathrm{S}_4^{(0)} &= x_0x_1x_2 + x_1x_2x_3 + x_0x_1 + x_0x_2 + x_0x_3 + x_1x_3 + x_1 + x_2 + x_3
\end{aligned}$$

The S-box has a generalized Feistel structure [12]. Therefore, it is not difficult to deduce that the inverse of $T_{4,k} \circ \mathrm{S}_4$ is of the form

$$I_{4,k} = (T_{4,k} \circ \mathrm{S}_4)^{-1} = \mathrm{N}_4 \circ \mathrm{R}_4 \circ \mathrm{N}_4 \circ \mathrm{R}_4 \circ \mathrm{N}_4 \circ \mathrm{R}_4 \circ \mathrm{N}_4 \circ T_{4,k}\,,$$

where $R_4(x_3, x_2, x_1, x_0) = (x_0, x_3, x_2, x_1)$. It follows that $I_{4,k}$ is of the form $(I_{4,k}^{(3)}, I_{4,k}^{(2)}, I_{4,k}^{(1)}, I_{4,k}^{(0)})$ where

$$\begin{aligned}
I_{4,k}^{(3)} &= x_1x_2x_3 + x_0x_1 + x_0x_3 + x_1x_2(k_3 + 1) + x_1x_3(k_2 + 1) + x_2x_3k_1 \\
&\quad + x_1(k_2k_3 + k_0 + k_2 + k_3) + x_2(k_1k_3 + k_1 + 1) + x_3(k_1k_2 + k_0 + k_1 + 1) \\
&\quad + x_0(k_1 + k_3) + k_1k_2k_3 + k_0k_1 + k_0k_3 + k_1k_2 + k_1k_3 + k_2 + k_3\,, \\
I_{4,k}^{(2)} &= x_0x_3 + x_2x_3 + x_0(k_3 + 1) + x_2(k_3 + 1) + x_3(k_0 + k_2) + x_1 + k_0k_3 + k_2k_3 \\
&\quad + k_0 + k_1 + k_2\,, \\
I_{4,k}^{(1)} &= x_2x_3 + x_2(k_3 + 1) + x_3(k_2 + 1) + x_0 + k_2k_3 + k_0 + k_2 + k_3 + 1\,, \\
I_{4,k}^{(0)} &= x_0x_2x_3 + x_1x_2x_3 + x_0x_2(k_3 + 1) + x_0x_3k_2 + x_1x_2k_3 + x_1x_3(k_2 + 1) \\
&\quad + x_2x_3(k_0 + k_1) + x_0x_1 + x_0(k_2k_3 + k_1 + k_2 + 1) + x_1(k_2k_3 + k_0 + k_3 + 1) \\
&\quad + x_2(k_0k_3 + k_1k_3 + k_0 + 1) + x_3(k_0k_2 + k_1k_2 + k_1) + k_0k_2k_3 + k_1k_2k_3 \\
&\quad + k_0k_1 + k_0k_2 + k_1k_3 + k_0 + k_1 + k_2 + 1\,.
\end{aligned}$$

We observe that if $k_3 = k_2 = 0$, then the component $I_{4,k}^{(1)}$ differs from $S_4^{(3)}$ by the constant k_0 for any value of k_1. This implies that $D_{4,(0,0,k_1,k_0)}^{(3)} = x_1 + k_0$.

5.3 Linear cryptanalysis

To analyze $D_{m,k}$ in more detail, we use the statistical framework of linear cryptanalysis [5, 11].

The important concept here is a *linear approximation*, i.e., an ordered pair of linear masks $(u, v) \in \mathbb{F}_2^m \times \mathbb{F}_2^m$ that determine linear combinations of output and input bits, respectively. A mask u defines a *linear functional*

$$x \mapsto u^\top x = u_0x_0 + \cdots + u_{m-1}x_{m-1}\,.$$

We measure the quality of a linear approximation with the correlation between the linear functionals defined by the masks.

Definition 48. *The (signed)* correlation *between the linear functional defined by the mask $u \in \mathbb{F}_2^m$ at the output of a function $G\colon \mathbb{F}_2^m \to \mathbb{F}_2^m$ and the linear functional defined by the mask $v \in \mathbb{F}_2^m$ at its input is defined as*

$$C_G(u, v) = \frac{1}{2^m} \sum_{x \in \mathbb{F}_2^m} (-1)^{u^\top G(x) + v^\top x}\,.$$

The $2^m \times 2^m$ matrix C_G with entries $C_G(u, v)$ is called the *correlation matrix* of the function G. We call a linear approximation with a correlation of one or minus one *perfect*.

In addition to specifying masks at the input and output of $D_{m,k}$, we may also specify intermediate masks.

Definition 49. *A sequence* $(u, v, w) \in \mathbb{F}_2^m \times \mathbb{F}_2^m \times \mathbb{F}_2^m$ *is called a* linear trail *of* $\mathrm{D}_{m,k}$ *if it satisfies the following conditions:*

1. $\mathrm{C}_{\mathrm{S}_m}(u, v) \neq 0$;
2. $\mathrm{C}_{\mathrm{S}_m}(v, w) \neq 0$.

Each of the trails contributes to the correlation of the linear approximation.

Definition 50. *The* correlation contribution *of a linear trail* (u, v, w) *over* $\mathrm{D}_{m,k}$ *equals*

$$\mathrm{C}_{\mathrm{D}_{m,k}}(u, v, w) = (-1)^{v^\top k}\, \mathrm{C}_{\mathrm{S}_m}(u, v)\, \mathrm{C}_{\mathrm{S}_m}(v, w)\,.$$

From the theory of correlation matrices [5], it follows that

$$\begin{aligned}\mathrm{C}_{\mathrm{D}_{m,k}}(u, v) &= \sum_{v \in \mathbb{F}_2^m} \mathrm{C}_{\mathrm{D}_{m,k}}(u, v, w) \\ &= \sum_{v \in \mathbb{F}_2^m} (-1)^{v^\top k}\, \mathrm{C}_{\mathrm{S}_m}(u, v)\, \mathrm{C}_{\mathrm{S}_m}(v, w)\,.\end{aligned}$$

5.4 Linear trails of the double S-box structure

We can now translate the observations from Section 5.2 into the language of linear cryptanalysis. The observations state that the linear approximation $(\texttt{1000}, \texttt{0010})$ of $\mathrm{D}_{4,(0,0,k_1,k_0)}$ is perfect for all $k_0, k_1 \in \mathbb{F}_2$.

One way of seeing this is directly from the fact that

$$\begin{aligned}(\texttt{1000})^\top \mathrm{D}_{4,(0,0,k_1,k_0)} &= \mathrm{D}^{(3)}_{4,(0,0,k_1,k_0)} \\ &= x_1 + k_0 \\ &= (\texttt{0010})^\top x + k_0\,.\end{aligned}$$

Hence, the correlation is one if k_0 is zero and minus one otherwise.

An alternative view is the following. Due to the equivalence of vectorial Boolean functions and their correlation matrices [5], equality of $\mathrm{S}_4^{(3)}$ and $I_{4,k}^{(1)}$ implies equality of row $\texttt{1000}$ of $\mathrm{C}_{\mathrm{S}_4}$ and row $\texttt{0010}$ of $\mathrm{C}_{I_{4,k}}$. The latter corresponds to column $\texttt{0010}$ of $C_{T_{4,k} \circ \mathrm{S}_4}$. These are exactly the two vectors that we need to multiply in order to compute $\mathrm{C}_{\mathrm{D}_{4,k}}(\texttt{1000}, \texttt{0010})$. Using the orthogonality relations [10], it is not difficult to show that this correlation is either one or minus one, depending on the constant difference between $\mathrm{S}_4^{(3)}$ and $I_{4,k}^{(1)}$, which only influences the sign.

In general, we have computed all the non-trivial perfect linear approximations for each of the 2^m subtweakeys. This was accomplished by considering all the possible linear trails over $\mathrm{D}_{4,k}$. The results are found in Table 5.1 for the case $m = 4$, i.e., for the 4-bit S-box, and in Table 5.2 for the case $m = 8$, i.e., for the 8-bit S-box. The first column lists the output masks and the third column lists the input masks. An asterisk denotes that the linear approximation holds for any subtweakey bit in that position. It turns out

that in both cases such linear approximations exist for a quarter of the subtweakeys. We call subtweakeys for which this property holds *weak*.

Consider a fixed subtweakey. If (u_1, w_1) and (u_2, w_2) are two perfect linear approximations, then their sum $(u_1 + u_2, w_1 + w_2)$ is again a perfect linear approximation, as evidenced by the tables. Moreover, the pair $(0, 0)$ is always a perfect linear approximation. It follows that the perfect linear approximations for a fixed subtweakey form a linear subspace of $\mathbb{F}_2^m \times \mathbb{F}_2^m$.

5.5 Patching the problem

To patch the problem, we search within a specific subset of S-boxes that are *permutation equivalent* [4] to the original.

Definition 51. *Two functions* $\mathrm{F}\colon \mathbb{F}_2^m \to \mathbb{F}_2^m$ *and* $\mathrm{G}\colon \mathbb{F}_2^m \to \mathbb{F}_2^m$ *are called* permutation equivalent *if there exist bit permutations* σ *and* τ *such that*

$$\mathrm{F} = \tau \circ \mathrm{G} \circ \sigma\,.$$

A bit permutation τ *is a permutation of* $\{0, \dots, m-1\}$ *that has been extended to* $\mathbb{F}_2^m$ *by*

$$(x_{m-1}, \dots, x_0) \mapsto (x_{\tau(m-1)}, \dots, x_{\tau(0)})\,.$$

Many of the cryptographic properties of an S-box are preserved by permutation equivalence, e.g., the algebraic degree, the differential uniformity, the linearity, and the branch number. Moreover, the impact of a bit permutation on the implementation cost is negligible. For example, in hardware it amounts to rewiring of the signals. We have restricted our search to those permutation equivalent S-boxes for which σ is the identity.

Any bit permutation applied to the output bits of S_4 permutes the columns of its correlation matrix. Indeed, we have

$$\mathrm{C}_{\mathrm{G}}(u, v) = \mathrm{C}_{\mathrm{S}_4}(u, \tau^{-1}(v))\,.$$

Table 5.3 lists the bit permutations τ and the ratio of subtweakeys for which there exist non-trivial perfect linear approximations. For example, the row "(x_2, x_1, x_0, x_3) 0" corresponds to the bit permutation $\tau = \mathrm{L}_4$ for which no subtweakeys are weak. It turns out that there exist many permutation equivalent S-boxes for which the double S-box structure does not have non-trivial perfect linear approximations for any subtweakey.

Similarly, for the 8-bit S-box we found that there exist many permutation equivalent S-boxes for which there exist no non-trivial perfect linear approximations. An example of such an S-box is obtained by applying the bit permutation $\tau(x_7, x_6, x_5, x_4, x_3, x_2, x_1, x_0) = (x_7, x_5, x_6, x_4, x_3, x_2, x_1, x_0)$. Because the number of possible bit permutations is large, we did not include them all here.

Table 5.1: Perfect linear approximations of $S_4 \circ T_{4,k} \circ S_4$ and their constituent linear trails.

output mask u	intermediate mask v	input mask w	subtweakey k	$C_{D_{4,k}}(u,w)$	$C_{T_{4,k}}(v,v)$	$C_S(u,v)$	$C_S(v,w)$
1000	0001	0010	$00{*}k_0$	$(-1)^{k_0}$	$(-1)^{k_0}$	1/2	1/2
	0101				$(-1)^{k_0}$	−1/2	−1/2
	1001				$(-1)^{k_0}$	−1/2	−1/2
	1101				$(-1)^{k_0}$	−1/2	−1/2
1010	0001	1110	0001	1	−1	−1/4	1/4
	0011				−1	1/4	−1/4
	0100				1	−1/2	−1/2
	0101				−1	1/4	−1/4
	0110				1	−1/2	−1/2
	0111				−1	−1/4	1/4
	1001				−1	−1/4	1/4
	1011				−1	1/4	−1/4
	1101				−1	−1/4	1/4
	1111				−1	1/4	−1/4
0010	0001	1100	0001	−1	−1	1/4	1/4
	0011				−1	1/4	1/4
	0100				1	1/2	−1/2
	0101				−1	−1/4	−1/4
	0110				1	−1/2	1/2
	0111				−1	−1/4	−1/4
	1001				−1	1/4	1/4
	1011				−1	1/4	1/4
	1101				−1	1/4	1/4
	1111				−1	1/4	1/4
0010	0001	1110	0011	−1	−1	1/4	1/4
	0011				1	1/4	−1/4
	0100				1	1/2	−1/2
	0101				−1	−1/4	−1/4
	0110				−1	−1/2	−1/2
	0111				1	−1/4	1/4
	1001				−1	1/4	1/4
	1011				1	1/4	−1/4
	1101				−1	1/4	1/4
	1111				1	1/4	−1/4
1010	0001	1100	0011	1	−1	−1/4	1/4
	0011				1	1/4	1/4
	0100				1	−1/2	−1/2
	0101				−1	1/4	−1/4
	0110				−1	−1/2	1/2
	0111				1	−1/4	−1/4
	1001				−1	−1/4	1/4
	1011				1	1/4	1/4
	1101				−1	−1/4	1/4
	1111				1	1/4	1/4

Table 5.2: Perfect linear approximations of $S_8 \circ T_{8,k} \circ S_8$ and their constituent linear trails.

output mask u	intermediate mask v	input mask w	subtweakey k	$C_{D_{8,k}}(u,w)$	$C_{T_{8,k}}(v,v)$	$C_S(u,v)$	$C_S(v,w)$
01000000	00010000	00001000	00*k_4****	$(-1)^{k_4}$	$(-1)^{k_4}$	1/2	1/2
	01010000				$(-1)^{k_4}$	−1/2	−1/2
	10010000				$(-1)^{k_4}$	−1/2	−1/2
	11010000				$(-1)^{k_4}$	−1/2	−1/2
10010000	00001000	00000010	0001****	−1	1	−1/2	1/2
	00011000				−1	−1/4	−1/4
	00101000				1	1/2	−1/2
	00111000				−1	−1/4	−1/4
	01011000				−1	1/4	1/4
	01111000				−1	1/4	1/4
	10011000				−1	1/4	1/4
	10111000				−1	1/4	1/4
	11011000				−1	1/4	1/4
	11111000				−1	1/4	1/4
11010000	00001000	00001010	0001****	1	1	−1/2	−1/2
	00011000				−1	−1/4	1/4
	00101000				1	−1/2	−1/2
	00111000				−1	1/4	−1/4
	01011000				−1	1/4	−1/4
	01111000				−1	−1/4	1/4
	10011000				−1	1/4	−1/4
	10111000				−1	−1/4	1/4
	11011000				−1	1/4	−1/4
	11111000				−1	−1/4	1/4
10010000	00001000	00001010	0011****	1	1	−1/2	−1/2
	00011000				−1	−1/4	1/4
	00101000				−1	1/2	−1/2
	00111000				1	−1/4	−1/4
	01011000				−1	1/4	−1/4
	01111000				1	1/4	1/4
	10011000				−1	1/4	−1/4
	10111000				1	1/4	1/4
	11011000				−1	1/4	−1/4
	11111000				1	1/4	1/4
11010000	00001000	00000010	0011****	−1	1	−1/2	1/2
	00011000				−1	−1/4	−1/4
	00101000				−1	−1/2	−1/2
	00111000				1	1/4	−1/4
	01011000				−1	1/4	1/4
	01111000				1	−1/4	1/4
	10011000				−1	1/4	1/4
	10111000				1	−1/4	1/4
	11011000				−1	1/4	1/4
	11111000				1	−1/4	1/4

Table 5.3: Permutation equivalent S-boxes and their ratio of weak subtweakeys.

$\tau(x_3, x_2, x_1, x_0)$	Ratio of weak subtweakeys
(x_3, x_2, x_1, x_0)	4/16
(x_2, x_3, x_1, x_0)	6/16
(x_3, x_1, x_2, x_0)	0
(x_2, x_1, x_3, x_0)	0
(x_1, x_3, x_2, x_0)	0
(x_1, x_2, x_3, x_0)	2/16
(x_3, x_2, x_0, x_1)	0
(x_2, x_3, x_0, x_1)	0
(x_3, x_1, x_0, x_2)	0
(x_2, x_1, x_0, x_3)	0
(x_1, x_3, x_0, x_2)	5/16
(x_1, x_2, x_0, x_3)	0
(x_3, x_0, x_2, x_1)	7/16
(x_2, x_0, x_3, x_1)	0
(x_3, x_0, x_1, x_2)	0
(x_2, x_0, x_1, x_3)	0
(x_1, x_0, x_3, x_2)	6/16
(x_1, x_0, x_2, x_3)	0
(x_0, x_3, x_2, x_1)	10/16
(x_0, x_2, x_3, x_1)	8/16
(x_0, x_3, x_1, x_2)	0
(x_0, x_2, x_1, x_3)	0
(x_0, x_1, x_3, x_2)	0
(x_0, x_1, x_2, x_3)	0

5.6 Conclusion

The main message that we want to communicate is that the composition of individually strong cryptographic functions may produce a weaker function for a large subset of the round tweakey space. In SKINNY, this weakness holds for *any* cipher key, because the subtweakeys are computed from the both the cipher key and the tweak, the latter of which is chosen by the user. In small structures, such undesired properties can be practically revealed through a combination of algebraic and statistical analysis. This shows that counting the number of active S-boxes in trails may have little meaning. Such properties could have been avoided by moving to a slightly different function at a negligible implementation cost.

We did not expect this kind of problem to exist for the 8-bit version of the SKINNY S-box. However, like the 4-bit S-box, in the composition of the two 8-bit S-boxes, the first stage of the second S-box and the final stage of the first S-box are the same, leading to cancellation. If the matrix that is used in the MixColumns step did not have a row with a single one, then this double S-box structure would not exist. As a result, this particular problem would not be there.

Acknowledgements. Joan Daemen and Daniël Kuijsters are supported by the European Research Council under the ERC advanced grant agreement under grant ERC-2017-ADG Nr. 788980 ESCADA.

References

1. C. Beierle, J. Jean, S. Kölbl, G. Leander, A. Moradi, T. Peyrin, Y. Sasaki, P. Sasdrich, and S. M. Sim. "The SKINNY Family of Block Ciphers and Its Low-Latency Variant MANTIS". In: *Advances in Cryptology - CRYPTO 2016 - 36th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 14-18, 2016, Proceedings, Part II*. Ed. by M. Robshaw and J. Katz. Vol. 9815. Lecture Notes in Computer Science. Springer, 2016, pp. 123–153. DOI: 10.1007/978-3-662-53008-5_5. URL: https://doi.org/10.1007/978-3-662-53008-5%5C_5.
2. E. Biham and A. Shamir. "Differential Cryptanalysis of DES-like Cryptosystems". In: *Advances in Cryptology - CRYPTO '90, 10th Annual International Cryptology Conference, Santa Barbara, California, USA, August 11-15, 1990, Proceedings*. Ed. by A. Menezes and S. A. Vanstone. Vol. 537. Lecture Notes in Computer Science. Springer, 1990, pp. 2–21. DOI: 10.1007/3-540-38424-3_1. URL: https://doi.org/10.1007/3-540-38424-3%5C_1.
3. N. Bordes, J. Daemen, D. Kuijsters, and G. V. Assche. "Thinking Outside the Superbox". In: *Advances in Cryptology - CRYPTO 2021 - 41st Annual International Cryptology Conference, CRYPTO 2021, Virtual Event, August 16-20, 2021, Proceedings, Part III*. Ed. by T. Malkin and C. Peikert. Vol. 12827. Lecture Notes in Computer Science. Springer, 2021, pp. 337–367. DOI: 10.1007/978-3-030-84252-9_12. URL: https://doi.org/10.1007/978-3-030-84252-9%5C_12.
4. C. Carlet. *Boolean Functions for Cryptography and Coding Theory*. Cambridge University Press, 2021. DOI: 10.1017/9781108606806.
5. J. Daemen. *Cipher and hash function design, strategies based on linear and differential cryptanalysis, PhD Thesis*. K.U.Leuven, 1995.
6. J. Daemen and V. Rijmen. *The Design of Rijndael - The Advanced Encryption Standard (AES), Second Edition*. Information Security and Cryptography. Springer, 2020. ISBN: 978-3-662-60768-8. DOI: 10.1007/978-3-662-60769-5.
7. T. Iwata, M. Khairallah, K. Minematsu, and T. Peyrin. "Duel of the Titans: The Romulus and Remus Families of Lightweight AEAD Algorithms". *IACR Trans. Symmetric Cryptol.* 2020:1, 2020, pp. 43–120. DOI: 10.13154/tosc.v2020.i1.43-120. URL: https://doi.org/10.13154/tosc.v2020.i1.43-120.
8. J. Jean. *TikZ for Cryptographers*. https://www.iacr.org/authors/tikz/. 2016.

9. J. Jean, I. Nikolic, and T. Peyrin. "Tweaks and Keys for Block Ciphers: The TWEAKEY Framework". In: *Advances in Cryptology - ASIACRYPT 2014 - 20th International Conference on the Theory and Application of Cryptology and Information Security, Kaoshiung, Taiwan, R.O.C., December 7-11, 2014, Proceedings, Part II*. Ed. by P. Sarkar and T. Iwata. Vol. 8874. Lecture Notes in Computer Science. Springer, 2014, pp. 274–288. DOI: 10.1007/978-3-662-45608-8_15. URL: https://doi.org/10.1007/978-3-662-45608-8%5C_15.

10. R. Lidl and H. Niederreiter. *Finite fields*. Second. Vol. 20. Encyclopedia of Mathematics and its Applications. Cambridge University Press, Cambridge, 1997. ISBN: 0-521-39231-4.

11. M. Matsui. "Linear Cryptanalysis Method for DES Cipher". In: *Advances in Cryptology - EUROCRYPT '93, Proceedings*. Ed. by T. Helleseth. DOI: 10.1007/3-540-48285-7_33.

12. K. Nyberg. "Generalized Feistel Networks". In: *Advances in Cryptology - ASIACRYPT '96, International Conference on the Theory and Applications of Cryptology and Information Security, Kyongju, Korea, November 3-7, 1996, Proceedings*. Ed. by K. Kim and T. Matsumoto. Vol. 1163. Lecture Notes in Computer Science. Springer, 1996, pp. 91–104. DOI: 10.1007/BFb0034838. URL: https://doi.org/10.1007/BFb0034838.

13. M. S. Turan, K. McKay, D. Chang, C. Calik, L. Bassham, J. Kang, and J. Kelsey. *Status Report on the Second Round of the NIST Lightweight Cryptography Standardization Process*. en. 2021. DOI: https://doi.org/10.6028/NIST.IR.8369. URL: https://tsapps.nist.gov/publication/get_pdf.cfm?pub_id=932630.

14. D. Verbakel. "Influence of Design on Differential and Linear Propagation Properties of Block Cipher Family SKINNY". Bachelor's Thesis. Nijmegen, the Netherlands: Radboud University, 2021.

6 Koala: A Low-Latency Pseudorandom Function

Parisa Amiri Eliasi[1], Yanis Belkheyar[1], Joan Daemen[1], Santosh Ghosh[2], Daniël Kuijsters[1], Alireza Mehrdad[1], Silvia Mella[1], Shahram Rasoolzadeh[1], Gilles Van Assche[3]

1 – Radboud University, The Netherlands
2 – Intel Labs, USA
3 – STMicroelectronics, Belgium

My contributions. This chapter is based on work accepted at Selected Areas in Cryptography 2024. I contributed to the design of the ExpandBlock function, the selection of the bit shuffle π, and the ordering of π and the mixing layer θ. Additionally, I was responsible for the comprehensive cryptanalysis of the PRF, in collaboration with Yanis.

Abstract. This paper introduces the Koala PRF, which maps a variable-length sequence of 64-bit input blocks to a single 257-bit output block. Its design focuses on achieving low latency in its implementation in ASIC. To construct Koala, we instantiate the recently introduced Kirby construction with the Koala-P permutation and add an input encoding layer. The Koala-P permutation is obtained as the 8-fold iteration of a simple round function inspired by that of Subterranean. Based on careful preliminary cryptanalysis, we made a variant of the Subterranean permutation by reordering and modifying it in a way that does not introduce any implementation overhead and enhances the cryptographic resistance of the resulting PRF. Indeed, we demonstrate that Koala exhibits a high resistance against integral, cube, division property, and higher-order differential attacks. Additionally, we compare the hardware implementation of Koala with the smallest latency with state-of-the-art low-latency PRF Orthros and Gleeok and the block cipher Prince in the same ASIC synthesis setup. Our results show that Koala outperforms these primitives not only in terms of latency but also with respect to various other performance metrics.

6.1 Introduction

The design of cryptographic primitives with minimum evaluation time in hardware implementation, so-called low-latency cryptography, is a relatively young line of research. Modern digital technologies often require a high level of security, but are expected to operate within very short time frames.

Important examples of such technologies are memory encryption and integrity mechanisms provided by, for example, IBM's SecureBlue, Intel's SGX, and AMD's SEV. Smart cards, like the ones of NXP and STMicroelectronics, perform local memory encryption in an ultra-constrained setting.

Another example is formed by the secure caches in modern CPU's. This application has received significant attention in the last few years, for microarchitectural attacks, e.g., Meltdown and Spectre, have revealed serious security shortcomings in widely deployed high-end processors. Many hardware-based mitigations for such attacks call for a higher level of encrypted communication inside of CPU's, as well as between CPU's and their surrounding hardware components. To implement new features of this kind in the next generations of mainstream processors, without causing a large performance penalty, low-latency encryption primitives are among the most important building blocks. Suffice it to say that the design of low-latency primitives is an important domain of research.

While various primitives have been developed with a focus on low latency, a significant portion of them are (tweakable) block ciphers. We just mention Prince [9, 10], Mantis [5], Qarma [2], Speedy [28], BipBip [6] and Scarf [12].

Interestingly, in recent times low-latency pseudorandom functions (PRF) have been proposed in the form of Orthros [4] and Gleeok [1], allowing ultra-fast stream encryption or authentication of short messages. Both are based on the sum-of-block-ciphers paradigm: To achieve beyond birthday bound PRF security, in the former the output is the sum of two block cipher invocations and in the latter even three. We investigate a different way to achieve n bits of security, namely with a $2n$-bit permutation and a feedforward, leading to a more efficient implementation in terms of area and latency.

In this paper, we present the design of a PRF with a variable-length input and fixed-length output suitable for a low-latency implementation in hardware as an ASIC. Koala is an instantiation of the Kirby [29] construction with a new permutation Koala-P inspired by Subterranean [18], and an additional input encoding. Moreover, Koala can be used as a stream cipher by taking as input the nonce followed by a counter. For this cipher the marginal cost per 256-bit keystream block is one call to Koala-P. We compare the performance of Koala with that of Orthros and the two instances of Gleeok, to the best of our knowledge the only PRFs in the literature with the main goal of providing a low-latency ASIC implementation. Our synthesis results, in Table 6.4, show that Koala has a lower latency and outperforms Orthros and Gleeok in various other performance measures. We believe that Koala is a promising new addition to the family of low-latency cryptographic primitives, and we welcome any third-party cryptanalysis.

Contribution. The main contributions of this paper are as follows:

- The design of Koala, a low-latency PRF that maps a variable-length sequence of 64-bit input blocks to a single 257-bit output block.
- An integral cryptanalysis of Koala, using bit-based division properties and the open source implementation of all algorithms used.[1]

[1] https://github.com/parisaeliasi/KoalaHW

- An RTL design of Koala in Verilog, an evaluation of the corresponding ASIC performance, and a comparison with Orthros Gleeok and Prince.

Organization of the paper. The paper is organized as follows. We establish some notation and conventions in Section 6.2. In Section 6.3, we present the specification of the permutation Koala-P, the pseudorandom function Koala, and the security claim of Koala. We present a short formalism to describe conditional cube attacks in Section 6.4 and in Section 6.5, we use this formalism and bit based integral distinguisher to analyse Koala. Bounds on the weights of linear and differential trails over Koala-P are provided in Section 6.6. In Section 6.7, we provide the rationale for all components of Koala. Finally, we present a hardware implementation in Verilog in Section 6.8, together with area and latency figures for implementation in ASIC. Appendices contains some figures, missing proofs for the interested reader, along with avalanche behaviour and differential, linear and integral distinguishers.

6.2 Notation and conventions

We fix the notation and conventions that are used throughout the paper.

We denote the cardinality of a set S by $|S|$. For sets S and T, we write $\mathrm{Maps}[S, T]$ for the set of all functions from S to T. By the set $\mathbb{N}$ we mean the non-negative integers, i.e, $0 \in \mathbb{N}$. Typically, n and m denote elements of $\mathbb{N}$. A finite sequence $s = (s_0, s_1, \ldots, s_{n-1})$ of elements of a set S is called an n-tuple. In particular, we reserve the word (bit) string for n-tuples over the set $\{0, 1\}$. We may also call a bit string a block if it has a fixed length of either 64 or 257 bits. The n-bit string consisting of all ones is denoted as 1^n. When we endow the set $\{0, 1\}$ with the structure of a finite field, we write $\mathbb{F}_2$ instead. Indices of tuples are computed modulo n, i.e., these indices are assumed to be elements of $\mathbb{Z}/n\mathbb{Z}$. If s and s' are two bit strings, then we write $s \parallel s'$ for the concatenation of s and s'. For example, $(0, 0, 1) \parallel (1, 0, 1)$ is equal to $(0, 0, 1, 1, 0, 1)$. We often treat n-bit strings as (bit) vectors $u = (u_0, u_1, \ldots, u_{n-1})$ in the n-dimensional vector space $\mathbb{F}_2^n$ over the field $\mathbb{F}_2$. We write e_i^n for the ith standard basis vector of $\mathbb{F}_2^n$. That is to say, e_i^n has a 1 in position i and zeros elsewhere. Sometimes, we refer to vectors as points. We make the set $\mathbb{F}_2^n$ into a partially ordered set by defining $u \preceq v$ if and only if $u_i \leq v_i$ for all $i \in \mathbb{Z}/n\mathbb{Z}$. The Hamming weight of a vector u is defined as the number of its non-zero coordinates. That is, $\mathrm{HW}(u) = |\{i : i \in \mathbb{Z}/n\mathbb{Z} \wedge u_i \neq 0\}|$. We define an affine subspace of $\mathbb{F}_2^n$ to be any set of the form $a + L$, where $a \in \mathbb{F}_2^n$ is a point and L is a linear subspace of $\mathbb{F}_2^n$. Let S be a subset of $\mathbb{F}_2^n$ and f a function defined on $\mathbb{F}_2^n$. We write $f|_S$ for the restriction of f to S.

6.3 Specification of Koala

Our design consists of two layers of abstraction: a permutation called Koala-P, and a PRF called Koala, that consists of a prefix-free input encoding function and the instantiation of the Kirby construction with Koala-P.

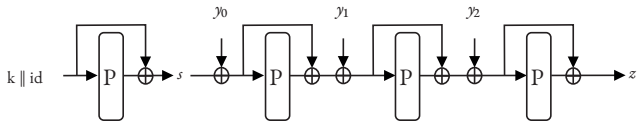


Figure 6.1: Illustration of Kirby applied to a 3-tuple of input blocks.

First, in Section 6.3.1, we recall the Kirby construction, introduced in [29]. Second, we specify the Koala-P permutation in Section 6.3.2. Third, we present the specification of the Koala PRF in Section 6.3.3 and its security claim in Section 6.3.4.

6.3.1 The Kirby construction

Kirby is a construction for building a variable-input-length pseudorandom function (VIL-PRF) from a permutation. This construction is specified in Algorithm 3 and illustrated in Figure 6.1.

To summarize Algorithm 3, Kirby is parameterized by a b-bit permutation P and a key length κ. It operates on a b-bit state that is initialized with a κ-bit secret key k and a $(b - \kappa)$-bit identifier. Then, it alternates between absorption of b-bit input blocks and transformations of the state by means of a call to the permutation P and a feed-forward. The input tuple of strings is assumed to be a codeword in a prefix code [13] (sometimes called a prefix-free code). It returns the final value of the b-bit state as the output. The paper [29] contains a proof of multi-user PRF security in the random permutation model, i.e., security against *generic* attacks.

From now on, we use the term key to refer to the master key k and secret to any intermediate state unknown to the attacker.

Algorithm 3 Definition of construction Kirby[P, κ] copied from [29], where P is a b-bit permutation and κ is a positive integer.

Input
- k — A κ-bit key string.
- id — A $(b - \kappa)$-bit key identifier string.
- y — An n-tuple of b-bit blocks with $n \geq 1$.

Output
- z — A b-bit block.

$s \leftarrow \mathrm{k} \parallel \mathrm{id}$
$s \leftarrow s \oplus \mathrm{P}(s)$
for $i = 0$ to $n - 1$ **do**
  $s \leftarrow s \oplus y_i$
  $s \leftarrow s \oplus \mathrm{P}(s)$
end for
$z \leftarrow s$
return z

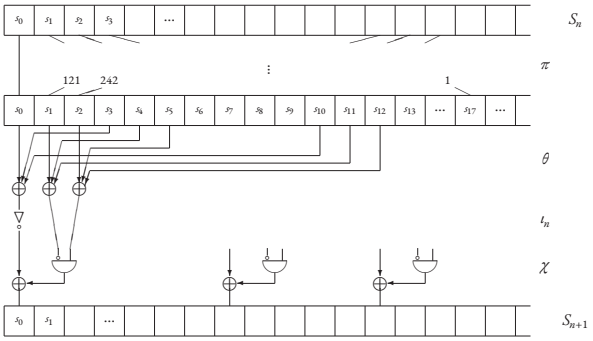


Figure 6.2: The Koala-P round function.

6.3.2 The Koala-P permutation

Koala-P is a permutation of $\mathbb{F}_2^{257}$ parameterized by the number of rounds $r \leq 8$. It is obtained by the self-composition of a round function, which, in turn, consists of a sequence of step functions.

First, we introduce the step functions: a bit shuffle π, a mixing layer θ, a round constant addition ι_j, and a non-linear layer χ. These functions are defined by how they compute the bit with index $i \in \mathbb{Z}/257\mathbb{Z}$ of a state vector $s \in \mathbb{F}_2^{257}$ according to the following rules:

$$
\begin{aligned}
&\pi : s_i \leftarrow s_{121i}\,, \\
&\theta : s_i \leftarrow s_i + s_{i+3} + s_{i+10}\,, \\
&\iota_j : s_i \leftarrow \begin{cases} s_i + 1 & \text{if } i = 0 \text{ and } j \notin \{2, 5, 6\}\,, \\ s_i & \text{otherwise}\,, \end{cases} \\
&\chi : s_i \leftarrow s_i + s_{i+2} + s_{i+1}s_{i+2}\,.
\end{aligned}
$$

Second, we define the round function R_j parameterized by the round index j as

$$\mathrm{R}_j = \chi \circ \iota_j \circ \theta \circ \pi\,.$$

Note that the only difference between the round functions lies in the value of the round constant. Lastly, we denote the composition of r rounds as

$$\text{Koala-P}[r] = \bigcirc_{j=0}^{r-1} \mathrm{R}_j\,.$$

6.3.3 The Koala PRF

The Koala PRF is composed of the following two parts:

- Kirby[Koala-P[8], κ]: the instantiation of Kirby with the permutation Koala-P[8] in which the key length κ is left as a parameter.
- An encoding function EncodePrefixFree defined in Algorithm 5 that maps an n-tuple of 64-bit blocks into a n-tuple of 257-bit blocks.
- An encoding of k and id as k||id||lenght(id) instead of k||id in the original Kirby construction, with length the encoding of the bitlength of id encoded in a single byte.

The ExpandBlock function makes it possible to use 64-bit blocks as input to the Kirby instance. Each 64-bit input block is transformed into a 256-bit string by the ExpandBlock function defined in Algorithm 4. Every 64-bit input block is split into a sequence of 32 2-bit strings. Each of these 2-bit strings naturally encodes an integer value between 0 and 3. This value serves as an index of the single non-zero element in a 4-bit string. The bits of this string are then diffused to different positions of the corresponding 256-bit output string.

The 256-bit strings are each padded with the bit 0, except for the last string, which is padded with the bit 1. This padding is what guarantees that the input tuple y to Algorithm 3 is an element of a prefix code.

The encoding of the k and id is injective, allowing use of different lengths without risking state collision for different keys. Concretely, given a κ-bit key, k, a $(257-\kappa)$-bit key identifier, id, and an n-tuple, d, of 64-bit blocks, we define Koala by

$$\text{Koala}[\kappa](\text{k}, \text{id}, d) = \text{Kirby}[\text{Koala-P}[8], \kappa](\text{k}, \text{id}, \text{EncodePrefixFree}(d)) .$$

6.3.4 The Koala security claim

We present a claim of multi-user PRF security of Koala in the case of μ users. We assume the existence of s identifiers and suppose that μ_i users share the ith identifier. Hence, we have $\mu = \mu_1 + \cdots + \mu_s$.

Claim 1. *We consider an adversary that is restricted to the following resources:*

- *The computational complexity is N and it is equal to the number of evaluations of Koala-P*[8].
- *The data complexity is M and it is equal to the number of distinct input blocks that are processed by Koala.*

The advantage of an adversary in distinguishing an array of μ instances of Koala$[\kappa]$ *loaded with μ independent κ-bit keys, sampled randomly and uniformly, from an array of μ independent random oracles, is upper bounded by*

$$\frac{M(M-1)}{2^{257}} + \frac{2NM}{2^{257}} + \frac{\sum_{i=1}^{s} \mu_i(\mu_i - 1)}{2^{\kappa+1}} + \frac{N \max_i \mu_i}{2^{\kappa}} .$$

Algorithm 4 Definition of ExpandBlock.

Input
s A 64-bit block.
Output
t A 256-bit string.

for $i = 0$ to 255 **do**

$$t_i \leftarrow \begin{cases} (s_{2i}+1)(s_{2i+1}+1) & \text{if } i \in [0,31]\,, \\ (s_{2i}+1)s_{2i+1} & \text{if } i \in [32,63]\,, \\ s_{2i}(s_{2i+1}+1) & \text{if } i \in [64,95]\,, \\ s_{2i}s_{2i+1} & \text{if } i \in [96,127]\,, \\ 0 & \text{otherwise}\,, \end{cases}$$

where the indices of s are computed modulo 64.

end for
return $(t_0, t_1, \ldots, t_{255})$

Algorithm 5 Definition of EncodePrefixFree

Require: $n \geq 1$
Input
d An n-tuple of 64-bit blocks.
Output
y An n-tuple of 257-bit blocks.

for $i = 0$ to $n-2$ **do**
$y_i \leftarrow \mathsf{ExpandBlock}(d_i) \parallel 0$
end for
$y_{n-1} \leftarrow \mathsf{ExpandBlock}(d_{n-1}) \parallel 1$
return $(y_0, y_1, \ldots, y_{n-1})$

This claim follows the proven bound of Kirby in [29] Lemma 1 page 15 against generic attacks using a κ-bit key.

6.4 Formalism for integral cryptanalysis

Together with differential and linear cryptanalysis, integral cryptanalysis form the three most important attack vectors. We use *integral attacks* as an umbrella term for attacks relying on summing the outputs of a function over a well-chosen input set, using different heuristics for constructing the set. To improve the understandability of our explanations of the attacks mounted against Koala, we first describe the general method used for integral attacks with the minimum mathematical formalism necessary to describe such attacks.

6.4.1 Framework of integral attacks

Integral attacks consist of an offline phase followed by an online phase:

Offline Phase is an analysis step where the adversary accesses the secret dependent polynomial representations of the step functions of a primitive. They apply rewrite rules to these polynomial representations in order to simplify them, e.g., to eliminate variables and lower the degree. Importantly, the rewrite rules determine an affine input space V. Using combinatorial arguments involving the degree or by propagating an initial division property vector [36], the adversary is able to determine the vector of coefficients of some target monomial. To be able to mount a successful attack, this vector should either be a constant that does not depend on the secret at all or depend on the secret in a way that leads to a system of equations that is easy to solve, e.g., linear dependence. The outcome of this step is an affine input space V and a target monomial x^u.

Online Phase is an execution step where the adversary accesses a cryptographic oracle for a fixed master key. They recover the vector of coefficients of the target monomial x^u by summing over the affine input space V that was obtained during the offline phase. The vector of coefficients is then used as a distinguisher or to set up a system of equations in the secret bits that may lead to the recovery of the master key.

We restrict ourselves to input sets that form an affine space. Within this restriction, examples of integral attacks include higher-order differential cryptanalysis [27], square attacks [17], and (conditional) cube attacks [20, 26]. We present a unified mathematical foundation upon which these attacks are built.

This section is organized as follows. In Section 6.4.2, we make explicit the link between functions defined on an affine space and their representation on this space as a multivariate polynomial, called the algebraic normal form. In Section 6.4.3, we introduce an notion of the derivative of a function and show how it can be computed by means of the summation of outputs of the function.

6.4.2 Algebraic normal form

To understand how to find input spaces for an integral attack, we need to explain how to represent the restriction of a vectorial Boolean function to some affine space as a tuple of multivariate polynomials: the algebraic normal form (ANF). We present the necessary tools and results from computational commutative algebra and make the relation between the algebraic normal form and substitutions, which determine the input sets, explicit. We also illustrate in 5 the notation and terminology used. For an accessible introduction to computational commutative algebra, we refer to the book by Cox et al. [14].

A monomial in the variables $x_0, \dots, x_{n-1}$ is a product of the form $x_0^{u_0} \cdots x_{n-1}^{u_{n-1}}$ with $u \in \mathbb{N}^n$. To abbreviate, we write this as x^u. The degree of the monomial x^u is defined as $u_0 + \cdots + u_{n-1}$. Polynomials are finite linear combinations of monomials with coefficients in $\mathbb{F}_2$. The degree of a polynomial is the largest of the degrees of its monomials. The zero polynomial has degree $-\infty$. We denote the set of polynomials in the variables $x_0, \dots, x_{n-1}$ and with coefficients in $\mathbb{F}_2$ by $R_n = \mathbb{F}_2[x_0, \dots, x_{n-1}]$. These variables correspond to the bits which are controlled by an adversary, e.g., the input bits.

Let $p_0, \ldots, p_{n-1}$ be polynomials of the form $p_i = x_i$ or $p_i = c_i + \sum_{j=i+1}^{n-1} a_{ij} x_j$ for constants $c_i \in \mathbb{F}_2$ and coefficients $a_{ij} \in \mathbb{F}_2$. During cryptanalysis, we make use of a set of rewrite rules of the form $x_i \to p_i$, i.e., we *substitute* x_i with the polynomial p_i. Rules of the form $x_i \to x_i$ are said to be *trivial* in the sense that no substitution is performed. A set of rewrite rules defines a set of polynomials of the form $x_i - p_i$, which is completely specified by a tuple (A, c), where $A = (a_{ij})$ is an $n \times n$ matrix over $\mathbb{F}_2$ and $c = (c_0, \ldots, c_{n-1})$ is a vector in $\mathbb{F}_2^n$. The matrix A is in row echelon form, up to a permutation of its rows, which implies that the order in which the corresponding rewrite rules are applied does not matter. The tuple (A, c) defines the affine space $V = \{v \in \mathbb{F}_2^n : Av = c\}$ of points that satisfy the equation $Av = c$.

We have seen that a rewrite rule of the form $x_i \to p_i$ gives us a relation of the form $x_i = p_i$. Moreover, we have relations of the form $x_i^2 = x_i$ due to the fact that the square on $\mathbb{F}_2$ is the identity map. We can introduce these relations by working with polynomials modulo the ideal I generated by the set

$$G = \{x_0^2 - x_0, \ldots, x_{n-1}^2 - x_{n-1}, x_0 - p_0, \ldots, x_{n-1} - p_{n-1}\}.$$

For our purposes, the central algebraic object is the quotient ring R_n/I.

Polynomials in R_n give rise to elements of $\mathrm{Maps}[V, \mathbb{F}_2]$. Indeed, for any point $a \in V$, there is a unique ring homomorphism $\varepsilon_a \colon R_n \to \mathbb{F}_2$ with $\varepsilon_a(x_i) = a_i$ given by substituting x_i by a_i. This leads to a map $\phi \colon R_n \to \mathbb{F}_2^V$ that is defined by $\phi(p) = f$ with $f(a) = \varepsilon_a(p)$ for all $a \in V$. The kernel of ϕ is equal to I. By the first isomorphism theorem for rings [14, p. 247], there is an isomorphism $\overline{\phi}$ between $\mathbb{F}_2^V$ and R_n/I.

The set G forms *a* Gröbner basis [14, p. 78] for I with respect to the lexicographic order. Define $W = \{u \in \mathbb{F}_2^n : u_i = 0 \text{ if } x_i \neq p_i\}$ as the set of vectors for which the ith component is zero if x_i is eliminated by a substitution. The remainder of any polynomial $p \in R_n$ on division by G, denoted $\overline{p}^G$, is unique and of the form

$$\overline{p}^G = \sum_{u \in W} \alpha_u x^u,$$

for certain constant bits $\alpha_u \in \mathbb{F}_2$ [14, p. 83]. Therefore, the set of all possible remainders after division by G, which we denote as R_G, forms a complete set of coset representatives of I in R_n. Indeed, let $\psi \colon R_n \to R_n$ be defined by $\psi(p) = \overline{p}^G$ for all $p \in R_n$. The kernel of ψ is equal to I. By the first isomorphism theorem for rings, there is an isomorphism $\overline{\psi}$ between R_n/I and R_G.

To conclude, we have an isomorphism $\mathcal{N} = \overline{\psi} \circ \overline{\phi}$ between the set of Boolean functions defined on V and the set of remainders R_G. We are now able to make precise how a function is represented on V.

Definition 52. *Let $f \colon V \to \mathbb{F}_2$ be a Boolean function defined on V. The representation of f as a multivariate polynomial, called the algebraic normal form (ANF) of f, is defined as the unique remainder $\mathcal{N}(f)$ upon division by G.*

The degree of a remainder $p \in R_G$ with $p \neq 0$ is defined as $\deg(p) = \max\{\mathrm{HW}(u) : u \in W \text{ and } \alpha_u \neq 0\}$.

Table 6.1: Truth table of f.

x	000	001	010	011	100	101	110	111
$f(x)$	0	1	0	0	1	0	1	1

Definition 53. *Let $f\colon V \to \mathbb{F}_2$ be a Boolean function defined on V. The algebraic degree of f, denoted by* $\deg(f)$*, is defined as the degree of its ANF.*

If f depends on a secret vector $s \in \mathbb{F}_2^\kappa$, e.g., a secret key or state, then the coefficients α_u of $\mathcal{N}(f)$ are Boolean functions of the secret bits, i.e., α_u maps the secret s to some bit $\alpha_u(s) \in \mathbb{F}_2$. In this case, we can rewrite the definition of the degree as $\deg(f) = \max\{\mathrm{HW}(u) : u \in W \text{ and there exists an } s \in \mathbb{F}_2^\kappa \text{ with } \alpha_u(s) \neq 0\}$. Note that our definitions match with the usual definitions of ANF and algebraic degree in the case that both A and c are zero.

There is a straightforward generalization of these notions to vectorial Boolean functions defined on V.

Definition 54. *The algebraic normal form of $F = (f_0, \dots, f_{m-1})\colon V \to \mathbb{F}_2^m$ is defined as $\mathcal{N}(F) = (\mathcal{N}(f_0), \dots, \mathcal{N}(f_{m-1})) \in R_n^m$. Its algebraic degree is defined as* $\deg(F) = \max\{\deg(f_0), \dots, \deg(f_{m-1})\}$.

We illustrate how to apply rewrite rules to $\mathcal{N}(f)$, where f is some Boolean function, in order to change its properties, such as the presence of certain monomials. The resulting polynomial is the ANF of the restriction of f to the affine space determined by the rewrite rules.

Example 5. *The function $f\colon \mathbb{F}_2^3 \to \mathbb{F}_2$ is defined by the truth table in Table 6.1. It follows that*

$$\mathcal{N}(f)(x_0, x_1, x_2) = x_0 + x_2 + x_1x_2\,.$$

Therefore, the algebraic degree of f is 2. Now we make the isomorphism $\mathcal{N}$ implicit.

We apply the rewrite rule $x_1 \to x_2$. This rule, together with the trivial rules, defines the matrix

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

and the constant $c = (0, 0, 0)$. Clearly, A is in row echelon form, up to a permutation of its rows. Moreover, $V = \{v \in \mathbb{F}_2^3 : Av = 0\} = \{v \in \mathbb{F}_2^3 : v_1 = v_2\}$. When we restrict f to V, i.e., when we consider $f|_V\colon V \to \mathbb{F}_2$, we find that its ANF is equal to x_0. The restriction has algebraic degree 1 and it depends on a single variable.

An alternative way of wording this is that we compose f with the map $L\colon \mathbb{F}_2^2 \to V$ given by $(x_0, x_1) \mapsto (x_0, x_1, x_1)$ and that the algebraic normal form of $f \circ L$ is equal to x_0.

Like in the example, we make implicit in the next section the correspondence between Boolean functions and their representation as a tuple of remainders.

6.4.3 Properties of derivatives

The integral attacks that we consider in this section, rely on practically computable properties of the derivative of a Boolean function. All definitions and results are extended to the case of vectorial Boolean functions by applying them to each coordinate Boolean function.

Definition 55. *For vectors $u, v \in \mathbb{F}_2^n$, define the derivative of the monomial x^v with respect to u by*

$$\partial_u x^v = \begin{cases} x^{v-u} & \text{if } u \preceq v\,, \\ 0 & \text{otherwise}\,, \end{cases}$$

and extend linearly to functions $f : \mathbb{F}_2^n \to \mathbb{F}_2$. We call $\partial_u f$ the derivative of f with respect to u.

Note that this definition coincides with that of the usual partial derivative.

Example 6. *Let $f : \mathbb{F}_2^3 \to \mathbb{F}_2$ be given by $f(x) = x_0 + x_2 + x_1x_2$. Its derivatives are equal to*

$$\begin{aligned} \partial_{(0,0,0)} f(x) &= x_0 + x_2 + x_1x_2 & \partial_{(1,0,0)} f(x) &= 1 \\ \partial_{(0,0,1)} f(x) &= x_1 + 1 & \partial_{(1,0,1)} f(x) &= 0 \\ \partial_{(0,1,0)} f(x) &= x_2 & \partial_{(1,1,0)} f(x) &= 0 \\ \partial_{(0,1,1)} f(x) &= 1 & \partial_{(1,1,1)} f(x) &= 0 \end{aligned}$$

The first important property of the derivative is the duality between the derivatives of f and outputs of f on an affine space by means of integral.

Proposition 23. *Let $f : \mathbb{F}_2^n \to \mathbb{F}_2$ and $a, u \in \mathbb{F}_2^n$. We have*

$$\begin{aligned} f(x + a) &= \sum_{0 \preceq u \preceq a} \partial_u f(x)\,, \text{and} \\ \partial_u f(x) &= \sum_{0 \preceq a \preceq u} f(x + a)\,. \end{aligned}$$

See in Appendix A, 28 for the proof.

The following corollary shows how to compute the coefficient α_u of x^u in f by summing over the outputs of f corresponding to inputs for which u takes on all possible values.

Corollary 4. *Let $f : \mathbb{F}_2^n \to \mathbb{F}_2$ and $a, u \in \mathbb{F}_2^n$. We have*

$$\alpha_u = \sum_{0 \preceq a \preceq u} f(a)\,.$$

Proof. This follows from the second equality in 23 and the fact that $\partial_u f(0) = \alpha_u$, by definition. □

The second important property of the derivative concerns its degree.

Proposition 24. *The degree of the derivative of f with respect to u satisfies*

$$\deg\big(\partial_u f\big) \leq \deg\big(f\big) - \mathrm{HW}(u)\,.$$

Proof. By definition, we have $\partial_u f = \sum_{u \leq v} \alpha_v x^{v-u}$. Let w be such that $\alpha_w \neq 0$ and $\deg(\partial_u f) = \mathrm{HW}(w-u)$. Using that $u \leq w$ and that x^w is a monomial in f, we find that $\deg\big(\partial_u f\big) = \mathrm{HW}(w-u) = \mathrm{HW}(w) - \mathrm{HW}(u) \leq \deg\big(f\big) - \mathrm{HW}(u)$. □

The coefficient of any monomial x^u with the Hamming weight of u exceeding the degree of the function is 0.

Proposition 25. *If* $\mathrm{HW}(u) \geq \deg\big(f(x)\big)$*, then* $\partial_u f(x)$ *is the coefficient* α_u *of* x^u *in* f*. In particular, if* $\mathrm{HW}(u) > \deg\big(f(x)\big)$*, then this coefficient* α_u *is* 0*.*

Proof. If $\mathrm{HW}(u) \geq \deg\big(f(x)\big)$, then $\deg\big(\partial_u f(x)\big) \leq 0$. This implies that $\partial_u f(x)$ is a constant, i.e., $\partial_u f(x) = \partial_u f(a)$ for any $a \in \mathbb{F}_2^n$. In particular, this is true for a equal to 0. By definition, it follows that $\partial_u f(0) = \alpha_u$. If $\mathrm{HW}(u) > \deg\big(f(x)\big)$, then $\deg\big(\partial_u f(x)\big) < 0$, which implies that α_u is 0. □

6.5 Integral attacks applied to Koala

In this section we focus on the class of integral attacks. They forms an important attack vector to consider in the analysis of Koala, due to the fact that the Koala-P round function has degree 2. In particular, we restrict ourselves to analyzing the substructure $\mathscr{E}_r$ of Koala in which only a single block is processed and consider a round-reduced version of Koala-P.

Definition 56. *Define an expansion function* $\gamma\colon \mathbb{F}_2^{257} \times \mathbb{F}_2^{64} \to \mathbb{F}_2^{257}$ *by*

$$\gamma(s, x) = (\mathsf{ExpandBlock}(x) \parallel 1) + s\,.$$

The substructure $\mathscr{E}_r\colon \mathbb{F}_2^{257} \times \mathbb{F}_2^{64} \to \mathbb{F}_2^{257}$ *is given by*

$$\mathscr{E}_r(s, x) = \gamma(s, x) + \text{Koala-P}[r](\gamma(s, x))\,.$$

The summary of the following is that we believe that $\mathscr{E}_r$ with the number of rounds $r \geq 6$ is secure against integral attacks.

We first investigate distinguishing bit-based division properties. The division property was introduced in [35] as a generalization of integral distinguishers. Based on previous works [19, 24, 25, 34, 36, 37], we created different tools to search for two-subset and two types of three-subset division properties within round-reduce version of Koala. Then, we look at cube and conditional cube distinguishers, exploiting the inner structure of the ExpandBlock function and the round function to search for integral distinguishers with smaller input size. Based on the results found, we conjecture on the feasibility of key recovery attacks using those distinguishers. In both cases, the goal of the attack is to find an affine subspace V of

$\mathbb{F}_2^{64}$, the domain of the 64-bit input string x, such that the ANF of $\mathscr{E}_r$ on this subspace has a coefficient that is independent of or linear in key variables, for some monomial in input variable.

6.5.1 Bit-based division property analysis

We divided our work into two steps, first using the two-subset division property and then using different types of three-subset division property. For further explanation on the division property we refer to [35] for basic concepts and the two subset division property, and to [24] for the three subset division property.

Using the algorithm from [34], and the model from [19] for the two subset division property, we check whether distinguishers exist within the round-reduced version of Koala. The model from [19] was very powerful to model the large state of Koala, and combined with the algorithm from Sun et al., we managed to model the propagation of division trails and to compute the existence of distinguisher up to 6 rounds. This technique, consisting in modeling the propagation of division trail using linear constraint, can lead to false positive results due to the lossy modeling of the constraint. However, from a designer's point of view, finding no distinguisher is enough, as this model captures all valid two-subset division trails. We found some distinguishers for up to 5 rounds but none for 6, showing the absence of exploitable two-subset division property for 6 rounds.

Then, we look at the three-subset-division property. We also used the model from [19] to model the propagation of division trail combined with the algorithm from [37]. For a specific set of input, we could compute the coefficient of the monomial containing all input variables after a certain number of rounds.

The result obtained was the monomial's presence, absence, or unknown status for each output coordinate, meaning for the latter that either the tool did not find the result or that such input is unlikely to result in an exploitable distinguisher. Due to the degree 2 round function used and the result found with the two-subset division property, we assume that there are distinguishers for up to 5 rounds. Therefore, we investigate 5 and 6 rounds distinguishers with our three-subset division property tools. As for five rounds, the expected maximum degree is 64 (2^5 for the round function time 2 for the ExpandBlock). This mean that for all secrets s for each output bit coordinates $\bigoplus_{x \in \mathbb{F}_2^{64}} \mathscr{E}_5(s, x) = 0$, leading to a 5-round integral distinguisher. Therefore, we investigated for 6 rounds, and we found that with the same input set, there is no exploitable distinguisher for each output coordinate. We assume that this result came from the presence of monomials of the form $\prod_{i=0}^{63} x_i \prod_{j \in J_p} s_j$ for each coordinate p after 6 rounds. Therefore, we search if some quadratic secret dependency, meaning $|J_p| = 2$, could lead to an exploitable distinguisher. For all pairs of secret-bit tested, we did not found any distinguisher for 6 rounds. We provide in `https://github.com/parisaeliasi/KoalaHW` all code used to compute those results, and we give in Appendix C some of the affine space leading to distinguisher for reduced round version.

6.5.2 Conditional cube attack

To push the analysis further, we consider what happens when we restrict our view of $\mathscr{E}_r$ to non-trivial affine subspaces of $\mathbb{F}_2^{64}$. These affine subspaces are obtained by applying substitutions that limit the in-

teraction of variables through the rounds. In other words, we looked at a subspace of the input vector space that can decrease the degree of the ANF of specific output coordinates.

A variable x_i is said to interact with a variable x_j in F if x_j appears in $\partial_{e_i''} F$. When it does not interact with any other variable, we call it isolated.

Definition 57. *Let $F\colon \mathbb{F}_2^n \to \mathbb{F}_2^m$ be a vectorial Boolean function and let $i \in \mathbb{Z}/n\mathbb{Z}$. We call the variable x_i* isolated *in F if $\deg(\partial_{e_i''} F) \leq 0$, i.e., if the derivative is a constant. We call F* linear *with respect to a set of variables $x_{i_1}, \ldots, x_{i_l}$ if these variables are isolated in F. By linearization of F, we mean the application of substitutions, after which F is linear with respect to the remaining variables.*

Linearizing γ: The following proposition shows that linearization of γ with respect to the variables $x_{i_1}, \ldots, x_{i_l}$, by applying suitable substitutions that lead to an affine space V, causes the absence of the monomial $x_{i_1} \cdots x_{i_l}$ in $\mathscr{C}_r|_V$. Intuitively, this is a consequence of the function having a much lower algebraic degree when restricted to particular affine subspaces than it has on the entire vector space. For a proof, we refer to Appendix A.

Proposition 26. *Let $r \geq 0$, $l \geq 2^r + 1$, and $\{i_1, \ldots, i_l\} \subseteq \mathbb{Z}/64\mathbb{Z}$ be a subset of indices of size l. If V is an affine subspace of $\mathbb{F}_2^{64}$ such that $x_{i_1}, \ldots, x_{i_l}$ are isolated in $\gamma|_V$, then $\partial_{e_{i_1}^{64} + \cdots + e_{i_l}^{64}} \mathscr{C}_r|_V = 0$.*

Each monomial of degree 2 in γ is of the form $x_i x_{i+1}$ for some index $i \in \mathbb{Z}/64\mathbb{Z}$. To linearize such a monomial, i.e., to have it depend on only a single variable, we can restrict ourselves to substitutions of the form $x_i \to x_{i+1}$ and $x_i \to a$ for a constant $a \in \mathbb{F}_2$.

In other terms, linearization of γ fixes 32 of the 64 input variables x_i. Therefore, using 26 and choosing r equal to 5, we find a distinguisher over $\mathscr{C}_5$.

Linearizing γ and R_0: The following proposition shows how to decrease the number of variables that are involved in the target monomial, i.e., to decrease the size of the input set over which we need to sum to obtain the coefficient of this target monomial. For a proof, we refer to 30 in Appendix A.

Proposition 27. *Let $r \geq 1$, $l \geq 2^r$, and $\{i_1, \ldots, i_l\} \subseteq \mathbb{Z}/64\mathbb{Z}$ be a subset of indices of size l. If V is an affine subspace of $\mathbb{F}_2^{64}$ such that x_{i_1} is isolated in $\mathrm{R}_0 \circ \gamma|_V$ and $x_{i_2}, \ldots, x_{i_l}$ are isolated in $\gamma|_V$, then $\partial_{e_{i_1}^{64} + \cdots + e_{i_l}^{64}} \mathscr{C}_r|_V = 0$.*

With 27, we sketch how to use a conditional cube attack [26] to recover particular bits of the secret. Let V_g be an affine subspace of $\mathbb{F}_2^{64}$ that depends on a guess $g \in \mathbb{F}_2^m$ for some subset of bits of the secret s. We call P_i the property that x_{i_1} is isolated in $\mathrm{R}_0 \circ \gamma|_{V_g}$. The attack consists in finding such V_g for which a correct secret guess will make P_i true and false for an incorrect guess. We obtain V_g by applying substitutions that depend on g. For example, $\mathscr{C}_r$ adds x_i to s_i, so if we apply the rewrite rule $x_i \to g_j$, where g_j is a guess for s_i, then a correct guess for s_i effectively removes the effect of s_i in any further processing. To verify our

guess, we recover the coefficient of the monomial $x_{i_1} \cdots x_{i_l}$ in $\mathscr{E}_r|_{V_g}$ by means of summation in the online phase. We write $u \in_R S$ if u is randomly and uniformly selected from the set S, and for $s \in_R \mathbb{F}_2^{257}$, we have

$$g = (s_{i_1}, \ldots, s_{i_m}) \Longrightarrow \partial_{e_{i_1}^{64}+\cdots+e_{i_l}^{64}} \mathscr{E}_r|_{V_g}(s, \cdot) = 0\,,$$
$$g \neq (s_{i_1}, \ldots, s_{i_m}) \Longrightarrow \Pr\left(\partial_{e_{i_1}^{64}+\cdots+e_{i_l}^{64}} \mathscr{E}_r|_{V_g}(s, \cdot) \neq 0\right) \approx 1\,.$$

We used that technique first to analyze a simpler version of our scheme: an Even-Mansour construction [21] in which the permutation is a round-reduced variant of the Subterranean permutation. As both elements are already well known, this was the starting point of our design. We found a key recovery attack for 6 rounds, using 32 isolated variables. The attack led to the recovery of key bits 0 and 2, requiring three days of computation on a desktop computer and it is possible to use this attack to recover each pair of bits; each can be performed in parallel. Consequently, a theoretical attack on 7 and 8 rounds exist using respectively 64 and 128 isolated variables. Those attacks would work the same with the Koala-P permutation instead of the Subterranean permutation as the components of the round function are very similar. However, together with the ExpandBlock function, we did not manage to attack the same number of rounds. The restriction from 257 to 64 bits for the input reduces the number of possible input sets for the attacker. With the degree 2 ExpandBlock function, it also reduces the number of rounds required to reach the maximum degree term in the ANF. Based on our observation, by using linearization, we can obtain the degree estimation as shown in Table 6.10 in the Appendix C.

From the three-subset division property, we saw that the degree after 5 rounds reaches 64, following the upper bound. Attempts at attacking more than 5 rounds, the trivial input set containing all 64 input variables can be used as a distinguisher, or the input set with 32 variables chosen carefully to linearize the ExpandBlock function. However, none of those methods can attack 6 rounds as the input is too small. Linearizing the input injection and the first round would mean that after 6 rounds, it could be possible to find the output coordinate for which the maximum degree would be 32. To linearize one variable for the input injection and the first round, we need to set 10 variables to constant, meaning that, at most, 6 variables can be linearized for these two rounds. As for the conditional cube attack above, we investigate a combination of variables linearized for the ExpandBlock function and linearized for the ExpandBlock function and the first round. So, let's assume we linearize one variable for the input injection and the first round. Then, we have 54 variables left, and to linearize those for the input injection, we reduce the space to 27. This leads us to think it is impossible to attack 6 rounds using this technique.

6.6 Trail bounds of Koala-P

In this section, we present bounds on the weights of differential and linear trails over the Koala-P permutation. We support this analysis with the best linear and differential trails for up to 3 rounds in Appendix D. Since we introduced a new permutation Koala-P, we decided to investigate first the permutation alone without considering the input injection. However we believe that with the result provided and the restriction on the input to 64-bit due to the input ExpandBlock function, it is very unlikely

Table 6.2: Lower bounds on the restriction weight of differential trails in Koala-P.

number of rounds	1	2	3	4	5	6	7	8
Koala-P	2	8	26	[52, 60]	≥ 54	≥ 60	≥ 78	≥ 104
Subterranean	2	8	25	58	≥ 62	≥ 78	≥ 80	≥ 116

that a 7/8-round differential with high enough probability or a 7/8-round linear approximation with high enough correlation could be found that would be useful in an attack on Koala.

6.6.1 Bounds on differential trails

To investigate the differential propagation properties of Koala-P, we used the differential trail search approach introduced in [31]. For more details, we refer the reader to [30, 31]. The general idea of this approach is to generate all 2-round *trail cores* with high differential probability (DP) and extend them iteratively to longer trail cores. A trail core represents a set of trails that are equal in all intermediate differences and only their input and the output difference are different. The *restriction weight* w_r of a trail core is the minimum among the restriction weights of all trails in it. For a differential trail Q, we use this restriction weight to approximate the differential probability $\mathrm{DP}(Q)$. Hence, if $w_r \ll b$ (the permutation width), then $\mathrm{DP}(Q) \approx 2^{-w_r(Q)}$.

We report on the lower bounds on the restriction weights of trails for different numbers of rounds of Koala-P and also Subterranean in Table 6.2. The bounds for Koala-P are tight for up to 3 rounds since we scanned the space up to restriction weight 26.

For 4-round trails, we scanned the space up to restriction weight 51 and found there is no trail up to this weight. During our search, we found a 4-round trail with restriction weight 60, implying that the best 4-round trail should weigh between 52 and 60. This means that 4-round trail for Koala-P are likely to have a lower bound close the one for Subterranean. For 5, 6, and 7 rounds, we found no trails, but the space was scanned up to the limits listed in Table 6.2. Moreover, in the case of 8 rounds, since each 8-round trail can be divided into two 4-round trails and since all 4-round trails have weight at least 52, each 8-round trail has weight at least $2 \times 52 = 104$.

6.6.2 Bounds on linear trails

For the linear trail search we could not build further on a similar work for Subterranean as there are no results known. Instead, we adapted works on Simon and Speck in [22, 32] to create a mixed integer linear programming (MILP) model for the propagation of linear masks. Our model provides lower bounds on the correlation weight of linear trails for a low number of rounds, where the correlation weight of a trail is the binary log of its correlation squared [15]. We use the Gurobi optimizer [23] to solve the model and find linear trails with the minimum correlation weight over 1,2,3 and 4 rounds.

During our trail search, we scanned the space up to correlation weight 26 and found a tight bound on the correlation weight of up to 3 rounds. For 4 rounds, we found a trail with the weight 54. Since the

Table 6.3: Lower bounds on the correlation weight of linear trails in Koala-P.

number of rounds	1	2	3	4
correlation weight	2	8	26	[38, 54]

search is top-down, we only know that the minimum weight for a trail of 4 rounds is between 38 and 54. Table 6.3 represents the lower bounds on the correlation weight of up to 4 rounds.

6.6.3 Clustering

Trails may cluster, differential trails that have the same input and output differences contribute to the same differential. Similarly, linear trails that have the same input and output masks contribute to the same linear approximation. Even if each contribution is small, the sum of all the contributions might not be. Still, as studied in [8], in permutations such as Koala-P, the maximum DP of differentials and the maximum correlation of linear approximations is typically very close to that of a single dominant trail. We decided to leave the study of clustering as future work.

6.7 Design rationale of Koala

This section presents the rationale behind the design of Koala. The starting point of the design was the Even-Mansour construction, instantiated with 8 iterations of the Subterranean round function for use in counter mode. We selected the round function for its short critical path, consisting of one 2-bit `NAND` gate and three 2-bit `XOR` gates, together with an `INV` gate. As we alluded to in Section 6.5, the evolution from this initial design to Koala has been driven by the goal of resistance against integral attacks. Indeed, when we allow an adversary to inject 257-bit blocks, a practical attack exists on 6 rounds and a theoretical attack on 7 rounds. Hence, 8 rounds would not be sufficient. Instead of changing the number of rounds, we started looking for changes in the design preferably with no or small implementation overhead.

The first step was to consider the round function itself. We changed ι to remove symmetry between the rounds that could possibly be exploited in cryptanalysis, e.g., slide attacks [7]. We changed θ and π to increase the number of variables that appear in the derivatives of the first few rounds for any variable. Finally, we reversed the order of the step functions. In particular, we moved χ to the end of the round to increase the diffusion of input before the non-linear layer of the first round. Also, at the permutation output, any linear layer after the non-linear does not contribute to its cryptographic strength. The result of these changes is the Koala-P permutation. However, those modifications alone were not enough to prevent 6 and 7-round attacks working on Subterranean. The next step was to allow only injecting a single 64-bit input block into the state instead of a 257-bit block. This ExpandBlock function is tailored to the Koala-P permutation in the sense that they have been designed together to resist integral attacks as described in 6.5. The ExpandBlock function essentially cuts the dimension of any affine space that an adversary can inject into half, and its implementation cost in terms of additional gates and gate delay

is small compared to that of an extra round. Due to the input restriction to 64-bit, we adopted the Kirby construction instead of Even-Mansour, as it allows for inputs consisting of an arbitrary number of 64-bit blocks. It is very suitable for low-latency applications, has a tight security bound in multi-user settings, and we handle the requirement of prefix-free input to Koala with a padding at the output of the ExpandBlock function.

6.8 Performance

We discuss a hardware architecture aimed for ASICs and report the synthesis results. The corresponding Verilog code and a software reference code for generating test vectors can be found at `https://github.com/parisaeliasi/KoalaHW`.

6.8.1 Hardware architecture of Koala

The block diagram for Koala is illustrated in Figure 6.3. It has one 257-bit state register S, a combinational circuit for computing $h(s, \text{sqz}) := \mathsf{ExpandBlock}(s) \parallel \text{sqz}$, a circuit for computing Koala-P, and control logic for absorbing and squeezing driven by two control signals: init and sqz.

- init = 1 the state is initialized with the image of key and id.
- init = 0 the operation is driven by sqz.
- sqz = 0 a non-final block absorbed, S updated and no output,
- sqz = 1 a final block absorbed, S not updated and output generated.

The circuit guarantees that the input to Koala is a prefix code by adding sqz in the input block, effectively indicating a final block. In stream cipher operation one first initializes the state, absorbs the blocks of the nonce with sqz = 0 and squeezes the keystream blocks by absorbing successive counter value blocks with sqz = 1. Four 2-bit `NOR` gates and two `INV` gates can encode the 2-bit input word (s_{2i} and s_{2i+1}) to a 4-bit output word, as explained in Algorithm 4. Koala-P is implemented with a fully unrolled circuit, where the logic of the 8 rounds is replicated and chained. Unrolling is the natural strategy to achieve low-latency, since it allows the evaluation of the whole permutation in one clock cycle.

6.8.2 Hardware results and comparison

We compare Koala with two other low-latency PRFs: Orthros [4] and Gleeok [1]. Both provide 128-bit output blocks. For additional comparison, we also consider the 64-bit block cipher Prince [9] and an instantiation of Koala where Koala-P is replaced by 8 rounds of the Subterranean permutation denoted by Kirby+sub.

For Orthros, Gleeok, and Prince, we used the RTL code publicly available [3, 11]. Note that these circuits are completely combinational. In fact, they do not need any flip-flop to store the intermediate

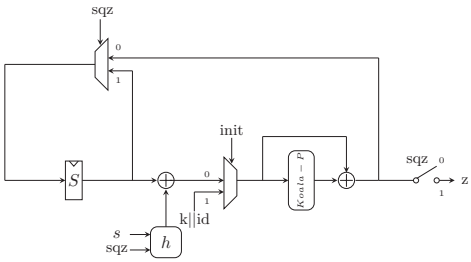


Figure 6.3: Block diagram of the Koala circuit.

cipher state. On the contrary, Koala's circuit has the storage element S to support variable-length inputs. Nevertheless, Koala has smaller area than Orthros and Gleeok.

The RTL codes were synthesized with Cadence Genus version 21.15 using the standard cell library Nangate 15nm. We ran the synthesis flow multiple times for each cipher with different timing constraints, until the clock period is just above the critical path of the circuit. In Table 6.4, we report the best results in terms of maximum throughput/area^2 for each cipher. The maximum throughput (MaxTp) is intended here as the maximum number of bits that a circuit can output per second, and it is computed as output width divided by latency. More results, including the minimum latency reached by each cipher, are given in Table 6.13.

We can observe that Koala and Kirby+sub achieve the lowest latency and highest throughput among all ciphers and have similar area. This confirms that the modifications we made to Subterranean round function do not introduce significant implementation overhead while improving the security as shown in Section 6.5.2. Koala takes twice the area of Prince, but compensates this with a 257-bit output, 4 times longer than that of Prince. Gleeok-128 occupies twice the area of Koala and its output is 128 bits, only half that of Koala. While Gleeok-256 has block width similar to that of Koala, but its area is more than 5 times bigger. With respect to the metric MaxTp/Area^2, Prince achieves the best tradeoff, thanks to its very compact circuit. However, when we consider the lowest latency in Table 6.13, Koala outperforms Prince, Orthros and Gleeok thanks to its small area and larger output width.

Table 6.4: Synthesis results for the Nangate 15nm library.

Cipher	Output width [bits]	Area [μm^2]	Area [GE]	Latency [ps]	MaxTp [Gbits/s]	MaxTp/Area [Mbits/(s × μm^2)]
Koala	257	4175	21236	395	651	156
Kirby+sub	257	4167	21196	399	644	155
Prince	64	1696	8627	482	133	78.4
Orthros	128	5993	30482	400	320	53.4
Gleeok-128	128	9887	50291	400	320	32.4
Gleeok-256	256	26043	132462	550	465	17.8

6.9 Conclusion

With the design of Koala, we provide an open-source implementation of a new PRF for low-latency that performs much better than Orthrosand Gleeokon several metrics. The security analysis performed and supported with all open-source tools used, shows that using 8 rounds of Koala-P with the ExpandBlock should be secure against known attacks.

Acknowledgements

This work was partially funded by Intel through the Crypto Frontiers Research Center. Alireza Mehrdad, Joan Daemen, and Daniël Kuijsters are supported by the European Research Council under the ERC advanced grant agreement under grant ERC-2017-ADG Nr. 788980 ESCADA. Shahram Rasoolzadeh is supported by the Netherlands Organisation for Scientific Research (NWO) under TOP grant TOP1.18.002 SCALAR. Parisa Amiri Eliasi is supported by the Cryptography Research Center of the Technology Innovation Institute (TII), Abu Dhabi (UAE), under the TII-Radboud project with the title Evaluation and Implementation of Lightweight Cryptographic Primitives and Protocols. Silvia Mella was funded by the Dutch Research Council (NWO) through the PROACT project (NWA.1215.18.014).

A Missing proofs

Proposition 28. *Let* $f\colon \mathbb{F}_2^n \to \mathbb{F}_2$ *and* $a, u \in \mathbb{F}_2^n$. *We have*

$$f(x+a) = \sum_{0 \le u \le a} \partial_u f(x)\,, \text{ and}$$
$$\partial_u f(x) = \sum_{0 \le a \le u} f(x+a)\,.$$

Proof. The first equality can be seen as follows. Using the ANF of f, we find that

$$\begin{aligned} f(x+a) &= \sum_{0 \le w} \alpha_w (x+a)^w \qquad = \sum_{0 \le w} \alpha_w \left(\sum_{0 \le u \le w} x^{w-u} a^u \right) \\ &= \sum_{0 \le w} \left(\sum_{0 \le u \le w} \alpha_w x^{w-u} a^u \right) = \sum_{0 \le u} \left(\sum_{u \le w} \alpha_w x^{w-u} a^u \right) \\ &= \sum_{0 \le u} \left(\sum_{u \le w} \alpha_w x^{w-u} \right) a^u \quad = \sum_{0 \le u \le a} \left(\sum_{u \le w} \alpha_w x^{w-u} \right) \\ &= \sum_{0 \le u \le a} \partial_u f(x)\,, \end{aligned}$$

where we have applied the definition of the derivative and used the fact that $a^u = 1$ if and only if $0 \le u \le a$. The second equality follows from the Möbius inversion formula [33, p. 264] applied to the first. □

Proposition 29. *Let $r \geq 0$, $l \geq 2^r + 1$, and $\{i_1, \ldots, i_l\} \subseteq [0, 63]$ be a subset of indices of size l. If V is an affine subspace of $\mathbb{F}_2^{64}$ such that $x_{i_1}, \ldots, x_{i_l}$ are isolated in $\gamma|_V$, then*

$$\partial_{e_{i_1}^{64} + \cdots + e_{i_l}^{64}} \mathscr{E}_r|_V = 0 \,.$$

Proof. This proof is in the same style as Theorem 2 from [26]. Let V be an affine subspace such that each x_{i_j} is isolated in $\gamma|_V$. By linearity, it suffices to prove both $\partial_{e_{i_1}^{64} + \cdots + e_{i_l}^{64}} \gamma|_V = 0$ and $\partial_{e_{i_1}^{64} + \cdots + e_{i_l}^{64}} \text{Koala-P}[r] \circ \gamma|_V = 0$. The first equality is trivial, so we only prove the second. To that end, let $f_1, \ldots, f_t$ be the monomials containing $x_{i_1}, \ldots, x_{i_l}$ in the output of $\gamma|_V$. By definition, the degree of each f_j is at most one with respect to $x_{i_1}, \ldots, x_{i_l}$. Now, any monomial T in Koala-P$[r] \circ \gamma|_V$ of maximum degree with respect to $x_{i_1} x_{i_2} \cdots x_{i_l}$ is of the form

$$T = f_1 f_2 \cdots f_h \quad \text{for some } h \in \mathbb{Z}_{\geq 0} \text{ with } h \leq 2^r \,,$$

because the algebraic degree of each R_j is 2. It follows that T contains at most h different $x_{i_1}, \ldots, x_{i_l}$. Suppose now that

$$\partial_{e_{i_1}^{64} + \cdots + e_{i_l}^{64}} T \neq 0 \,.$$

Then $x_{i_1} \cdot x_{i_2} \cdots x_{i_l}$ divides T, which implies that $h \geq l$. Therefore,

$$h \geq l \geq 2^r + 1 > 2^r$$

which is a contradiction. Since T does not appear in the derivative, any lower degree monomials do not appear either and the result follows. □

Proposition 30. *Let $r \geq 1$, $l \geq 2^r$, and $\{i_1, \ldots, i_l\} \subseteq [0, 63]$ be a subset of indices of size l. If V is an affine subspace of $\mathbb{F}_2^{64}$ such that x_{i_1} is isolated in $\mathrm{R}_0 \circ \gamma|_V$ and $x_{i_2}, \ldots, x_{i_l}$ are isolated in $\gamma|_V$, then*

$$\partial_{e_{i_1}^{64} + \cdots + e_{i_l}^{64}} \mathscr{E}_r|_V = 0 \,.$$

Proof. This proof has been adapted from Theorem 2 from [26]. Let V be an affine subspace of $\mathbb{F}_2^{64}$ such that x_{i_1} is isolated in $\mathrm{R}_0 \circ \gamma|_V$ and $x_{i_2}, \ldots, x_{i_l}$ are isolated in $\gamma|_V$. By linearity, it suffices to prove both $\partial_{e_{i_1}^{64} + \cdots + e_{i_l}^{64}} \gamma|_V = 0$ and $\partial_{e_{i_1}^{64} + \cdots + e_{i_l}^{64}} \text{Koala-P}[r] \circ \gamma|_V = 0$. The first equality is trivial, so we only prove the second. To that end, let $f_1, \ldots, f_s$ be the monomials containing x_{i_1} in $\mathrm{R}_0 \circ \gamma|_V$. By definition, the degree of each f_i is exactly one with respect to x_{i_1}. Similarly, let $g_1, \ldots, g_t$ be the monomials containing $x_{i_2}, \ldots, x_{i_l}$ in $\mathrm{R}_0 \circ \gamma|_V$. By definition the degree of each g_j is at most two with respect to $x_{i_2}, \ldots, x_{i_m}$. Moreover, x_{i_1} does not divide any g_j, because that would contradict the assumption that it is isolated. Now, any monomial T in Koala-P$[r] \circ \gamma|_V$ of maximum degree with respect to $x_{i_1} x_{i_2} \cdots x_{i_l}$ is of the form

$$T = f_1 f_2 \cdots f_h g_1 g_2 \cdots g_{h'} \quad \text{for some } h, h' \in \mathbb{Z}_{\geq 0} \text{ with } h + h' \leq 2^{r-1} \,,$$

because the algebraic degree of G is 2. It follows that T contains at most $2h'$ different $x_{i_2}, \dots, x_{i_l}$ and at most one x_{i_1}. Suppose now that

$$\partial_{e_{i_1}^{64}+\dots+e_{i_l}^{64}} T \neq 0 .$$

Then $x_{i_1} \cdot x_{i_2} \cdots x_{i_l}$ divides T, which implies that $2h' \geq l$ and $h \geq 1$. Therefore,

$$h + h' \geq 1 + \frac{l}{2} = 1 + \frac{2^r}{2} = 1 + 2^{r-1} > 2^{r-1} ,$$

which is a contradiction. Since T does not appear in the derivative, any lower degree monomials do not appear either and the result follows. □

B Diffusion test

In Table 6.5 we use the definition of [16] for the avalanche dependency weight and entropy. We report on the avalanche behaviour for the Subterranean permutation, the Koala-P permutation and the Koala-P permutation with the input injection. We provide at `https://github.com/parisaeliasi/KoalaHW` the C code use to produce those result.

Table 6.5: Diffusion test for the Subterranean permutation, the Koala-P permutation and the Koala-P with the ExpandBlock function. For each we compute the dependency (D), the weight (W) and the entropy (E).

	Subterranean			Koala-P			Koala-P + ExpandBlock		
number of round	D	W	E	D	W	E	D	W	E
1	9	6.00	5.99	9	6.002	5.99	36	12.18	30.90
2	81	36.00	65.20	81	35.99	65.20	167	55.43	140.72
3	255	109.20	236.54	251	108.75	230.97	257	122.95	254.62
4	257	128.36	256.99	257	128.39	256.99	257	128.50	256.99

C Integral distinguishers

We give in the Table 6.6, Table 6.8 and Table 6.9 integral distinguishers found with our tool for reduced-round versions of Koala. We represent the input affine space used with the list of input variable indexes, and the output bit coordinate correspond to the output bit index after r round where this affine space leads to a integral distinguisher. We also provide at `https://github.com/parisaeliasi/KoalaHW` all code to reproduce our result and to search for integral distinguisher.

Due to the input injection we can consider terms in monomials as a product of two input variable. Therefore, we know in advance that if x_{2i} appears than it will always be together with x_{2i+1}. This strongly reduces the search space for monomials.

D Differential and linear trails

In this section we provide the trail with least weight for 1, 2 and 3 round as found by our tools. We represent trails with a list of pair of indices, one for the round number and one for the index position within the state.

E Additional hardware results

In Table 6.13, we present more synthesis results and highlight the best result for each metric.

Table 6.6: 1 Round integral distinguisher

Affine Space	Output Bit Coordinate
[24, 25, 28, 29]	6, 233
[32, 33, 60, 61]	14, 44, 74
[0, 1, 36, 37]	48, 78
[16, 17, 52, 53]	184, 214
[2, 3, 38, 39]	65, 95
[8, 9, 12, 13]	97, 127
[10, 11, 14, 15]	114, 144, 174
[22, 23, 50, 51]	216, 246

Table 6.7: 2 Rounds integral distinguisher

Affine Space	Output Bit Coordinate
[24, 25, 28, 29, 32, 33, 60, 61]	157, 161, 164, 166, 168
[0, 1, 36, 37, 16, 17, 52, 53]	29, 33, 36, 39, 43
[2, 3, 38, 39, 22, 23, 50, 51]	62, 66, 69, 72, 73, 76

Table 6.8: 3 Rounds integral distinguisher

Affine Space	Output Bit Coordinate
[24, 25, 28, 29, 32, 33, 60, 61, 0, 1, 36, 37, 16, 17, 52, 53]	87, 94, 97, 155, 165, 206, 213, 216
[2, 3, 38, 39, 8, 9, 12, 13, 10, 11, 14, 15, 22, 23, 50, 51]	38, 200

Table 6.9: 4 Rounds integral distinguisher

Affine Space	Output Bit Coordinate
[24, 25, 28, 29, 32, 33, 60, 61, 0, 1, 36, 37, 16, 17, 52, 53, 2, 3, 38, 39, 8, 9, 12, 13, 10, 11, 14, 15, 22, 23, 50, 51]	15, 49, 54, 84, 91, 94, 239

Table 6.10: Upper bound on the degree growth based on the type of linearization used.

	number of rounds					
type of linearization	1	2	3	4	5	6
0 round	4	8	16	32	64	128
ExpandBlock function	2	4	8	16	32	64
ExpandBlock function + first round	1	2	4	8	16	32

Table 6.11: Differential trail for Koala

Round	Weight	Indexes
1	2	[1, 0]
2	6	[2, 0], [2, 247], [2, 254]
3	18	[3, 0], [3, 65], [3, 72], [3, 75], [3, 141], [3, 148], [3, 151], [3, 247], [3, 254]

Table 6.12: Linear trail for Koala

Round	Weight	Indexes
1	2	[1, 0]
2	6	[2, 0], [2, 106], [2, 182]
3	18	[3, 0], [3, 26], [3, 82], [3, 102], [3, 106], [3, 158], [3, 177], [3, 182], [3, 233]

Table 6.13: Extended synthesis results on Nangate 15nm.

Cipher	Output width [bits]	Area [μm^2]	Area [GE]	Latency [ps]	MaxTp [Gbits/s]	MaxTp/Area [Mbits/(s × μm^2)]	MaxTp/Area^2 [Gbits/(s × mm^4)]
Koala	257	4079.67	20750	472	544	133.34	32.715
		4175.07	21236	395	651	155.92	37.326
		5639.80	28686	300	857	151.95	26.933
		6621.41	33678	290	886	133.80	20.213
Kirby+sub	257	4167.30	21196	399	644	154.53	37.089
		5203.97	26469	300	857	164.68	31.633
		6035.42	30698	290	886	146.80	24.329
Prince	64	1696.19	8627	482	133	78.41	46.152
		1935.95	9847	450	142	73.38	37.947
		2957.03	15040	410	156	52.75	17.852
Orthros	128	5898.98	30004	499	257	43.57	07.372
		5978.75	30410	449	285	47.67	07.975
		5993.05	30482	400	320	53.39	08.910
		7295.73	37108	370	346	47.42	06.499
		8556.58	43521	360	356	41.60	04.856
Gleeok-128	128	9726.98	49474	436	294	30.22	03.103
		9887.61	50291	400	320	32.36	03.273
		13270.55	67498	370	346	26.07	01.964
Gleeok-256	256	25986.71	132175	600	427	16.43	00.632
		26043.19	132462	550	465	17.85	00.686
		29288.50	148969	520	492	16.79	00.574
		31468.54	160057	510	502	15.95	00.507

References

1. R. Anand, S. Banik, A. Caforio, T. Ishikawa, T. Isobe, F. Liu, K. Minematsu, M. Rahman, and K. Sakamoto. "Gleeok: A Family of Low-Latency PRFs and its Applications to Authenticated Encryption". *IACR Trans. Cryptogr. Hardw. Embed. Syst.* 2024:2, 2024, pp. 545–587. DOI: 10.46586/TCHES.V2024.I2.545-587.
2. R. Avanzi. "The QARMA Block Cipher Family. Almost MDS Matrices Over Rings With Zero Divisors, Nearly Symmetric Even-Mansour Constructions With Non-Involutory Central Rounds, and Search Heuristics for Low-Latency S-Boxes". *IACR Trans. Symmetric Cryptol.* 2017:1, 2017, pp. 4–44.
3. S. Banik. *Orthros*. 2021. URL: https://github.com/subhadeep-banik/orthros.
4. S. Banik, T. Isobe, F. Liu, K. Minematsu, and K. Sakamoto. "Orthros: A Low-Latency PRF". *IACR Trans. Symmetric Cryptol.* 2021:1, 2021, pp. 37–77.
5. C. Beierle, J. Jean, S. Kölbl, G. Leander, A. Moradi, T. Peyrin, Y. Sasaki, P. Sasdrich, and S. M. Sim. "The SKINNY Family of Block Ciphers and Its Low-Latency Variant MANTIS". In: *Advances in Cryptology - CRYPTO 2016 - 36th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 14-18, 2016, Proceedings, Part II*. Ed. by M. Robshaw and J. Katz. Vol. 9815. Lecture Notes in Computer Science. Springer, 2016, pp. 123–153. DOI: 10.1007/978-3-662-53008-5_5. URL: https://doi.org/10.1007/978-3-662-53008-5%5C_5.
6. Y. Belkheyar, J. Daemen, C. Dobraunig, S. Ghosh, and S. Rasoolzadeh. "BipBip: A Low-Latency Tweakable Block Cipher with Small Dimensions". *IACR Trans. Cryptogr. Hardw. Embed. Syst.* 2023:1, 2023, pp. 326–368.
7. A. Biryukov and D. Wagner. "Slide Attacks". In: *Fast Software Encryption*. Ed. by L. Knudsen. Springer Berlin Heidelberg, Berlin, Heidelberg, 1999, pp. 245–259. ISBN: 978-3-540-48519-3.
8. N. Bordes, J. Daemen, D. Kuijsters, and G. V. Assche. "Thinking Outside the Superbox". In: *Advances in Cryptology - CRYPTO 2021 - 41st Annual International Cryptology Conference, CRYPTO 2021, Virtual Event, August 16-20, 2021, Proceedings, Part III*. Ed. by T. Malkin and C. Peikert. Vol. 12827. Lecture Notes in Computer Science. Springer, 2021, pp. 337–367. DOI: 10.1007/978-3-030-84252-9_12. URL: https://doi.org/10.1007/978-3-030-84252-9%5C_12.
9. J. Borghoff, A. Canteaut, T. Güneysu, E. B. Kavun, M. Knezevic, L. R. Knudsen, G. Leander, V. Nikov, C. Paar, C. Rechberger, P. Rombouts, S. S. Thomsen, and T. Yalçin. "PRINCE - A Low-Latency Block Cipher for Pervasive Computing Applications - Extended Abstract". In: *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*. Ed. by X. Wang and K. Sako. Vol. 7658. Lecture Notes in Computer Science. Springer, 2012, pp. 208–225.

10. D. Bozilov, M. Eichlseder, M. Knezevic, B. Lambin, G. Leander, T. Moos, V. Nikov, S. Rasoolzadeh, Y. Todo, and F. Wiemer. "PRINCEv2 - More Security for (Almost) No Overhead". In: *Selected Areas in Cryptography - SAC 2020 - 27th International Conference, Halifax, NS, Canada (Virtual Event), October 21-23, 2020, Revised Selected Papers*. Ed. by O. Dunkelman, M. J. J. Jr., and C. O'Flynn. Vol. 12804. Lecture Notes in Computer Science. Springer, 2020, pp. 483–511.

11. A. Caforio. *Gleeok*. 2023. URL: https://github.com/qantik/gleeok.

12. F. Canale, T. Güneysu, G. Leander, J. P. Thoma, Y. Todo, and R. Ueno. "SCARF - A Low-Latency Block Cipher for Secure Cache-Randomization". In: *32nd USENIX Security Symposium, USENIX Security 2023, Anaheim, CA, USA, August 9-11, 2023*. Ed. by J. A. Calandrino and C. Troncoso. USENIX Association, 2023.

13. T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms*. 2nd. The MIT Press, 2001. ISBN: 0262032937.

14. D. A. Cox, J. Little, and D. O'Shea. *Ideals, Varieties, and Algorithms*. Fourth. Undergraduate Texts in Mathematics. Springer, 2015. ISBN: 978-3-319-16720-6.

15. J. Daemen. *Cipher and hash function design, strategies based on linear and differential cryptanalysis, PhD Thesis*. http://jda.noekeon.org/. K.U.Leuven, 1995.

16. J. Daemen, S. Hoffert, G. Van Assche, and R. Van Keer. "The design of Xoodoo and Xoofff". *IACR Trans. Symmetric Cryptol.* 2018:4, 2018, pp. 1–38. DOI: 10.13154/tosc.v2018.i4.1-38.

17. J. Daemen, L. R. Knudsen, and V. Rijmen. "The Block Cipher Square". In: *FSE*. Vol. 1267. LNCS. 1997, pp. 149–165.

18. J. Daemen, P. M. C. Massolino, A. Mehrdad, and Y. Rotella. "The Subterranean 2.0 Cipher Suite". *IACR Trans. Symmetric Cryptol.* 2020:S1, 2020, pp. 262–294.

19. P. Derbez and B. Lambin. "Fast MILP Models for Division Property". *IACR Trans. Symmetric Cryptol.* 2022:2, 2022, pp. 289–321. DOI: 10.46586/TOSC.V2022.I2.289-321.

20. I. Dinur and A. Shamir. "Cube Attacks on Tweakable Black Box Polynomials". *IACR Cryptol. ePrint Arch.*, 2008, p. 385.

21. S. Even and Y. Mansour. "A Construction of a Cipher from a Single Pseudorandom Permutation". *J. Cryptol.* 10:3, 1997, pp. 151–162. DOI: 10.1007/S001459900025. URL: https://doi.org/10.1007/s001459900025.

22. K. Fu, M. Wang, Y. Guo, S. Sun, and L. Hu. "MILP-Based Automatic Search Algorithms for Differential and Linear Trails for Speck". In: *Fast Software Encryption - 23rd International Conference, FSE 2016, Bochum, Germany, March 20-23, 2016, Revised Selected Papers*. Ed. by T. Peyrin. Vol. 9783. Lecture Notes in Computer Science. Springer, 2016, pp. 268–288.

23. Gurobi Optimization, LLC. *Gurobi Optimizer Reference Manual*. 2023.

24. Y. Hao, G. Leander, W. Meier, Y. Todo, and Q. Wang. “Modeling for Three-Subset Division Property without Unknown Subset”. *J. Cryptol.* 34:3, 2021, p. 22.

25. Y. Hao, G. Leander, W. Meier, Y. Todo, and Q. Wang. “Modeling for Three-Subset Division Property Without Unknown Subset - Improved Cube Attacks Against Trivium and Grain-128AEAD”. In: *Advances in Cryptology - EUROCRYPT 2020 - 39th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Zagreb, Croatia, May 10-14, 2020, Proceedings, Part I*. Ed. by A. Canteaut and Y. Ishai. Vol. 12105. Lecture Notes in Computer Science. Springer, 2020, pp. 466–495. DOI: 10.1007/978-3-030-45721-1_17.

26. S. Huang, X. Wang, G. Xu, M. Wang, and J. Zhao. “Conditional Cube Attack on Reduced-Round Keccak Sponge Function”. *IACR Cryptol. ePrint Arch.*, 2016, p. 790.

27. X. Lai. “Higher Order Derivatives and Differential Cryptanalysis”, 1994.

28. G. Leander, T. Moos, A. Moradi, and S. Rasoolzadeh. “The SPEEDY Family of Block Ciphers Engineering an Ultra Low-Latency Cipher from Gate Level for Secure Processor Architectures”. *IACR Trans. Cryptogr. Hardw. Embed. Syst.* 2021:4, 2021, pp. 510–545.

29. C. Lefevre, Y. Belkheyar, and J. Daemen. *Kirby: A Robust Permutation-Based PRF Construction*. Cryptology ePrint Archive, Paper 2023/1520. https://eprint.iacr.org/2023/1520. 2023.

30. A. Mehrdad, S. Mella, L. Grassi, and J. Daemen. “Differential Trail Search in Cryptographic Primitives with Big-Circle Chi: Application to Subterranean”. *IACR Trans. Symmetric Cryptol.* 2022:2, 2022, pp. 253–288.

31. S. Mella, J. Daemen, and G. Van Assche. “New techniques for trail bounds and application to differential trails in Keccak”. *IACR Trans. Symmetric Cryptol.* 2017:1, 2017, pp. 329–357. DOI: 10.13154/tosc.v2017.i1.329-357.

32. D. Shi, L. Hu, S. Sun, L. Song, K. Qiao, and X. Ma. “Improved linear (hull) cryptanalysis of round-reduced versions of SIMON”. *Sci. China Inf. Sci.* 60:3, 2017, 39101:1–39101:3.

33. R. P. Stanley. *Enumerative Combinatorics*. Vol. 1. Cambridge Studies in Advanced Mathematics 49. Cambridge University Press, Cambridge, NY, 2012. ISBN: 9781107015425 1107015421 9781107602625 1107602629.

34. L. Sun, W. Wang, and M. Wang. “MILP-aided bit-based division property for primitives with non-bit-permutation linear layers”. *IET Inf. Secur.* 14:1, 2020, pp. 12–20. DOI: 10.1049/IET-IFS.2018.5283.

35. Y. Todo. “Structural Evaluation by Generalized Integral Property”. In: *Advances in Cryptology - EUROCRYPT 2015 - 34th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Sofia, Bulgaria, April 26-30, 2015, Proceedings, Part I*. Ed. by E. Oswald and M. Fischlin. Vol. 9056. Lecture Notes in Computer Science. Springer, 2015, pp. 287–314.

36. Y. Todo and M. Morii. “Bit-Based Division Property and Application to Simon Family”. *IACR Cryptol. ePrint Arch.*, 2016, p. 285.

37. S. Wang, B. Hu, J. Guan, K. Zhang, and T. Shi. "MILP Method of Searching Integral Distinguishers Based on Division Property Using Three Subsets". *IACR Cryptol. ePrint Arch.*, 2018, p. 1186. URL: https://eprint.iacr.org/2018/1186.

7 Ciminion: Symmetric Encryption Based on Toffoli-Gates over Large Finite Fields

Christoph Dobraunig[1], Lorenzo Grassi[1], Anna Guinet[1], Daniël Kuijsters[1]

1 – Lamarr Security Research, Austria
2 – Graz University of Technology, Austria
3 – Radboud University, The Netherlands

My contributions. This chapter is based on work accepted at Eurocrypt 2021. I was responsible for the linear cryptanalysis of the cipher during the design phase, which led to a break of the original design. This was addressed by modifying the round constant addition layer. Additionally, I made significant contributions to the algebraic cryptanalysis of the cipher.

Abstract. Motivated by new applications such as secure Multi-Party Computation (MPC), Fully Homomorphic Encryption (FHE), and Zero-Knowledge proofs (ZK), the need for symmetric encryption schemes that minimize the number of field multiplications in their natural algorithmic description is apparent. This development has brought forward many dedicated symmetric encryption schemes that minimize the number of multiplications in $\mathbb{F}_{2^n}$ or $\mathbb{F}_p$, with p being prime. These novel schemes have lead to new cryptanalytic insights that have broken many of said schemes. Interestingly, to the best of our knowledge, all of the newly proposed schemes that minimize the number of multiplications use those multiplications exclusively in S-boxes based on a power mapping that is typically x^3 or x^{-1}. Furthermore, most of those schemes rely on complex and resource-intensive linear layers to achieve a low multiplication count.

In this paper, we present Ciminion, an encryption scheme minimizing the number of field multiplications in large binary or prime fields, while using a very lightweight linear layer. In contrast to other schemes that aim to minimize field multiplications in $\mathbb{F}_{2^n}$ or $\mathbb{F}_p$, Ciminion relies on the Toffoli gate to improve the non-linear diffusion of the overall design. In addition, we have tailored the primitive for the use in a Farfalle-like construction in order to minimize the number of rounds of the used primitive, and hence, the number of field multiplications as far as possible.

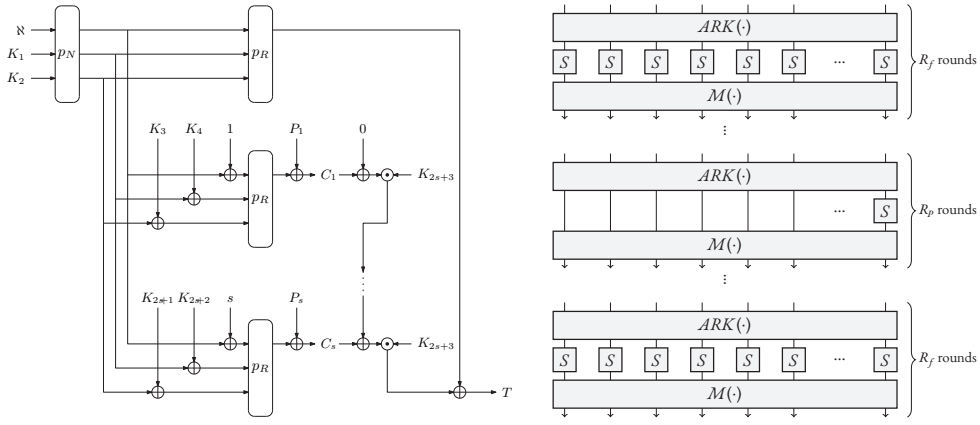
7.1 Introduction

Recently, several symmetric schemes have been proposed to reduce the number of field multiplications in their natural algorithmic description, often referred to as the multiplicative complexity. These ciphers fall into two main categories. The first one contains ciphers that minimize the use of multiplications in $\mathbb{F}_2$, for instance, Flip [53], Keyvrium [22], LowMC [4], and Rasta [33]. The second category is comprised of ciphers having a natural description in larger fields, which are mostly binary fields $\mathbb{F}_{2^n}$ and prime fields $\mathbb{F}_p$. Examples include MiMC [3], GMiMC [2], Jarvis [8], Hades [40], Poseidon [39] and Vision and Rescue [6]. The design of low multiplicative complexity ciphers is motivated by applications such as secure Multi-Party Computation (MPC), Fully Homomorphic Encryption (FHE), and Zero-Knowledge proofs (ZK). These recent ciphers based on specialized designs highly outperform "traditionally" designed ones in these applications. The search of minimizing the multiplicative complexity while providing a sufficient security level is an opportunity to explore and evaluate innovative design strategies.

The sheer number of potentially devastating attacks on recently published designs implies that the design of schemes with low multiplicative complexity has not reached a mature state yet. Indeed, we count numerous attacks on variants of LowMC [32, 58], Flip [34], MiMC [35], GMiMC [15, 19], Jarvis [1], and Starkad/Poseidon [15]. Attacks that are performed on schemes defined for larger fields mostly exploit weaknesses of the algebraic cipher description, e.g., Gröbner bases attacks on Jarvis [1] or higher-order differential attacks on MiMC [35]. Nonetheless, attack vectors such as differential cryptanalysis [17] and linear cryptanalysis [51] do not appear to threaten the security of these designs. Indeed, the latter two techniques seem to be able to attack only a tiny fraction of the rounds compared to algebraic attacks.

Interestingly, the mentioned ciphers working over larger fields are inspired by design strategies proposed in the 1990s to mitigate differential cryptanalysis. For example, MiMC resembles the Knudsen-Nyberg cipher [55], Jarvis claims to be inspired by the design of Rijndael [27, 28], while Hades, Vision, and Rescue take inspiration from Shark [59]. The latter ciphers have a linear layer that consists of the application of a single MDS matrix to the state. An important commonality between all those examples is a non-linear layer that operates on individual field elements, e.g., cubing single field elements or computing their inverse. Furthermore, design strategies naturally working over larger fields easily prevent differential cryptanalysis. However, algebraic attacks seem to be their main threat. Therefore, it is worth exploring different design strategies to increase the resistance against algebraic attacks.

Our Design: Ciminion. In that spirit, Ciminion offers a different design approach in which we do not apply non-linear transformations to individual field elements. Instead, we use the ability of the multiplication to provide non-linear diffusion between field elements. Our cipher is built upon the Toffoli gate [61], which is a simple non-linear bijection of field elements that transforms the triple (a, b, c) into the triple $(a, b, ab + c)$. The binary version of the Toffoli gate is used as a building block in modern ciphers, such as FRIET [60], which inspired our design. In addition to this, the S-box of Xoodoo [26] can also be described as the consecutive application of three binary Toffoli gates. With respect to the linear layer, we learned from ciphers like LowMC [4] that very heavy linear layers can have a considerably negative impact on the performance of applications [31]. Therefore, we decide to pair the Toffoli gate with



(a) Farfalle Construction (b) A Hades-like scheme.

Figure 7.1: Comparison of a Farfalle construction and a Hades-like scheme.

a relatively lightweight linear layer to construct a cryptographic permutation on triples of field elements. Compared to the designs that use a non-linear bijection of a single field element, e.g., cubing in $\mathbb{F}_{2^n}$ for odd n, we can define our permutation on any field, and then provide a thorough security analysis for prime fields and binary fields.

We do not use a bare primitive in the applications, but we employ primitives in a mode of operation. Indeed, instead of constructing a primitive of low multiplicative complexity, our goal is to provide a cryptographic function of low multiplicative complexity. We achieve this by using a modified version of the Farfalle construction to make it possible to perform stream encryption. Farfalle [12] is an efficiently parallelizable permutation-based construction with a variable input and output length pseudorandom function (PRF). It is built upon a primitive, and modes are employed on top of it. The primitive is a PRF that takes as input a key with a string (or a sequence of strings), and produces an arbitrary-length output. The Farfalle construction involves two basic ingredients: a set of permutations of a b-bit state, and the so-called rolling function that is used to derive distinct b-bit mask values from a b-bit secret key, or to evolve the secret state. The Farfalle construction consists of a *compression layer* that is followed by an *expansion layer*. The compression layer produces a single b-bit accumulator value from a tuple of b-bit blocks representing the input data. The expansion layer first (non-linearly) transforms the accumulator value into a b-bit rolling state. Then, it (non-linearly) transforms a tuple of variants of this rolling state which are produced by iterating the rolling function, into a tuple of (truncated) b-bit output blocks. Both the compression and expansion layers involve b-bit mask values derived from the master key.

We slightly modify Farfalle (see Figure 7.3) and instantiate it with two different permutations: p_C for the compression part, and p_E for the expansion part. Those two permutations are obtained by iterating the same round function, but with a different number of rounds. In our construction, the permutation p_C takes an input that is the concatenation of a nonce $\mathcal{N}$ and a secret key, and it derives a secret

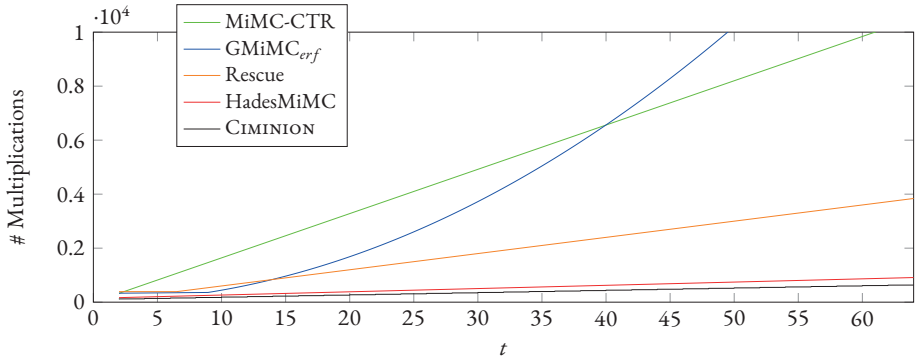


Figure 7.2: Number of MPC multiplications of several designs over $\mathbb{F}_p^t$, with $p \approx 2^{128}$ and $t \geq 2$ (security level of 128 bits).

intermediate state from this input. Then, the intermediate state is updated by using a simple rolling function, and fixed intermediate keys. From this intermediate state, the keystream for encrypting the plaintext is derived by using the permutation p_E. In order to prevent backward computation, the outputs of the expansion layers are truncated. Our security analysis that is presented in section 7.4 shows that p_E requires a significantly lower number of rounds than p_C. The relatively low number of multiplications that is used per encrypted plaintext element leads to a remarkably overall low multiplicative complexity. The full specification for Ciminion is presented in section 7.2. A detailed rationale of the choices made during the design process is given in section 7.3. A reference implementation can be found at https://github.com/ongetekend/ciminion.

A Concrete Use Case: Multi-Party Computation. The primary motivation of our design is to explore the limits on the use of non-linear operations in cipher design, while limiting the use of linear operations, and ensuring a secure design. The main body of our paper is thus dedicated to cryptanalysis which is accompanied by one specific use-case, namely Secure Multi-Party Computation.

MPC is a subfield of cryptography that aims to create methods for parties to jointly compute a function over their inputs, without exposing these inputs. In recent years, MPC protocols have converged to a linearly homomorphic secret sharing scheme, whereby each participant is given a share of each secret value. Then, each participant locally adds shares of different secrets to generate the shares of the sum of the secrets. In order to get data securely in and out of a secret-sharing-based MPC system, an efficient solution is to directly evaluate a symmetric primitive within such system. In this setting, "traditional" PRFs based on, e.g., AES or SHA-3 are not efficient. Indeed, they were designed with different computing environments in mind. Hence, they work over data types that do not easily match the possible operations in the MPC application. As developed in [42], "traditional" PRFs like AES and SHA-3 are rather bit/byte/word-oriented schemes, which complicate their representation using arithmetic in $\mathbb{F}_p$ or/and $\mathbb{F}_{2^n}$ for large integer n, or prime p.

From a theoretical point of view, the problem of secure MPC is strongly connected to the problem of masking a cryptographic implementation. This observation has been made in [44, 45]. The intuition behind is that both masking and MPC aim to perform computations on shared data. In more detail, the common strategy behind these techniques is to combine random and unknown masks with a shared secret value, and to perform operations on these masked values. Only at the end of the computation, the values are unmasked by combining them, in a manner that is defined by the masking scheme. In our scheme, we use a linear sharing scheme, because affine operations (e.g., additions, or multiplications with a constant) are non-interactive and resource efficient, unlike the multiplications that require some communication between the parties. The number of multiplications required to perform a computation is a good estimate of the complexity of an MPC protocol.

However, in practice, other factors influence the efficiency of a design. For instance, while one multiplication requires one round of communication, a batch of multiplications can be processed into a single round in many cases. In that regard, Ciminion makes it possible to batch several multiplications due to the parallel execution of p_E. Another alternative to speed up the processing of messages is to execute some communication rounds in an offline/pre-computation phase before receiving the input to the computation. This offline phase is cheaper than the online rounds. For example, in the case of Ciminion, precomputing several intermediate states is possible by applying p_C to different nonces $\mathcal{N}$. As a result, for the encryption of arriving messages, those intermediate states only have to be expanded, and processed by p_E to encrypt the plaintext.

section 7.5 demonstrates that our design Ciminion has a lower number of multiplications compared to several other schemes working over larger fields. The comparison of the number of multiplications in MPC applications to the ciphers that are presented in the literature, is shown in Figure 7.2, when working over a field $\mathbb{F}_p^t$ with $p \approx 2^{128}$ and $t \geq 1$, and with a security level of 128 bits (which the most common case in the literature). It indicates that our design needs approximately $t + 14 \cdot \lceil t/2 \rceil \approx 8 \cdot t$ multiplications compared to $12 \cdot t$ multiplications that are required by HadesMiMC, or $60 \cdot t$ multiplications that is needed by Rescue. These two schemes that have recently been proposed in the literature are our main competitors. Additionally, our design employs a low number of linear operations when compared with other designs present in the literature. Indeed, Ciminion grows linearly w.r.t. t, whereas the number of linear operations grows quadratically in HadesMiMC and Rescue. That is because their rounds are instantiated via the multiplication with a $t \times t$ MDS matrix. Even if the cost of a linear operation is considerably lower than the cost of a non-linear one in MPC applications, it is desirable to keep both numbers as low as possible. Our design has this advantage.

7.2 Specification

7.2.1 Mode

In order to create a nonce-based stream-encryption scheme, we propose to work with the mode of operation described in Figure 7.3. First, the scheme takes a nonce $\mathcal{N}$ along with two subkey elements K_1

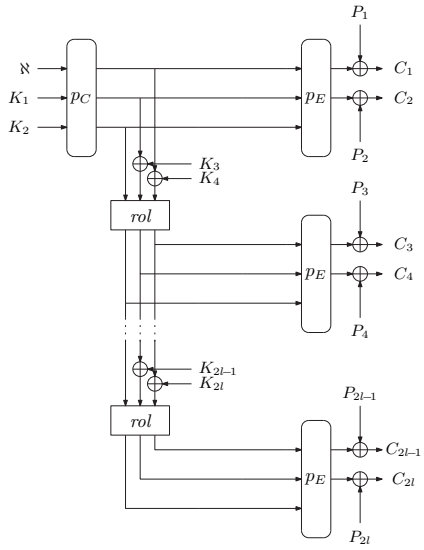


Figure 7.3: Encryption with Ciminion over $\mathbb{F}_{2^n}$. The construction is similar over $\mathbb{F}_p$ ($\oplus$ is replaced by +, the addition modulo p).

and K_2 as input, and processes these input with a permutation p_C to output an intermediate state. This intermediate state is then processed by a permutation p_E, and truncated to two elements so that two plaintext elements P_1 and P_2 can be encrypted. If more elements need to be encrypted, the intermediate state can be expanded by repeatedly performing an addition of two subkey elements to the intermediate state, then followed by a call to the rolling function *rol*. After each call to the rolling function *rol*, two more plaintext elements P_{2i} and P_{2i+1} can be encrypted thanks to the application of p_E to the resulting state. We consider the field elements as atomic, and therefore, our mode can cope with a different number of elements without the need for padding. The algorithmic description of the mode of operation that is described in Figure 7.3, is provided in App. I.

7.2.2 Permutations

We describe two permutations of the vector space $\mathbb{F}_q^3$. They act on a *state* of triples $(a, b, c) \in \mathbb{F}_q^3$. The first permutation is defined for a prime number $q = p$ of $\log_2(p) \approx d$ bits, while the second permutation is specified for $q = 2^d$. Both permutations are the result of the repeated application of a round function. Their only difference is the number of repeated applications that we call *rounds*. As presented in Figure 7.3, we employ two permutations p_C and p_E that have respectively N and R rounds.

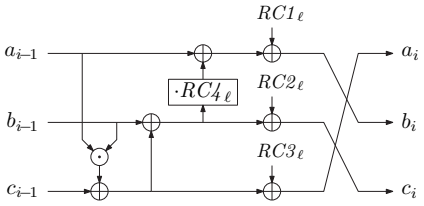


Figure 7.4: Round function f_i.

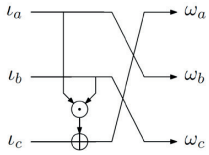


Figure 7.5: *rol*.

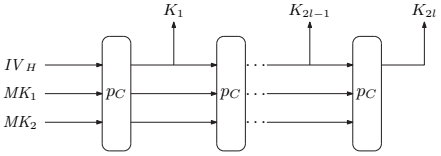


Figure 7.6: Key generation.

Round Function. We write f_i for round i. It uses four round constants RC_ℓ, with $\ell = i$ for p_C, and $\ell = i + N - R$ for p_E. We assume that $RC4_\ell \notin \{0, 1\}$. For each $i \geq 1$, f_i maps a state $(a_{i-1}, b_{i-1}, c_{i-1})$ at its input to the state (a_i, b_i, c_i) at its output, where the relation between these two states is

$$\begin{bmatrix} a_i \\ b_i \\ c_i \end{bmatrix} := \begin{bmatrix} 0 & 0 & 1 \\ 1 & RC4_\ell & RC4_\ell \\ 0 & 1 & 1 \end{bmatrix} \cdot \begin{bmatrix} a_{i-1} \\ b_{i-1} \\ c_{i-1} + a_{i-1} \cdot b_{i-1} \end{bmatrix} + \begin{bmatrix} RC3_\ell \\ RC1_\ell \\ RC2_\ell \end{bmatrix}.$$

7.2.3 The rolling function

Our rolling function *rol* is a simple NLFSR as depicted in Figure 7.5. The rolling function takes three field elements ι_a, ι_b, and ι_c at the input. It outputs three field elements: $\omega_a := \iota_c + \iota_a \cdot \iota_b$, $\omega_b := \iota_a$, and $\omega_c := \iota_b$. The latter variables form the input of the permutation p_E in our Farfalle-like mode Figure 7.3.

7.2.4 Subkeys and round constants

Subkeys Generation. We derive the subkey material K_i from two master keys MK_1, and MK_2. As a result, the secret is shared in a compact manner, while the expanded key is usually stored on a device, and used when needed. To expand the key, we use the sponge construction [14] instantiated with the permutation p_C. The value IV_H can be made publicly available, and is typically set to one.

Round Constants Generation. We generate the round constants $RC1_\ell$, $RC2_\ell$, $RC3_\ell$, and $RC4_\ell$ with Shake-256 [13, 54]. The detail is provided in App. A.

Table 7.1: Proposed number of rounds based on f. The security level s must satisfy $64 \leq s \leq \log_2(q)$, and $q \geq 2^{64}$, where q is the number of elements in the field.

Instance	p_C	p_E (two output words per block)
Standard	$s+6$	$\max\left\{\left\lceil\frac{s+37}{12}\right\rceil, 6\right\}$
Data limit $2^{s/2}$ elements	$\frac{2(s+6)}{3}$	$\max\left\{\left\lceil\frac{s+37}{12}\right\rceil, 6\right\}$
Conservative	$s+6$	$\max\left\{\left(\left\lceil\frac{3}{2}\cdot\frac{s+37}{12}\right\rceil\right), 9\right\}$

7.2.5 Number of rounds and security claim for encryption

In this paper, we assume throughout that *the security level of s bits satisfies the condition* $64 \leq s \leq \lfloor\log_2(q)\rfloor$. This implies that $q \geq 2^{64}$.

In Table 7.1, we define three sets of round numbers for each permutation in our encryption scheme:

- The "standard" set guarantees s bit of security; in the following sections, we present our security analysis that supports the chosen number of rounds for this case.
- For our MPC application, we propose a number of rounds if the data available to the attacker is limited to $2^{s/2}$; our security analysis that supports the chosen number of rounds for this case is presented in App. F.
- Finally, we present a "conservative" number of rounds where we arbitrarily decided to increase the number of rounds by 50% of the standard instance.

Since many cryptanalytic attacks become more difficult with an increased number of rounds, we encourage to study reduced-round variants of our design to facilitate third-party cryptanalysis, and to estimate the security margin. For this reason, it is possible to specify toy versions of our cipher, i.e., with $q < 2^{64}$ which aim at achieving, for example, only 32 bits of security.

7.3 Design rationale

7.3.1 Mode of operation

In order to provide encryption, our first design choice is to choose between a mode of operation that is built upon a block cipher or a cryptographic permutation. In either case, a datapath design is necessary. However, a block cipher requires an additional key schedule, unlike a cryptographic permutation. If a designer opts for a block cipher, the key schedule can be chosen to be either a non-linear, an affine, or a trivial transformation, where the round keys are equal to the master key apart from round constants. In this case, the designer has to be careful, because a poor key schedule leads to weaknesses and attacks [19]. Considering that the research in low multiplicative complexity ciphers is a relatively new research area, we decided to limit our focus to the essential components of a primitive. Therefore, we opted for permutation-based cryptography.

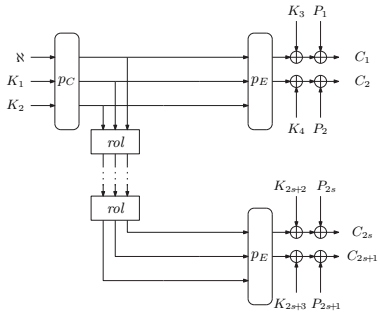


Figure 7.7: Intermediate step in constructing Figure 7.3

Since we consider the application of low multiplicative ciphers in areas that have enough resources to profit from parallel processing, we base our mode of operation on the Farfalle construction [12] as depicted in Figure 7.1a. The Farfalle construction is a highly versatile construction that provides many functionalities.

A Modified Version of Farfalle. As already mentioned in the introduction, our mode of operation resembles the Farfalle construction. In this section, we explain and support the modifications that we performed on the original Farfalle construction, as depicted in Figure 7.1a. The aim of those modifications is to both increase the resistance of the construction against algebraic attacks which are the most competitive ones in our scenario, and to increase its efficiency in our target application scenario, that is to say to minimize the number of multiplications. We focus first on the security aspect, before explaining in further detail how we reach our efficiency goal.

Our first modification is for simplicity. Since the functionality provided by the Farfalle construction to compress information is not needed, we merge p_c and p_d to a single permutation p_C.

Our second modification is to truncate the output. This prevents meet-in-the-middle style attacks that require the knowledge of the full output.

The third modification is to manipulate different keys K_i (see Figure 7.7) instead of employing the same key k' for each output block. Since we aim to have a permutation with a low degree, Gröbner bases are the main threat. For the scheme that is depicted in Figure 7.7, an attacker has to exploit equations of the form $f(x) + K_i = y$ and $f(x') + K_i = y'$, with $f(x) - f(x') = y - y'$ for a Gröbner basis attack. We describe this scenario in more detail in subsection 7.4.4.

Our last modification is to move the keys K_i from the output of p_E to the input of our rolling function, and hence, effectively to the input of p_E (Figure 7.3). Figure 7.3 is our final construction, and it provides two main benefits. First, having the keys at the input does not make it possible to easily cancel them by computing the difference of the output as described before. Hence, this adds an additional barrier in mounting successful Gröbner basis attacks. Second, we can use a simple non-linear rolling function, because the addition of the key stream during the rolling function prevents the attacker from easily detecting short cycles within it.

Minimizing the Number of Multiplications. One main reason to use the Farfalle construction is that its three permutations p_c, p_d, and p_e do not have to provide protection against all possible attack vectors. Indeed, the permutation p_e alone does not have to provide resistance against higher-order differential attacks [48, 49]. The latter are particular algebraic attacks that exploit the low degree polynomial descriptions of the scheme. Resistance against higher-order differential attacks (higher-order attacks in short) can be provided by the permutations p_c, and p_d, and it inherently depends on the algebraic degree that a permutation achieves. Hence, requiring protection against higher-order attacks provides a lower bound on the number of multiplications that are needed in a permutation. In a nutshell, since p_e does not have to be secure against higher-order attacks, we can use a permutation with fewer multiplications. This benefits the multiplication count of the scheme, since the permutations p_c and p_d are called only once independently of the number of output words.

The Rolling Function. An integral part of the Farfalle construction is the rolling function *rol*. The permutations p_c and p_e (Figure 7.1a) in the Farfalle construction are usually chosen to be very lightweight, such that the algebraic degree is relatively low. Hence, to prevent higher-order attacks, the rolling function is chosen to be non-linear. In our modified version, the same is true up to the intermediate construction as depicted in Figure 7.7. In this case, *rol* has to be non-linear in order to use a permutation p_E of low degree. For our final construction (Figure 7.3), we do not see any straightforward way to exploit higher-order attacks due to the unknown keys at the inputs of p_E. Thus, we could use a linear rolling function *rol*, but we rather choose to use a simple non-linear *rol* for Ciminion. That is because it makes it possible to analyze the security of Figure 7.7, and to keep the same conclusion when we opt for the stronger version of Figure 7.3. In addition, we present Aiminion in App. B, a version of our design that does not follow this line of reasoning. Aiminion uses a linear rolling function, and nine rounds of p_E. We deem this version to be an interesting target for further analysis that aims to evaluate the security impact of switching from a non-linear to a linear rolling function.

Generating the Subkeys. Instead of sharing all subkeys K_i directly by communicating parties to encrypt messages, we specify a derivation of the subkeys K_i from two master keys MK_1, and MK_2. These subkeys can be generated in a single precomputation step. For the storage of the subkeys, trade-offs can be made to store as many subkeys as needed, and to split messages into lengths that match the stored subkey lengths.

7.3.2 The round function

Our round function is composed of three layers: a non-linear transformation, a linear transformation, and a round constant addition. Like classical designs, we employ the same non-linear and linear transformations for each round, but with different round constant additions. This makes it easier to implement, and to reduce code-size and area requirements. Nonetheless, some primitives that have been designed to lower the multiplicative complexity use a different linear layer for each round, like in LowMC [4].

Non-linear Transformation. Most primitives operating in large fields have a variant of powering field elements, e.g., x^3 or x^{-1}. These mappings became popular to guard against linear and differential cryptanalysis due to their properties [55]. The most popular design that uses such mappings is the AES [28],

where x^{-1} is used as part of its S-box. For ciphers that aim at a low multiplicative complexity, these power mappings are interesting because they often have an inverse of high degree, which provides protection against algebraic attacks. However, they impose some restrictions, e.g., the map $x \mapsto x^\alpha$ for integer $\alpha \geq 2$ is a bijection in $\mathbb{F}_q$ if and only if $\gcd(q-1, \alpha) = 1$ (e.g., $x \mapsto x^3$ is a permutation over $\mathbb{F}_{2^n}$ for odd n only). Hence, one has to consider several power values α in order for x^α to stay a permutation for any field. In a design that should make it possible to be instantiated for a wide variety of fields, considering those special cases complicates the design of the cipher.

Instead of a power mapping, the non-linear element in our designs is the Toffoli gate [61]. Indeed, algebraic attacks are the main threat against designs aiming to lower the multiplicative complexity, and the multiplications are the main cost factor in our design. It thus seems counter intuitive to spend the non-linear element on simply manipulating a single field element, as is the case for power mappings. Therefore, we choose to multiply two elements of the state, instead of operating on a single state element, in order to increase the non-linear diffusion. Furthermore, the Toffoli gate is a permutation for any field, and therefore we are not restricted to a specific field. We mitigate potential negative effects of the property of the Toffoli gate to provide the same degree in forward and backward direction by mandating its use only in modes that truncate the permutation output, and that never evaluate its inverse using the secret key.

Linear Transformation. We present the linear transformation in its matrix form, the coefficients of which must be carefully chosen. One possibility is to use an MDS matrix. Since an MDS matrix has the highest branch number [24] among all possible matrices, it plays an important role in proving lower bounds on the linear and differential trail weight. However, we do not need to rely on MDS matrices as the field multiplications already have advantageous properties against linear and differential attacks.

Another option is to randomly choose the coefficients of the matrix for each round, and then verify that the matrix is invertible. This strategy was used in one of the first low multiplicative complexity designs, namely LowMC [4]. However, the drawback is that random matrices contribute significantly to the cost of the primitive in some scenarios, and the security analysis becomes more involved. Hence, we have decided to use a much simpler linear layer.

In order to provide sufficient diffusion, complex equation systems, and low multiplicative complexity, the degree of the functions that output equations depending on the input variables must grow as fast as possible. By applying a single multiplication per round, the degree doubles per round in the best scenario. However, this also depends on the linear layer. For instance, this layer could be a simple layer permuting the elements (e.g., the 3×3 circulant matrix $circ(0, 0, 1)$), for which the univariate degree of a single element only grows according to a Fibonacci sequence. To ensure that the univariate degree of a single element doubles per round, the result of the previous multiplication has to be reused in the

multiplication of the next round. This is also applicable to the inverse of the permutation. Hence, we decided to use the following matrix for the linear layer:

$$M = \begin{bmatrix} 0 & 0 & 1 \\ 1 & RC4 & RC4 \\ 0 & 1 & 1 \end{bmatrix} \qquad (\text{and} \qquad M^{-1} = \begin{bmatrix} 0 & 1 & -RC4 \\ -1 & 0 & 1 \\ 1 & 0 & 0 \end{bmatrix}),$$

Here, $M_{0,2}, M_{1,2}, {M^{-1}}_{0,2}, {M^{-1}}_{1,2} \neq 0$ with $M_{i,j}$ denoting the element of the matrix M at row i and column j. The use of the round constant $RC4 \notin \{0, 1\}$ is motivated by aiming to improve the diffusion, and to avoid a weakness with respect to linear cryptanalysis that we discuss in subsection 7.4.1.

About Quadratic Functions. In addition to the matrix multiplication, another (semi-)linear transformation[1] over a binary field $\mathbb{F}_{2^n}$ is the quadratic permutation $x \mapsto x^2$. This transformation can be exploited as a component in the round function (e.g., as a replacement of the multiplication by $RC4$) to both increase the diffusion and the overall degree of the function that describes the scheme. However, we do not employ it for several reasons. First, even if the quadratic permutation is linear over $\mathbb{F}_{2^n}$, its cost in an application like MPC might not be negligible. Indeed, the quadratic permutation costs one multiplication as detailed in [42]. As a result, even if it makes it possible to reduce the overall number of rounds due to a faster growth of the degree, the overall number of multiplications[2] would not change for applications like MPC. Secondly, the quadratic function is not a permutation over $\mathbb{F}_p$ for a prime $p \neq 2$. Thus, its introduction implies having to work with two different round functions: one for the binary case and one for the prime case. Since our goal is to present a simple and elegant general scheme, we decided not to use it.

Round Constants. The round constants break up the symmetry in the design. They prevent the simplification of the algebraic description of the round function. However, as we manipulate many round constants, and since they influence the rounds in a complex manner, we use an extendable output function to obtain round constant values without an obvious structure. We performed some experiments where we added round constants to one or two state elements. These instances provided simpler algebraic descriptions. Considering the small costs of manipulating dense round constants, we decide to use three round constants to complicate the algebraic description of the cipher, even after a few rounds.

7.4 Security analysis

We present our security analysis of Ciminion with respect to "standard" application of the attacks that are found in the literature. This analysis determines the required number of rounds to provide some level of confidence in its security. Due to page limitation, further analysis is presented in App. D-E.

[1] A function f over $(\mathbb{F}, +)$ is semi-linear if for each $x, y \in \mathbb{F}$: $f(x + y) = f(x) + f(y)$. It is linear if it is semi-linear and if for each $x \in \mathbb{F}$: $f(\alpha \cdot x) = \alpha \cdot f(x)$.

[2] A minimum number of multiplications is required to reach maximum degree, which is one of the property required by a cryptographic scheme to be secure.

First and foremost, the number of rounds that guarantees security up to s bits are computed under the assumption that the data available to the attacker is limited to 2^s, except if specified in a different way. Moreover, we do not make any claim about the security against related-key attacks and known- or chosen-key distinguishers (including the zero-sum partitions). The latter are out the scope of this paper.

We observe that the attack vectors penetrating the highest number of rounds are algebraic attacks. On the contrary, traditional attacks, such as differential and linear cryptanalysis, are infeasible after a small number of rounds. As detailed in the following, in order to protect against algebraic attacks and higher-order differential attacks, we increase the number of rounds proportionally to the security level s. A constant number of rounds is added to prevent an adversary from guessing part of the key or the initial or middle state, or to linearize part of the state. Hence, the numbers of rounds for p_C and p_E are respectively $s + 6$ and $\left\lceil \frac{s+19}{12} + 1.5 \right\rceil$ for the standard security level.

7.4.1 Linear cryptanalysis

Linear cryptanalysis [51] is a known-plaintext attack that abuses high *correlations* [25] between sums of input bits and sums of output bits of a cryptographic primitive. However, classical correlation analysis is not restricted to solely primitives operating on elements of binary fields. In this section, we apply the existing theory developed by Baignères et al. [9] for correlation analysis of primitives that operate on elements of arbitrary sets to the permutations defined in section 7.2.

General Correlation Analysis. An application of the theory to ciphers operating on elements of binary fields is presented by Daemen and Rijmen [29]. Classical correlation analysis is briefly recalled in App. C.1. In this section, we apply the theory to the more general case of primitives operating on elements of $\mathbb{F}_q$ where $q = p^d$. Henceforth, we suppose that $f : \mathbb{F}_q^l \to \mathbb{F}_q^m$.

Correlation analysis is the study of *characters*, and their configuration in the l-dimensional vector space $L^2(\mathbb{F}_q^l)$ of complex-valued functions $\mathbb{F}_q^l \to \mathbb{C}$. The space $L^2(\mathbb{F}_q^l)$ comes with the inner product $\langle g, h \rangle = \sum g(x)\overline{h(x)}$, which defines the norm $\|g\| = \sqrt{\langle g, g \rangle} = q^{\frac{l}{2}}$.

A character is an additive homomorphism from $\mathbb{F}_q^l$ into $S := \{z \in \mathbb{C} : |z| = 1\}$. It is well-known that *any* character on $\mathbb{F}_q^l$ is of the form

$$\chi_u(x) = e^{\frac{2\pi i}{p} \operatorname{Tr}_p^q(u^\top x)},$$

for some $u \in \mathbb{F}_q^l$. We recall that for $q = 2$ we have that $\chi_u(x) = (-1)^{u^\top x}$, which appears in classical correlation analysis. Here, $\operatorname{Tr}_p^q(x) = x + x^2 + \cdots + x^{p^{l-1}} \in \mathbb{F}_p$ is the *trace function*. For this reason, $u^\top x$ is called a *vectorial trace parity* and u a *trace mask vector*. We call the ordered pair (u, v) a linear approximation of f, where u is understood to be the mask at the input and v to be the mask at the output of f.

We define the vectorial trace parity correlation in the following definition.

Definition 58 (Correlation)**.**

$$\mathrm{C}_f(u, v) = \frac{\langle \mu_u, \mu_v \circ f \rangle}{\|\mu_u\| \|\mu_v \circ f\|} = \frac{1}{q^l} \sum_{x \in \mathbb{F}_q^l} e^{\frac{2\pi i}{p} \operatorname{Tr}_p^q(u^\top x - v^\top f(x))}$$

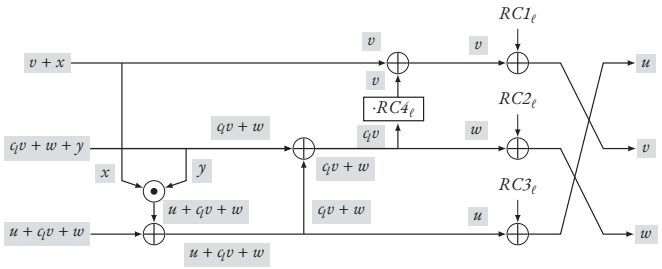


Figure 7.8: Mask propagation in f

This helps us to define a more general linear probability metric as follows.

Definition 59 (Linear probability)**.** $\mathrm{LP}_f(u, v) = |\,\mathrm{C}_f(u, v)|^2$

The idea is then to consider the permutation as a circuit made of simple building blocks. Those blocks correspond to the operators that we apply, and for which we attach to each edge a trace mask vector. Importantly, these trace mask vectors are in one-to-one correspondence with characters. The goal of the attacker is to construct a linear trail from the end of the permutation to the beginning, with the goal of maximizing the linear probability of each building block. A list of the linear probabilities of each such building block can be found in App. C.2 to deduce the result of the analysis.

On Three-round Linear Trails. Figure 7.8 illustrates how the linear masks propagate through the round function when the linear probabilities of all building blocks are maximized. In this Figure, $c_\ell := RC4_\ell$. The attacker is able to choose u, v, and w freely at the beginning of the first round, and afterwards, a mask at the input of the next round is determined by a mask at the output of the former round. We write R_i for the i'th round function. Moreover, we use the notation $c_{ij} := c_i c_j$ and $c_{ijk} := c_i c_j c_k$, where the subscript refers to the round number. The masks evolve as follows:

$$\begin{pmatrix} u \\ v \\ w \end{pmatrix} \xrightarrow{R_0} \begin{pmatrix} v \\ c_1 v + w \\ u + c_1 v + w \end{pmatrix} \xrightarrow{R_1} \begin{pmatrix} c_1 v + w \\ u + (c_1 + c_{12})v + (1 + c_2)w \\ u + (1 + c_1 + c_{12})v + (1 + c_2)w \end{pmatrix}$$
$$\xrightarrow{R_2} \begin{pmatrix} u + (c_1 + c_{12})v + (1 + c_2)w \\ (1 + c_3)u + (1 + c_1 + c_{12} + c_{13} + c_{123})v + (1 + c_2 + c_3 + c_{23})w \\ (1 + c_3)u + (1 + 2c_1 + c_{13} + c_{12} + c_{123})v + (2 + c_2 + c_3 + c_{23})w \end{pmatrix}.$$

An implicit assumption in both Figure 7.8, and the mask derivation above, is that the masks at the output of the multiplication and at the input of the third branch are equal. However, an attacker can only make sure that this assumption is valid if the following system of equations has a non-zero solution:

$$\begin{pmatrix} 1 & c_1 & 1 \\ 1 & 1 + c_1 + c_{12} & 1 + c_2 \\ 1 + c_3 & 1 + 2c_1 + c_{13} + c_{12} + c_{123} & 2 + c_2 + c_3 + c_{23} \end{pmatrix} \begin{pmatrix} u \\ v \\ w \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix}.$$

If we denote by A the matrix above, then this happens if and only if the matrix is singular, i.e., if $\det(A) = c_2c_3 + 1 = 0$. If either c_2 or c_3 is equal to zero, then the condition does not hold. If both are non-zero, then the condition is equivalent to requiring that $c_2 = -c_3^{-1}$. In this case, we can freely choose one value, which determines the other. Hence, the probability that the condition holds is equal to $\frac{q-1}{q^2} < \frac{1}{q}$. Since $\log_2(q)$ is the security parameter, this probability is negligible and there exists no three-round trail with a linear probability of 1.

Clustering of Linear Trails. We have $\mathrm{LP}_f(u, v) \geq \sum_{Q \in \mathrm{LT}_f(u,v)} \mathrm{LP}(Q)$, where $\mathrm{LT}_f(u, v)$ is the set of linear trails contained in (u, v). If we suppose now that an attacker is able to find more than q linear trails, i.e., if $\left|\mathrm{LT}_f(u, v)\right| > q$, then we have $\mathrm{LP}_f(u, v) > \frac{1}{q}$. However, $\log_2(q)$ is the security parameter, therefore the latter condition is not feasible. In a nutshell, three rounds are sufficient to resist against linear cryptanalysis.

Round Constant Multiplication Necessity. If the multiplication by the round constant is not present, or $RC4_\ell = 1$, then the masks evolve as follows over a single round:

$$\begin{pmatrix} u \\ v \\ w \end{pmatrix} \xrightarrow{f^{-1}} \begin{pmatrix} v + x \\ v + w + y \\ u + v + w \end{pmatrix} \xrightarrow{\text{if } u=v \text{ and } x=y=w=0} \begin{pmatrix} v \\ v \\ 0 \end{pmatrix} \xrightarrow{f^{-1}} \begin{pmatrix} v \\ v \\ 2v \end{pmatrix},$$

where (x, y) is the mask vector at the input of the multiplication function, which, like u, v, and w, can be freely chosen. Hence, if we choose $u = v$, and $x = y = w = 0$, and since the characteristic of the field is equal to two, then a one-round approximation with a linear probability of one can be chained indefinitely. This is the reason behind including a multiplication by a non-trivial constant.

7.4.2 Differential cryptanalysis

Differential cryptanalysis exploits the probability distribution of a non-zero input difference leading to an output difference after a given number of rounds [17]. As Ciminion is an iterated cipher, a cryptanalyst searches for ordered sequences of differences over r rounds that are called *differential characteristics/-trails*. A differential trail has a Differential Probability (DP). Assuming the independence of the rounds, the DP of a differential trail is the product of the DPs of its one-round differences (Definition 60).

Definition 60 (One-round differential probability)**.** *Let* $(\alpha_a, \alpha_b, \alpha_c) \in \mathbb{F}_p^3$ *be the input of the round, and* $(\alpha_a^*, \alpha_b^*, \alpha_c^*) \in \mathbb{F}_p^3$ *the chosen non-zero input difference. The probability that an input difference is mapped to an output difference* $(\beta_a^*, \beta_b^*, \beta_c^*) \in \mathbb{F}_p^3$ *through one iteration of the round function* f *is equal to*

$$\frac{|f(\alpha_a^* + \alpha_a, \alpha_b^* + \alpha_b, \alpha_c^* + \alpha_c) - f(\alpha_a, \alpha_b, \alpha_c) = (\beta_a^*, \beta_b^*, \beta_c^*)|}{|\mathbb{F}_p^3|}.$$

The operation $+$ *is replaced by* $\oplus$ *in* $\mathbb{F}_{2^n}$.

However, in general, the attacker does not have any information about the intermediate differences of the differential trail. Hence, the attacker only fixes the input and the output differences over r rounds, and works with *differentials*. A differential is a collection of differential trails with fixed input and output differences, and free intermediate differences. The DP of a differential over r rounds is the sum of all DPs of the differential trails that have the same input and output difference over the same number of rounds as the differential.

In this paper, we perform the differential cryptanalysis by grouping fixed differences in *sets*. Those sets impose some conditions to satisfy between the differences of the branches of the round, and/or specify that some differences at the input of the branches equal zero. Then, given an input difference, we study the possible sets of output differences after a round, and we determine the DP that an input difference is mapped into an output difference over a round. The goal is to find the longest differential trail with the highest DP.

Toward this end, we build a state finite machine (more details in App. C.3) that represents all the encountered sets of differences as states associated to their differential probabilities. To construct the graph, we start with a difference of the form $\{(0, 0, x) | x \neq 0\}$, and we search for the possible sets of output differences until we have explored all the possibilities from each newly reached set. Hereafter, let us assume that the difference x is not zero. We see that an input difference from $\{(0, 0, x)\}$ is mapped into an output difference of the form $\{(x, RC4_\ell x, x)\}$ after one round with probability one. Indeed, since the input difference goes through the non-linear operation and stays unchanged, the output difference is simply the result of the linear operation applied to the input difference. For the other cases, a non-zero input difference propagates to an output difference over one round with probability equal to p^{-1} in $\mathbb{F}_p$, or 2^{-n} in $\mathbb{F}_{2^n}$. From those results, we determine the differential over three rounds with the highest DP.

On Three-round Differentials. The differential trail in $\mathbb{F}_p$ with the highest DP is

$$\{(0, 0, x)\} \xrightarrow{\text{prob. } 1} \{(x, RC4_\ell x, x)\} \xrightarrow{\text{prob. } p^{-1}} \{(-RC4_\ell x, x, 0)\} \xrightarrow{\text{prob. } p^{-1}} \{(0, 0, x)\},$$

where the fixed input difference x is equal to another fixed value in the following rounds, and satisfies the conditions imposed by the set (for details see App. C.3). Additionally, this differential trail holds if and only if the round constant $RC4_\ell$ introduced by the first round is equal to the round constant $RC4_\ell$ of the third round.

In $\mathbb{F}_{2^n}$, we obtain almost the same state finite machine as in Figure 7.9. The only exception is that the set of differences $\{(-RC4_\ell x, x, 0)\}$ corresponds to $\{(RC4_\ell x, x, 0)\}$, because $-z$ is equal to z for each $z \in \mathbb{F}_{2^n}$. Hence, the differential trail in $\mathbb{F}_{2^n}$ with the highest DP is

$$\{(0,0,x)\} \xrightarrow{\text{prob. } 1} \{(x, RC4_\ell x, x)\} \xrightarrow{\text{prob. } 2^{-n}} \{(RC4_\ell x, x, 0)\} \xrightarrow{\text{prob. } 2^{-n}} \{(0,0,x)\},$$

under the same conditions that in $\mathbb{F}_p$.

In summary, a fixed difference from $\{(0,0,x)\}$ is mapped to the difference of the form $\{(x, RC4_\ell x, x)\}$ after one round with probability one in $\mathbb{F}_{2^n}$ and in $\mathbb{F}_p$. Moreover, as depicted in Figure 7.9, an input difference can be mapped to an output difference of the form $\{(0,0,x)\}$ with DP p^{-1} (resp. 2^{-n}) if and only if this difference is of the form $\{(-RC4_\ell x, x, 0)\}$. This means that the *only* possible differential trail over three rounds with input and output differences of the form $\{(0,0,x)\}$ are the ones given before. The DP of this differential trail is expressed in the following Proposition.

Proposition 31. *A differential trail over three rounds has a probability at most equal to p^{-2} in $\mathbb{F}_p$ and 2^{-2n} in $\mathbb{F}_{2^n}$.*

The DP of all other differential trails over three round are at most equal to p^{-3} in $\mathbb{F}_p$ and 2^{-3n} in $\mathbb{F}_{2^n}$. Since the security level s satisfies $s \leq \log_2(p)$ in $\mathbb{F}_p$ and $s \leq n$ in $\mathbb{F}_{2^n}$, we therefore conjecture that three rounds are sufficient to guarantee security against "basic" differential distinguishers. We thus choose to have at least six rounds for the permutations p_E and p_C, which is twice the number of rounds necessary to guarantee security against "basic" differential/linear distinguishers. The minimal number of rounds for the permutations should provide security against more advanced statistical distinguishers.

7.4.3 Higher-order differential and interpolation attacks

If a cryptographic scheme has a simple algebraic representation, higher-order attacks [48, 49] and interpolation attack [46] have to be considered. In this part, we only focus on higher-order differential attacks. We conjecture that the number of rounds necessary to prevent higher-order differential attacks is also sufficient to prevent interpolation attacks (see details in App. D). This result is not novel, and the same applies for other schemes, like MiMC, as further explained in [35].

Background. We recall from Figure 7.3 that an attacker can only directly manipulate a single element, and the two other elements are the secret subkeys. We therefore operate with this single element to input value sets, while keeping the two other elements fixed. Each output element is the result of a non-linear function depending on the input element x, and two fixed elements that are the input of the permutation. Thus, we have $f_N(x) = p(x, const, const)$ in $\mathbb{F}_{2^n}$, and $f_p(x) = p(x, const, const)$ in $\mathbb{F}_p$.

A given function f_p over prime fields $\mathbb{F}_p$ is represented by $f_p(x) = \sum_{i=0}^{p-1} \kappa_i x^i$ with constants $\kappa_i \in \mathbb{F}_p$. The degree of the function $f_p(x)$ that we denote by $d_{\mathbb{F}_p}$, corresponds to the highest value i for which $\kappa_i \neq 0$. The same holds for a function f_n working over binary extension fields $\mathbb{F}_{2^n}$. For the latter, $f_N(x) = \bigoplus_{i=0}^{d} \kappa_i x^i$ with $\kappa_i \in \mathbb{F}_{2^n}$, and d_{ff2^n} is the degree of the function $f_n(x)$. Like previously, the degree is the highest value i for which $\kappa_i \neq 0$. In $\mathbb{F}_{2^n}$, the function can as well be represented by its algebraic norm

form (ANF) $\overrightarrow{f_n}(x_1, \ldots, x_n)$, whose output element j is defined by its coordinate function $f_{n,j}(x_1, \ldots, x_n) = \bigoplus_{u=(u_1,\ldots,u_2)} \kappa_{j,u} \cdot x_1^{u_1} \cdot \ldots \cdot x_n^{u_n}$ with $\kappa_{j,u} \in \mathbb{F}_2$. The degree $d_{\mathbb{F}_2^n}$ of $\overrightarrow{f_n}$ corresponds to the maximal Hamming weight of u for which $\kappa_{j,u} \neq 0$, that is to say $d_{\mathbb{F}_2^n} = \max_{i \leq d}\{hw(i) \mid \kappa_i \neq 0\}$.

For the last representation, as proved by Lai [49] and in[48], if we iterate over a vector space $\mathcal{V}$ having a dimension strictly higher than $d_{\mathbb{F}_2^n}$, we obtain the following result: $\bigoplus_{v \in \mathcal{V} \oplus \nu} f_n(v) = 0$. A similar result has also been recently presented for the prime case in [35, Proposition 2]. More precisely, if the degree of $f_p(x)$ is $d_{\mathbb{F}_p}$, then iterating over all elements of a multiplicative subgroup $\mathcal{G}$ of $\mathbb{F}_p^t$ of size $|\mathcal{G}| > d_{\mathbb{F}_p}$ leads to $\sum_{x \in \mathcal{G}} f_p(x) = f_p(0) \cdot |\mathcal{G}|$. The last sum is equal to zero modulo p since $|\mathcal{G}|$ is a multiple of p.

In order to provide security against higher-order differential attacks based on the presented zero-sums, we choose the number of rounds of our permutation to have a function of a degree higher than our security claim.

Overview of our Security Argument. In our construction, we assume that an attacker can choose the nonce $\mathcal{N}$, which is the input of the permutation p_C. For the first call of this permutation, we want to prevent an attacker to input value sets that always result in the same constant after the application of the permutation p_C. This requirement is necessary, since we assume in the remaining analysis that the output values of p_C are unpredictable by an attacker. We emphasize that if the output of the permutation p_C is guaranteed to be randomly distributed, then this is sufficient to prevent higher-order differential attacks. That is because the inverse of the final permutations p_E is never evaluated, and the attacker cannot construct an affine subspace in the middle of the construction.

Estimating the Degree of p_C: Necessary Number of Rounds. We study the evolution of the degrees $d_{\mathbb{F}_p}$ and $d_{\mathbb{F}_{2^n}}$ for the permutation p_C for which the round function f (Figure 7.3) is iterated r times. We conclude that the degree of the permutation p_C remains unchanged for two rounds, if an input element is present at branch a, and the input at the branch b is zero. For a higher number of rounds, the degree increases. We have chosen the affine layer to ensure that the output of the multiplication can affect both inputs of the multiplication in the next round. This should make it possible for the maximal possible degree of the output functions to increase faster than having affine layers without this property. In the best case, the maximal degree of the function can be doubled per round.

Considering both previous observations, a minimum of $s + 2$ rounds are required to obtain at least $d_{\mathbb{F}_p} \approx 2^s$, or $d_{\mathbb{F}_{2^n}} \approx 2^s$. As we want to ensure that the polynomial representation of p_C is dense, it is then advisable to add more rounds as a safety margin. In order to reach this goal, we arbitrarily decided to add four more rounds.

7.4.4 Gröbner basis attacks

Preliminary. To perform a Gröbner basis [21] attack, the adversary constructs a system of algebraic equations that represents the cipher. Finding the solution of those equations makes it possible for the attacker to recover the key that is denoted by the unknown variables $x_1, ..., x_n$ hereafter. In order to solve this system of equations, the attacker considers the *ideal* generated by the multivariate polynomials that define the system. A *Gröbner basis* is a particular generating set of the ideal. It is defined with respect to

a total ordering on the set of monomials, in particular the lexicographic order. As a Gröbner basis with respect to the lexicographic order is of the form

$$\{x_1 - h_1(x_n), \dots, x_{n-1} - h_{n-1}(x_n), h_n(x_n)\},$$

the attacker can easily find the solution of the system of equations. To this end, one method is to employ the well-known Buchberger's criterion [21], which makes it possible to transform a given set of generators of the ideal into a Gröbner basis. From a theoretic point of view, state-of-the-art Gröbner basis algorithms are simply improvements to Buchberger's algorithm that include enhanced selection criteria, faster reduction step by making use of fast linear algebra, and an attempt to predict reductions to zero. The best well-known algorithm is Faugère's F5 algorithm [11, 36].

Experiments highlighted that computing a Gröbner basis with respect to the lexicographic order is a slow process. However, computing a Gröbner basis with respect to the grevlex order can be done in a faster manner. Fortunately, the FGLM algorithm [37] makes it possible to transform a Gröbner basis with respect to the grevlex order to another with respect to the lexicographic order. To summarize, the attacker adopts the following strategy:

1. Using the F5 algorithm, compute a Gröbner basis w.r.t. the grevlex order.
2. Using the FGLM algorithm, transform the previous basis into a Gröbner basis w.r.t. the lexicographic order.
3. Using polynomial factorization and back substitution, solve the resulting system of equations.

Henceforth, we consider the following setting: let K be a finite field, let $A = K[x_1, \dots, x_n]$ be the polynomial ring in n variables, and let $I \subseteq A$ be an ideal generated by a sequence of polynomials $(f_1, \dots, f_r) \in A^r$ associated with the system of equations of interest.

Cost of the F5 Algorithm. In the best adversarial scenario, we assume that the sequence of polynomials associated with the system of equations is *regular*.[3] In this case, the F5 algorithm does not perform any redundant reductions to zero.

Write $F_{A/I}$ for the *Hilbert-Series* of the algebra A/I and $H_{A/I}$ for its *Hilbert polynomial*. The *degree of regularity* D_{reg} is the smallest integer such that $F_{A/I}(n) = H_{A/I}(n)$ for all $n \geq D_{\text{reg}}$. The quantity D_{reg} plays an important role in the cost of the algorithm. If the ideal I is generated by a regular sequence of degrees $d_1, \dots, d_r$, then its *Hilbert series* equals $F_{A/I}(t) = \frac{\prod_{i=1}^{r}(1+t+t^2+\dots+t^{d_i-1})}{(1-t)^{n-r}}$. From this, we deduce that $\deg(I) = \prod_{i=1}^{r} d_i$, and $D_{\text{reg}} = 1 + \sum_{i=1}^{r}(d_i - 1)$.

The main result is that if $f_1, \dots, f_r$ is a regular sequence in $K[x_1, \dots, x_n]$, then computing a Gröbner basis with respect to the grevlex order using the F5 algorithm can be performed within

$$\mathcal{O}\left(\binom{n + D_{\text{reg}}}{D_{\text{reg}}}^{\omega}\right)$$

[3] A sequence of polynomials $(f_1, \dots, f_r) \in A^r$ is called a regular sequence on A if the multiplication map $m_{f_i} : A/\langle f_1, \dots, f_{i-1}\rangle \to A/\langle f_1, \dots, f_{i-1}\rangle$ given by $m_{f_i}([g]) = [g][f_i] = [gf_i]$ is injective for all $2 \leq i \leq r$.

operations in K, where $2 \leq \omega \leq 3$ is the matrix multiplication exponent.

Costs of Gröbner Basis Conversion and of Back Substitution. FGLM is an algorithm that converts a Gröbner basis of I with respect to one order, to a Gröbner basis of I with respect to a second order in $\mathcal{O}(n \deg(I)^3)$ operations in K. Finally, as proved in [38], the cost of factorizing a univariate polynomial in $K[x]$ of degree d over $\mathbb{F}_{p^n}$ for a prime p is $\mathcal{O}(d^3 n^2 + dn^3)$.

Number of Rounds. After introducing the Gröbner Basis attack, we analyze the minimum number of rounds that is necessary to provide security against this attack. However, we first emphasize that:

- there are several ways to set up the system of equations that describes the scheme. For instance, we could manipulate more equations, and thus more variables, of lower degree. Alternatively, we could work with less equations, and thus less variables, of higher degree. In addition, we could consider the relation between the input and the output, or between the middle state and the outputs, and so on. In the following, we present some of these strategies, that seem to be the most competitive ones;

- computing the exact cost of the attack is far from an easy task. As largely done in the literature, we assume that the most expensive step is the "F5 Algorithm". If the cost of such a step is higher than the security level, we conclude that the scheme is secure against the analyzed attack.

A Weaker Scheme. Instead of using the model that is described in Figure 7.3, we analyze a weaker model as illustrated in Figure 7.7. In the latter, the key is added after the expansion part, instead of before the rolling function application. This weaker model is easier to analyze, and makes it possible to draw a conclusion regarding the security of our scheme. Thus, *we conjecture that if the scheme proposed in Figure 7.7 is secure w.r.t. Gröbner Basis attack, then the scheme in Figure 7.3 is secure*. Indeed, in the scheme proposed in Figure 7.7, it is always possible to consider the difference between two or more texts to remove the final key addition. For instance, given $f(x) + K = y$ and $f(x') + K = y'$, it follows that $f(x) - f(x') = y - y'$. As a result, the number of variables in the system of equations to be solved remains constant independently of the number of considered outputs. However, in Figure 7.3, given $g(x + K) = y$ and $g(x' + K) = y'$, this is not possible except if $g(\cdot)$ is inverted. Nevertheless, since it is a truncated permutation, this does not seem feasible, unless the part of the output which is truncated is either treated as a variable (that results to have more variables than equations) or guessed by brute force (that results in an attack whose cost is higher than the security level, and $2^s \leq q$). Such consideration leads us to conjecture that the number of rounds necessary to make the scheme proposed in Figure 7.7 secure is a good indicator of the number of rounds necessary to make the scheme in Figure 7.3 secure as well.

Input-Output Relation. The number of rounds must ensure that the maximum degree is reached. Based on that, we do not expect that the relation that holds between the input and the output, makes it possible for the attacker to break the scheme. In particular, let N be the nonce, and k_1, k_2 be the secret keys. If we assume that a single word is output, then an equation of degree 2^r can be expressed between each input $(N, k_1, k_2) \in (\mathbb{F}_q)^3$, and the output $T \in \mathbb{F}_q$ with r the number of rounds. Hence, if there are

two different initial nonces, then the attacker has to solve two equations in two variables. In that case, $D_{reg} = 1 + 2 \cdot (2^r - 1) \approx 2^{r+1}$. The cost of the attack is thus lower bounded by $\left[\binom{2+2^{r+1}}{2^{r+1}}\right]^{\omega} \geq \left[\frac{(1+2^{r+1})^2}{2}\right]^{\omega} \geq 2^{2r+1}$, where $\omega \geq 2$. Consequently, $2^{2r+1} \geq 2^s$ if the total number of rounds is at least $\left\lceil \frac{s-1}{2} \right\rceil$ (e.g., 64 for $s = 128$). Since the number of rounds for p_C is $s + 6$, this strategy does not outperform the previous attacks as expected.

Finally, we additionally consider a strategy where new intermediate variables are introduced to reduce the degree of the involved polynomials. We concluded that this strategy does not reduce the solving time as it increases the number of variables.

Middle State-Output Relation. There is another attack strategy that exploits the relation between the middle state and the outputs. In this strategy, only p_E is involved, and several outputs are generated by the same unknown middle state. For a given nonce N, let $(x_0^N, x_1^N, x_2^N) \in (\mathbb{F}_q)^3$ be the corresponding middle state. Since the key is added after the permutation p_E, we first eliminate the key by considering two initial nonces, and taking the difference of the corresponding output. This makes it possible to remove all the secret key material at the end, at the cost of having three more unknown variables in the middle.[4]

Hence, independently of the number of outputs that are generated, there are six variables, and thus simply the two middle states. That means that we need at least six output blocks, and an equivalent number of equations. Since two words are output for each call of p_E, we have six equations of degree 2^{r-1} and 2^r for the first two words, 2^r and 2^{r+1} for the next two words, and so on. We recall that every call of the rolling function increases the degree by a factor two, while the function that describes the output of a single block has a maximum degree, namely 2^r after r rounds for one word, and 2^{r-1} for the other two words. Hence, $D_{reg} = 1 + (2^{r-1} - 1) + 2 \cdot \sum_{i=0}^{1} (2^{r+i} - 1) + (2^{r+2} - 1) = 21 \cdot 2^{r-1} - 5 \approx 2^{r+3.4}$, and the cost of the attack is lower bounded by

$$\left[\binom{6 + 2^{r+3.4}}{2^{r+3.4}}\right]^{\omega} \geq \left[\frac{(1 + 2^{r+3.4})^6}{6!}\right]^{\omega} \geq 2^{12(r+3.4)-19},$$

where $\omega \geq 2$. Therefore, $2^{12(r+3.4)-19} \geq 2^s$ if the number of rounds for p_E is at least $\left\lceil \frac{s+19}{12} - 3.4 \right\rceil$ (e.g., 9 for $s = 128$). Like previously, potential improvement of the attack (e.g., an enhanced description of the equations) can lead to a lower computational cost. We thus decided to arbitrarily add five rounds as a security margin. We conjecture that at least $\left\lceil \frac{s+19}{12} + 1.5 \right\rceil$ rounds for p_E are necessary to provide some security (e.g., 14 for $s = 128$).

In addition, in order to reduce the degree of the involved polynomials, we studied the consequences of introducing new intermediate variables in the middle, e.g., at the output of the rolling function or among

[4] Another approach would be to involve the keys in the analysis. However, since the degree of the key-schedule is very high, the cost would then explode after few steps. It works by manipulating the degree of the key-schedule, or by introducing new variables for each new subkeys while keeping the degree as lower as possible. This approach does not seem to outperform the one described in the main text.

the rounds[5]. In that regard, we did not improve the previous results. Moreover, we also considered a scenario in which the attacker accesses more data, without being able to improve the previous results.

7.4.5 On the algebraic cipher representation

Algebraic attacks seem to be the most successful attack vector on ciphers that have a simple representation in larger fields, while restricting the usage of multiplications. Until now, we have mainly focused on the growth of the degree to estimate the costs of the algebraic attacks that we considered. However, this is not the only factor that influences the cost of an algebraic attack. It is well known that such attacks (including higher-order, interpolation, and Gröbner basis attacks) can be more efficient if the polynomial that represents the cipher is sparse. Consequently, it is necessary to study the algebraic representation of the cipher for a feasible number of rounds.

To evaluate the number of monomials that we have for a given degree, we wrote a dedicated tool. This tool produces a symbolic evaluation of the round function without considering a particular field or specific round constants. Nevertheless, it considers the fact that each element in $\mathbb{F}_{2^n}$ is also its inverse with respect to the addition. Since we do not instantiate any field and constants, the reported number of monomials might deviate from the real number of monomials here, e.g., due to unfortunate choices of round constants that sum to zero for some monomials. As a result, the entries in the tables are in fact upper bounds, but we do not expect high discrepancies between the numbers reported in the tables and the "real" ones.

Prime Case. First, we consider iterations of the round function f over $\mathbb{F}_p$. In Table 7.2, we evaluate the output functions at a_i, b_i, and c_i depending on the inputs a_0, b_0, and c_0 after a certain number of rounds $i \geq 2$. We count in Table 7.2 the number of monomials for a certain multivariate degree up to a fixed degree $d_{\mathbb{F}_p}$. Higher degree monomials might appear, but they are not presented in the table. To report this behavior, we do not input 0 in the table after the highest degree monomial. The column 'max' indicates the maximal number of monomials that can be encountered for three variables. As reported in Table 7.2, the number of monomials increases quite quickly, and we do not observe any unexpected behavior, or missing monomials of a certain degree.

Binary Case. Table 7.3 provides the number of monomials of a certain degree in $\mathbb{F}_{2^n}$. We notice that the diffusion is slower than in $\mathbb{F}_p$, and it may be because of the behavior of the addition that is self inverse in $\mathbb{F}_{2^n}$. More discussions on the algebraic cipher representation in the binary case can be found in App. D.

7.5 Comparison with other Designs

In this section, we compare the performance of our design with other designs that are presented in the literature for an MPC protocol using masked operations. We mainly focus on the number of multiplications in an MPC setting, which is often the metric that influences the most the cost in such a protocol. In

[5] For example, new variables can be introduced for each output of the rolling state. It results in having more equations with lower degrees. Our analysis suggests that this approach does not outperform the one described in the main text.

Table 7.2: Number of monomials of a certain degree for $\mathbb{F}_p$.

	Output	Degree																										
Round	Variable	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27
	max	1	3	6	10	15	21	28	36	45	55	66	78	91	105	120	136	153	171	190	210	231	253	276	300	325	351	378	406
2	a	1	3	4	3	1																							
	b	1	3	4	3	1																							
	c	1	3	4	3	1																							
3	a	1	3	6	8	11	8	6	3	1																			
	b	1	3	6	8	11	8	6	3	1																			
	c	1	3	6	8	11	8	6	3	1																			
4	a	1	3	6	10	15	19	24	28	33	28	24	19	15	10	6	3	1											
	b	1	3	6	10	15	19	24	28	33	28	24	19	15	10	6	3	1											
	c	1	3	6	10	15	19	24	28	33	28	24	19	15	10	6	3	1											
5	a	1	3	6	10	15	21	28	36	45	53	62	70	79	87	96	104	113	104	96	87	79	70	62	53	45	36	28	21
	b	1	3	6	10	15	21	28	36	45	53	62	70	79	87	96	104	113	104	96	87	79	70	62	53	45	36	28	21
	c	1	3	6	10	15	21	28	36	45	53	62	70	79	87	96	104	113	104	96	87	79	70	62	53	45	36	28	21

Table 7.3: Number of monomials of a certain degree for $\mathbb{F}_{2^n}$.

	Output	Degree																										
Round	Variable	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27
	max	1	3	6	10	15	21	28	36	45	55	66	78	91	105	120	136	153	171	190	210	231	253	276	300	325	351	378	406
2	a	1	3	4	2	1																							
	b	1	3	4	2	1																							
	c	1	3	4	2	1																							
3	a	1	3	6	7	7	3	3	0	1																			
	b	1	3	6	7	7	3	3	0	1																			
	c	1	3	6	7	7	3	3	0	1																			
4	a	1	3	6	9	15	14	19	12	13	5	6	2	3	0	0	0	1											
	b	1	3	6	9	15	14	19	12	13	5	6	2	3	0	0	0	1											
	c	1	3	6	9	15	14	19	12	13	5	6	2	3	0	0	0	1											
5	a	1	3	6	9	15	18	28	28	39	35	41	36	39	24	26	16	19	9	10	7	9	3	3	0	3	0	0	0
	b	1	3	6	9	15	18	28	28	39	35	41	36	39	24	26	16	19	9	10	7	9	3	3	0	3	0	0	0
	c	1	3	6	9	15	18	28	28	39	35	41	36	39	24	26	16	19	9	10	7	9	3	3	0	3	0	0	0

addition, we discuss the number of online and pre-computation/offline rounds, and we compare those numbers to the ones specified for other schemes. The influence of the last two metrics on the overall costs highly varies depending on the concrete protocol/application, and the concrete environment, in which an MPC protocol is used, e.g., network of computers vs. a system on chip. Finally, we consider the advantages and the disadvantages of our design w.r.t. the other ones.

7.5.1 MPC costs: Ciminion & related works

We compare the MPC cost of Ciminion with the cost of other designs that are published in the literature with $q \approx 2^{128}$, and $s = 128$ bits. We assume that the amount of data available to the attacker is fixed to $2^{s/2} = 2^{64}$, which is the most common case. Due to page limitation, we limit our analysis to Ciminion and HadesMiMC. The latter is the main competitive design currently present in the literature for the analyzed application. The detailed comparison with other designs (including MiMC, GMiMC, Rescue and Vision) is provided in App. G. A summary of the comparison is given in Table 7.4 and 7.5 for the binary and prime case, respectively.

Our design has the lowest minimum number of multiplications w.r.t. all other designs, in both $\mathbb{F}_p$ and $\mathbb{F}_{2^n}$. In $\mathbb{F}_q^t$ for $q \approx 2^{128}$, our design needs approximately $t + 14 \cdot \lceil t/2 \rceil \approx 8 \cdot t$ multiplications w.r.t. $12 \cdot t$ multiplications required by HadesMiMC or $60 \cdot t$ by Rescue. Additionally, our design has a low number of linear operations compared to other designs. For instance, for large $t \gg 1$, our design needs approximately $50 \cdot t$ affine operations (sums and multiplications with constants) while HadesMiMC

Table 7.4: Comparison on the MPC cost of schemes over $\mathbb{F}_{2^n}{}^t$ for $n = 128$ (or 129), and a security level of 128 bits. With the exception of Vision (whose number of offline rounds is equal to $\max\left\{20, 2 \cdot \left\lceil \frac{136+t}{t} \right\rceil\right\}$), the number of offline rounds for all other schemes is zero.

Scheme	Multiplications (MPC)		Online Rounds
	element in $\mathbb{F}_{2^n}{}^t$	asymptotically ($t \gg 1$)	
Ciminion	$8 \cdot t + 89$	8	$104 + \lceil t/2 \rceil$
MiMC-CTR	$164 \cdot t$	164	82
Vision	$t \cdot \max\left\{70, 7 \cdot \left\lceil \frac{136+t}{t} \right\rceil\right\}$	70	$\max\left\{50, 5 \cdot \left\lceil \frac{136+t}{t} \right\rceil\right\}$

Table 7.5: Comparison on the MPC cost of schemes over $\mathbb{F}_p{}^t$ for $p \approx 2^{128}$, and a security level of ≈ 128 bits. With the exception of Rescue (whose number of offline rounds is equal to $\max\{30; 6 \cdot \left\lceil \frac{32.5}{t} \right\rceil\}$), the number of offline rounds for all other schemes is zero.

Scheme	Multiplications (MPC)		Online Rounds
	element in $\mathbb{F}_p{}^t$	asymptotically ($t \gg 1$)	
Ciminion	$14 \cdot \lceil t/2 \rceil + t + 89$	8	$104 + \lceil t/2 \rceil$
MiMC-CTR	$164 \cdot t$	164	82
GMiMC$_{erf}$	$4 + 4t + \max\{4t^2, 320\}$	$4 \cdot t$	$2 + 2t + \max\{2t^2, 160\}$
Rescue ($\alpha = 3$)	$t \cdot \max\{60; 12 \cdot \left\lceil \frac{32.5}{t} \right\rceil\}$	60	$\max\{20; 4 \cdot \left\lceil \frac{32.5}{t} \right\rceil\}$
HadesMiMC	$12t + \max\{78 + \lceil \log_3(t^2) \rceil; 142\}$	12	$\max\{45 + \lceil \log_3(t) \rceil; 77\}$

requires approximately $12 \cdot t^2 + (157 + 4 \cdot \max\{32; \lceil \log_3(t) \rceil\}) \cdot t$ affine operations. However, this advantage comes at the price of having more online rounds than the other schemes. In particular, $104 + \lceil t/2 \rceil$ online rounds are required by our design whereas HadesMiMC and Rescue have respectively 78 and 20 online rounds.

Ciminion. For $q \approx 2^{128}$, and a security level of 128 bits with data limited to 2^{64}, the permutation p_C counts 90 rounds. In order to output $2t' - 1 \leq t \leq 2t'$ words, we call t' times the permutation p_E that is composed of 14 rounds, and $(t' - 1)$ times the rolling function. Therefore, for the binary and the prime case, the cost of Ciminion in MPC applications to generate t words is

$$\begin{aligned} \#\text{ multiplications:} &\quad 14 \cdot \lceil t/2 \rceil + (t - 1) + 90 \approx 8 \cdot t + 89\,, \\ \#\text{ online rounds:} &\quad 104 + \lceil t/2 \rceil\,, \\ \#\text{ affine operations:} &\quad 99 \cdot \lceil t/2 \rceil + 629 \approx 50 \cdot t + 629\,. \end{aligned}$$

The number of online rounds depends on t, because the rolling function is serial. It is noteworthy that the expansion part can be performed in parallel. We emphasize that the number of sums and multiplications with a constant[6] (denoted as "affine" operations) is proportional to the number of multiplications. That is one of the main differences w.r.t. to the Hades construction as we argue afterwards.

[6] Each round counts six additions and one multiplication with a constant.

HadesMiMC. HadesMiMC [40] is a block cipher that is proposed over $\mathbb{F}_p^t$ for a prime p such that $\gcd(p-1,3) = 1$, and $t \geq 2$. It combines $R_F = 2R_f$ rounds with a full S-box layer (R_f at the beginning, and R_f at the end), and R_P rounds with a partial S-box layer in the middle. Each round is defined with $R_i(x) = k_i + M \times S(x)$, where M is a $t \times t$ MDS matrix, and S is the S-box layer. This layer is defined as the concatenation of t cube S-boxes in the rounds with full layer, and as the concatenation of one cube S-Box and $t-1$ identity functions in the rounds with partial layer.

In addition, hash functions can be obtained by instantiating a Sponge construction with the Hades permutation, and a fixed key, like Poseidon & Starkad [39]. In [15], the authors present an attack on Starkad that exploits a weakness in the matrix M that defines the MixLayer. The attack takes advantage of the equation $M^2 = \mu \cdot I$. This attack can be prevented by carefully choosing the MixLayer (we refer to [43] for further detail). There is no attack that is based on an analogous strategy that has been proposed for the cipher[7].

In order to guarantee some security, R_F and R_P must satisfy a list of inequalities [40]. There are several combinations of (R_F, R_P) that can provide the same level of security. In that regard, authors of [40] present a tool that makes it possible to find the best combination that guarantees security, and minimizes the computational cost. For a security level of approximately $\log_2(p)$ bits, and with $\log_2(p) \gg t$, the combination (R_F, R_P) minimizing the overall number of multiplications is

$$(R_F, R_P) = \left(6, \max\left\{\left\lceil \frac{\log_3(p)}{2} \right\rceil + \lceil \log_3(t) \rceil; \lceil \log_3(p) \rceil - 2\lfloor \log_3(\log_2(p)) \rfloor\right\} - 2\right).$$

In MPC applications ($p \approx 2^{128}$ and $s = 128$ bits), the cost of HadesMiMC is

$$\begin{aligned}
\text{\# multiplications:} \quad & 2 \cdot (t \cdot R_F + R_P) = 12t + \max\{78 + \lceil \log_3(t^2) \rceil; 142\}, \\
\text{\# online rounds:} \quad & R_F + R_P = \max\{45 + \lceil \log_3(t) \rceil; 77\}, \\
\text{\# affine operations:} \quad & 2 \cdot t^2 \cdot R_F + (4 \cdot R_P + 1) \cdot t - 2 \cdot R_P \\
& \approx 12 \cdot t^2 + (157 + 4 \cdot \max\{32; \lceil \log_3(t) \rceil\}) \cdot t.
\end{aligned}$$

Parallel S-boxes can be computed in a single online round[8]. To compute the number of affine operations, we considered an equivalent representation of the cipher in which the MixLayer of the rounds, with a partial S-box layer, is defined by a matrix. In this matrix, only $3t-2$ entries are different from zero, that is to say the ones in the first column, in the first row , and in the first diagonal. (A $(t-1) \times (t-1)$ submatrix is an identity matrix.) The details are presented in [40, App. A]. Therefore, the total number of affine operations required grows quadratically w.r.t. the number of rounds with full S-box layer, and thus w.r.t. the number of multiplications.

[7] The main problem, in this case, regards the current impossibility to choose texts in the middle of the cipher by bypassing the rounds with full S-Box layer when the secret key is present.

[8] We refer to [42] on how to evaluate $x \to x^3$ within a single communication round.

Finally, we highlight that the number of multiplications is minimized when HadesMiMC takes as input the entire message. Indeed, let us assume that the input message is split into several parts, and that HadesMiMC is used in CTR mode (as suggested by the designers). In the analyzed case in which the security level is of the same order of the size of the field p, the number of rounds is almost constant, and independent of the parameter $t \geq 2$. It follows that using HadesMiMC in CTR mode would require more multiplications, because every process requires the computation of the rounds with a partial S-box layer, whereas this computation is needed only once when the message size equals the block size. We stress that a similar conclusion holds for Rescue/Vision, for which the total number of multiplications would barely change when they are used in CTR mode, rather than when the message size is equal to the block size.

7.5.2 Ciminion versus Hades: advantages and similarities

The previous comparison highlights that the two most competitive designs for MPC applications with a low multiplicative complexity are Ciminion and HadesMiMC. Referring to Fig. 7.1, we further develop the similarities and advantages between a block cipher based on a Hades design, and a cipher based on Farfalle. We present a brief comparison between our new design and the "ForkCipher" design that is proposed in [7] in App. G.2.

Similarities: Distribution of the S-Boxes. We focus our attention on the *distribution of the S-boxes, or more generally, the non-linear operations.* Both strategies employ a particular parallelization of the non-linear operations/S-boxes to their advantage, in order to minimize the number of non-linear operations. More precisely, each step is composed of t parallel non-linear operations in the external rounds, i.e., the rounds at the end and at the beginning. Furthermore, each step is composed of a single non-linear operation in the internal rounds.

Both strategies take advantage of an attacker that cannot *directly* access the state in the middle rounds, because the state is masked both by the external rounds or phases, and by the presence of a key. In a Farfalle design, the attacker knows that each output of the expansion phase always employs the same value at the input, without accessing those inputs. In a Hades design, the attacker is able to skip some rounds with a partial S-box layer by carefully choosing the texts (see [15]). However, they cannot access the texts without bypassing the rounds with the full S-box layer that depends on the key.

Having middle rounds with a single S-box makes it possible to reduce the overall number of non-linear operations. In addition, they ensure some security against algebraic attacks. Indeed, even a single S-box makes it possible to increase the overall degree of the scheme. For a concrete example, let (R_c, R_m, R_e) be the rounds for respectively the compression part, middle part and expansion part of Farfalle. Like previously, let (R_F, R_P) be the number of rounds with respectively a full and a partial S-box layer in Hades. The number of multiplications is respectively $(R_c + R_e) \cdot t + R_m$ and $R_F \cdot t + R_P$. If $R_P \gg R_F$ and $R_m \gg R_c + R_e$. For a similar number of round, i.e., proportional to $\approx R_P + R_F$ or/and $\approx R_m + R_c + R_e$, it is then necessary to reach the maximum degree. Our number of multiplications is lower compared to a classical design where the rounds have a full S-box layer.

Advantages. There are major differences between Farfalle-like designs and Hades-like designs, because of their primary intention. The Farfalle-like design aims to behave like a Pseudo-Random Function (PRF), and the Hades-like design like a Pseudo-Random Permutation (PRP). The latter is used as a PRF in the Counter mode (CTR).[9] Under the assumption that affine operations are cheaper than non-linear ones, designers of Hades defined the MixLayer as the multiplication with a $t \times t$ MDS matrix. Consequently, each round with full S-box layer counts t^2 multiplications with constants. However, when $t \gg 1$, linear operations cannot be considered as free anymore, and their presences influence the overall performance.

This problem is not present in a Farfalle-like design. Indeed, by construction, in the first R_c and the last R_e rounds, the MixLayer is not required. That implies that the first three words are never mixed with the following ones. On the contrary, the elements are simply added together to generate the input of the compression phase. In addition, the expansion part's input is generated through a non-linear rolling function whose cost grows linearly with t. Finally, since invertibility is not required, the number of input words can be lower than the number of output words to design a function from $\mathbb{F}_q^3$ to $\mathbb{F}_q^r$ for any $t \geq 1$. Thus, independently of the number of output words, one multiplication per round is present in the compression phase, contrary to $\mathcal{O}(t)$ of a Hades-like scheme.

Acknowledgements. We thank Joan Daemen for his guidance and support and the reviewers of Eurocrypt 2021 for their valuable comments that improved the paper. This work has been supported in part by the European Research Council under the ERC advanced grant agreement under grant ERC-2017-ADG Nr. 788980 ESCADA, the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (grant agreement No 681402), and the Austrian Science Fund (FWF): J 4277-N38.

A Round constants generation – details

As mentioned in the main part, the round constants $RC1_\ell$, $RC2_\ell$, $RC3_\ell$, and $RC4_\ell$ are generated using Shake-256 [13, 54]. We detail this process in this section.

For prime fields, a byte sequence of the ASCII characters "`GF(p)`" is absorbed with `p` denoting the numerical representation of the prime modulus. For instance, if we take the prime field 17, "`GF(17)`" is absorbed which hexadecimal representation is `0x474628313729`. The output sequence of Shake-256 is then split into $\lceil \log_2(p) \rceil$-bit unsigned integers Z_i. These values Z_i are next sequentially assigned to $RC1_\ell$, $RC2_\ell$, $RC3_\ell$, and $RC4_\ell$ for rising ℓ, as long as $1 < Z_i < p$. Otherwise, we discard Z_i, and we use instead the next unsigned integer $1 < Z_i < p$ of the sequence.

The round-constants generation process is analogous for fields over $\mathbb{F}_{2^n}$. In this case, we absorb a byte sequence corresponding to the ASCII characters "`GF(2)[X]/polynomial`", where the characters "`polynomial`" is the hexadecimal representation in capital letters of the irreducible polynomial. For example, in $\mathbb{F}_{2^8}$ with the irreducible polynomial $x^8 + x^4 + x^3 + x + 1$, we absorb "`GF(2)[X]/11B`" that

[9] This means that, in both cases, the cost of encryption and decryption is the same. That is because Farfalle-like and Hades-like designs are used as stream ciphers.

is represented by `0x47462832295B585D2F313142` in hexadecimal. Thereafter, the output sequence of Shake-256 is split into n-bit unsigned integers Z_i. These values Z_i are then sequentially assigned to $RC1_\ell$, $RC2_\ell$, $RC3_\ell$, and $RC4_\ell$ for rising ℓ as long as $Z_i > 1$. Otherwise, we discard Z_i, and we employ instead the next unsigned integer $Z_i > 1$ of the sequence.

B Aiminion: An aggressive evolution of Ciminion

For Ciminion, we modify the Farfalle [12] construction, in order to obtain a stronger design with a fewer successful attacks. In particular, moving the keys from the output of the construction (Figure 7.7) to the inputs of p_E (Figure 7.3) results in the two following observations.

1. The unknown keys K_i at the inputs of p_E prevent an attacker from knowing which inputs of p_E form an affine space. Hence, the rolling function *rol* does not have to be non-linear to achieve this property.

2. We cannot use the middle state-output relation (see subsection 7.4.4) to set up a system of equations to be solved using Gröbner bases. In fact, despite our attempts (not involving p_C), the system of equations is always under-determined. We have less equations than secret elements.

This leads us to Aiminion. Compared to Ciminion, we use the identity as rolling function *rol*, and we fix the number of rounds to nine for p_E, if we solely consider statistical attacks as a threat. This is three times the number of rounds where only bad differential/linear trails exists, as discussed in Table 7.6.

Table 7.6: Proposed number of rounds for Aiminion. The security level s must satisfy $64 \leq s \leq \log_2(q)$ and $q \geq 2^{64}$, where q is the number of elements in the field.

Instance	p_C	p_E (two words per block)
Data limit $2^{s/2}$ elements	$\frac{2(s+6)}{3}$	9

Aiminion has an lower number of multiplication per elements than Ciminion. Indeed, it shifts from eight for large t to only 4.5. However, this comes at the cost of interrupting the chain of arguments for security. In particular, probabilistically formed affine spaces at the inputs of p_E might still be detectable. We consider the evaluation of the complexity to find such an affine space as an interesting topic for future evaluation.

C Statistical attacks – details

C.1 Classical correlation analysis

In this section, we present the notions of classical correlation analysis in a fashion that eases the transition to the more general theory. In classical correlation analysis, we consider vectorial Boolean functions $f : \mathbb{F}_2^d \to \mathbb{F}_2^d$. From a more abstract point of view, classical correlation analysis is the study of certain

characters, and their configuration in the space $L^2(\mathbb{F}_2^d)$ of complex-valued functions $\mathbb{F}_2^d \to \mathbb{C}$. Let $S = \{z \in \mathbb{C} : |z| = 1\}$ be the multiplicative group of complex numbers of absolute value 1. An additive character of $\mathbb{F}_2^d$ is a homomorphism from $\mathbb{F}_2^d$ (considered as an abelian group) into S. The $L^2(\mathbb{F}_2^d)$-inner product is given by

$$\langle \chi, \psi \rangle = \frac{1}{2^d} \sum_{x \in \mathbb{F}_2^d} \chi(x)\overline{\psi(x)}.$$

A *parity* of a vector $x \in \mathbb{F}_2^d$ is the sum of a specific subset of its components. Any parity of x can be written as $u^\top x$ for some $u \in \mathbb{F}_2^d$. We call u a *mask*. Each mask u defines a unique character $\chi_u : \mathbb{F}_2^d \to \mathbb{C} \cap \{-1, 1\}$ given by

$$\chi_u(x) = (-1)^{u^\top x}.$$

Together, these notions lead to the definition of correlation.

Definition 61 (Parity Correlation)**.** *The correlation between an input mask u and output mask v with respect to a function f is defined as*

$$\mathrm{C}_f(u, v) = \langle \chi_u, \chi_v \circ f \rangle = \frac{1}{2^d} \sum_{x \in \mathbb{F}_2^d} (-1)^{u^\top x + v^\top f(x)}.$$

The number of known plaintext-ciphertext pairs required to mount a linear attack is inversely proportional to the square of the correlation. Hence, the square of the correlation serves as a measure to assess the effectiveness of a linear attack. It is called the *linear probability*.

Definition 62 (Linear Probability)**.** *The linear probability of an input mask u and output mask v with respect to a function f is defined as*

$$\mathrm{LP}_f(u, v) = \mathrm{C}_f(u, v)^2.$$

C.2 Proofs of linear probabilities

Proposition 32 (Orthogonality Relations)**.** *Let χ and ψ be additive characters of $\mathbb{F}_q$, then*

$$\sum_{x \in \mathbb{F}_q} \chi_b(x)\overline{\chi_c(x)} = \begin{cases} 0 & \text{if } b \neq c, \\ q & \text{otherwise.} \end{cases}$$

Proposition 33 (Linear Probabilities of the Multiplication Function)**.** *Let $m : \mathbb{F}_q \times \mathbb{F}_q \to \mathbb{F}_q$ be given by $m(x, y) = xy$, then the linear probabilities for all masks $u \in \mathbb{F}_q^2$ at the input of m and masks $v \in \mathbb{F}_q$ at the output of m are as seen in Table 7.7.*

u_1	u_2	v	$\mathrm{LP}_m((u_1,u_2),v)$
$*$	$*$	1	$\frac{1}{q^2}$
0	0	0	1
0	1	0	0
1	0	0	0
1	1	0	0

Table 7.7: Linear probabilities of m in $\mathbb{F}_q$. A 0 denotes a zero value, a 1 denotes a non-zero value, and a $*$ denotes any value.

Proof. Let $v \neq 0$. If $u_1 = 0$, then

$$\mathrm{LP}_m(u,v) = \left| \frac{1}{q^2} \sum_{x\in\mathbb{F}_q} \sum_{y\in\mathbb{F}_q} \chi_1(u_1x + u_2y - vxy) \right|^2 = \left| \frac{1}{q^2} \sum_{x\in\mathbb{F}_q} \sum_{y\in\mathbb{F}_q} \chi_1(u_2y - vxy) \right|^2$$
$$= \left| \frac{1}{q^2} \sum_{x\in\mathbb{F}_q} \sum_{y\in\mathbb{F}_q} \chi_1(u_2y)\overline{\chi_1(vxy)} \right|^2 = \frac{1}{q^2},$$

where the last equality is due to the fact that – by Proposition 32 – the inner sum equals q, if $x = v^{-1}u_2$, and 0 otherwise.

The case for which $u_2 = 0$ is analogous to the previous one. Therefore, we can suppose that $u_1 \neq 0$, and $u_2 \neq 0$. Let $a \in \mathbb{F}_q$, and $b \in \mathbb{F}_q$, be such that

$$b + u_1x + u_2y - vxy = (1 + ax)(b + u_2y).$$

Then,

$$\mathrm{LP}_m(u,v) = \left| \frac{1}{q^2} \sum_{x\in\mathbb{F}_q} \sum_{y\in\mathbb{F}_q} \chi_1(u_1x + u_2y - vxy) \right|^2.$$

By multiplying it by $|\chi_1(b)|^2 = 1$:

$$= \left| \frac{1}{q^2} \sum_{x\in\mathbb{F}_q} \sum_{y\in\mathbb{F}_q} \chi_1(b + u_1x + u_2y - vxy) \right|^2 = \left| \frac{1}{q^2} \sum_{x\in\mathbb{F}_q} \sum_{y\in\mathbb{F}_q} \chi_1((1 + ax)(b + vy)) \right|^2.$$

Note that $a \neq 0$ and $v \neq 0$, hence:

$$= \left| \frac{1}{q^2} \sum_{x'\in\mathbb{F}_q} \sum_{y'\in\mathbb{F}_q} \chi_1(x'y') \right|^2 = \frac{1}{q^2},$$

where the last equality is due to the fact that – by Proposition 32 – the inner sum equals q, if $x' = 0$, and 0 otherwise.

Next, let us consider that $v = 0$. From Proposition 32, we deduce that

$$\mathrm{LP}_m(u, 0) = \begin{cases} 1 & \text{if } u = (0, 0), \\ 0 & \text{otherwise.} \end{cases}$$

□

Proposition 34 (Linear Probabilities of the Round Constant Addition)**.** *Let $c : \mathbb{F}_q \to \mathbb{F}_q$ be given by $c(x) = x + c$, then*

$$\mathrm{LP}_c(u, v) = \begin{cases} 1 & \textit{if } u = v, \\ 0 & \textit{otherwise,} \end{cases}$$

for all masks $u, v \in \mathbb{F}_q$.

Proof.

$$\begin{aligned} \mathrm{LP}_c(u, v) &= \left| \frac{1}{q} \sum_{x \in \mathbb{F}_q} \chi_1(ux - vc(x)) \right|^2 = \left| \frac{1}{q} \sum_{x \in \mathbb{F}_q} \chi_1(ux - v(x + c)) \right|^2 \\ &= \left| \frac{1}{q} \sum_{x \in \mathbb{F}_q} \chi_1((u - v)x - vc) \right|^2 = \left| \frac{1}{q} \sum_{x \in \mathbb{F}_q} \chi_1((u - v)x) \overline{\chi_1(vc)} \right|^2 \\ &= \left| \frac{1}{q} \sum_{x \in \mathbb{F}_q} \chi_1((u - v)x) \right|^2 . \end{aligned}$$

The result now follows from Proposition 32. □

Proposition 35 (Linear Probabilities of the Addition)**.** *Let $a : \mathbb{F}_q \times \mathbb{F}_q \to \mathbb{F}_q$ be given by $a(x, y) = x + y$, then*

$$\mathrm{LP}_a(u, v) = \begin{cases} 1 & \textit{if } u = (v, v), \\ 0 & \textit{otherwise,} \end{cases}$$

for all masks $u \in \mathbb{F}_q^2$ at the input of a, and masks $v \in \mathbb{F}_q$ at the output of a.

Proof.

$$\begin{aligned}
\mathrm{LP}_a(u,v) &= \left| \frac{1}{q^2} \sum_{x\in\mathbb{F}_q} \sum_{y\in\mathbb{F}_q} \chi_1(u^\top(x,y) - va(x,y)) \right|^2 \\
&= \left| \frac{1}{q^2} \sum_{x\in\mathbb{F}_q} \sum_{y\in\mathbb{F}_q} \chi_1(u_1x + u_2y - v(x+y)) \right|^2 \\
&= \left| \frac{1}{q^2} \sum_{x\in\mathbb{F}_q} \sum_{y\in\mathbb{F}_q} \chi_1((u_1-v)x + (u_2-v)y) \right|^2 \\
&= \left| \frac{1}{q^2} \sum_{x\in\mathbb{F}_q} \chi_1((u_1-v)x) \sum_{y\in\mathbb{F}_q} \chi_1((u_2-v)y) \right|^2 .
\end{aligned}$$

By Proposition 32, both sums are non-zero if and only if $u_1 = u_2 = v$. In this case, they both evaluate to q. □

Proposition 36 (Linear Probabilities of the Branch Duplication)**.** *Let $d : \mathbb{F}_q \to \mathbb{F}_q \times \mathbb{F}_q$ be given by $d(x) = (x, x)$, then*

$$\mathrm{LP}_d(u,(v_1,v_2)) = \begin{cases} 1 & \text{if } u = v_1 + v_2, \\ 0 & \text{otherwise,} \end{cases}$$

for all masks $u \in \mathbb{F}_q$ at the input of d, and mask $(v_1, v_2) \in \mathbb{F}_q^2$ at the output of d.

Proof. Remember that $\mathrm{LP}_d(u,(v_1,v_2)) = \left| \frac{1}{q} \sum_{x\in\mathbb{F}_q} \chi_u(x)\overline{\chi_{v_1+v_2}(x)} \right|^2$. By Proposition 32, the sum is non-zero if and only if $u = v_1 + v_2$. In this case, it evaluates to q. □

Proposition 37 (Linear Probabilities of a $\mathbb{F}_q$-Linear Transformation.)**.** *Let $L : \mathbb{F}_q^m \to \mathbb{F}_q^m$ be a linear transformation, then*

$$\mathrm{LP}_L(u,v) = \begin{cases} 1 & \text{if } u = L^\top v, \\ 0 & \text{otherwise,} \end{cases}$$

for all masks $u, v \in \mathbb{F}_q^m$.

Proof.

$$\begin{aligned}
\mathrm{LP}_L(u,v) &= \left| \frac{1}{q} \sum_{x\in\mathbb{F}_q} \chi_1(u^\top x - v^\top Lx) \right|^2 = \left| \frac{1}{q} \sum_{x\in\mathbb{F}_q} \chi_1(u^T x - (L^\top v)^\top x) \right|^2 \\
&= \left| \frac{1}{q} \sum_{x\in\mathbb{F}_q} \chi_1((u - L^\top v)^\top x) \right|^2 .
\end{aligned}$$

The result now follows from Proposition 32. □

C.3 Differential attacks – details

As mentioned in subsection 7.4.2, we built a state finite machine in Figure 7.9 that represents all sets of encountered differences as states that are associated to their differential probabilities. We focus on the prime case, but the results are analogous in $\mathbb{F}_{2^n}$.

In order to understand the Figure, we mention that:

- all entries are fixed differences, where $(x, y, z) \neq 0$, with $x \neq y$, $x \neq z$, and $z \neq y$.

- in order to simplify the Figure, we use only three letters x, y, z to denote the differences for each branch. Most of the time, the value of the differences x, y, z at the input are *not* equal to the value of the differences x, y, z at the output. For instance, if we consider the one-round difference $\{(x, RC4x, x)\} \mapsto \{(0, x, y)\}$, the value of the input difference x is usually not equal to the value of the output difference x. Only in a few cases, the values of the differences x, y, z at the input are exactly equal to the value of the differences x, y, z at the output. This is the case for $\{(0, 0, x)\} \mapsto \{(x, RC4x, x)\}$ in $\mathbb{F}_{2^n}$.

- all arrows with a plain line represents a differential probability p^{-1} in $\mathbb{F}_p$ (similarly, 2^{-n} for the analogous case in $\mathbb{F}_{2^n}$); except for the arrow with a doted line that has a probability one.

- an arrow starting with a diamond and ending with a hollow head indicates that the round constant $RC4$ that is mentioned in the states before and after the considered round ($\mapsto$), comes from a previous round. In other words, if we consider the round ℓ, the round constant $RC4_j$ that is written in the difference states before/after this round, is introduced in a round $j < \ell$. In addition, it is possible that the round constant $RC4_\ell$ of the considered ℓ-th round is equal to the round constant of the previous or preceding state(s). This highly determines the output state of $\{(-RC4x, x, 0)\}$. For instance, if we examine the round ℓ that has the input differences $\{(-RC4_{j<\ell}x, x, 0)\}$, the output differences can be either $\{(x, RC4_{j<\ell}x, y)\}$ or $\{(x, y, z)\}$ depending on whether $RC4_{j<\ell} = RC4_\ell$, if none of the output differences is equal to zero.

D On interpolation attacks

In our keyed mode that is depicted in Figure 7.3, the subkeys K_i are derived from two master key elements MK_1 and MK_2. We analyze in this section how an attacker can gather many equations with the secrets K_1 and K_2 for the permutation p_C. As explained for higher-order differential attacks, the goal is to guarantee that the output of p_C appears to be randomly generated. For the sake of simplicity, we neglect in these observations any further additions of key elements and application of p_E. Moreover, we study the upper bounds on the number of monomials that is needed to solve a system of equations.

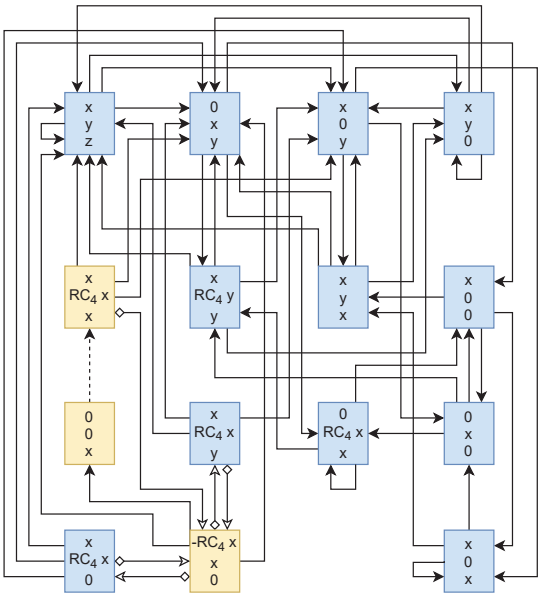


Figure 7.9: Differential trails for the round function in $\mathbb{F}_p$. The three-round differential trail with the highest DP from subsection 7.4.2, is highlighted in yellow.

An upper bound on the number of monomials.

We know from subsection 7.4.3 that the number of rounds required to reach a degree $d_{\mathbb{F}_p}$, or $d_{\mathbb{F}_{2^n}}$, of approximately 2^s is $s+2$ for either K_1 and K_2. Hence, the maximum number of possible monomials μ of the maximal possible combinations of K_1 and K_2 up to degree 2^s is 2^{2^s}. This means that if the diffusion is decent, the equations containing the secret variables are likely to have at least 2^s monomials.

The number of rounds that provides resistance against higher-order differential attacks should also guarantee sufficient protection against interpolation attacks [46] which use trivial linearization of all monomials. Indeed, this attack strategy aims to construct a polynomial corresponding to the encryption function without any knowledge of the secret key. If an adversary can construct such an interpolation polynomial without using the full code book, then they can potentially employ it to set up, e.g., a key-recovery attack.

In order to set up the system of equations that describes the scheme, the attacker first needs more than 2^s inputs/outputs, which exceeds the security level. In the following, let us assume that the attacker can collect 2^s of such equations. In order to solve this system of equations, the adversary could substitute each monomial by another variable in order to linearize the system. The cost of this attack is of $O(2^{s\omega})$ field operations with $\omega > 2$. However, that is behind the security level, if the chosen number of rounds guarantees security against the higher-order differential attack.

For completeness, we mention that the interpolation attack has a meet-in-the middle variant that significantly reduces the number of monomials in the system of equations which describes our rounds. However, in our construction that is illustrated in Figure 7.3, this variant is difficult to perform because the output is always truncated. Hence, part of the output changes while remaining secret. Each collected equation thus adds new secret variables to the system of equations. In that respect, we do not think that an meet-in-the-middle style approach would work for a trivial linearization. Nonetheless, equation solving approaches might succeed, like Gröbner bases that we discuss in subsection 7.4.4, but it is not restricted to it. Therefore, we study the equation systems obtained after a few rounds in subsection 7.4.5 to reveal any special exploitable structures in the equation systems. For example, we search for sparse equations.

Algebraic attack (over $\mathbb{F}_{2^n}$): case $RC4_i = 1$.

The behavior of our round function with the round constants $RC4_i$ set to one is presented in Table 7.8. If we compare the number of monomials in Table 7.8 with Table 7.3, the number of monomials in the Table 7.8 is clearly lower. Furthermore, if we consider the monomials of degree three until the round six, we only reach eight out of ten possible monomials, whereas we have all ten monomials in $\mathbb{F}_p$ (Table 7.2). A deeper study reveals that out of all possible monomials of degree three, $a_0 b_0 c_0$ and c_0^3 are missing. The first one is lacking due to the use of the Toffoli gate, and hence, the value c_0 is amended to $a_0 b_0 + c_0$ at the very beginning. In addition, c_0^3 is missing due to the interaction in the linear part. Hence, we observe that the polynomial is more sparse (or equivalently, less dense) than when $RC4_i \neq 1$.

E Other attack vectors and details

In this section, we discuss attack vectors that are not directly covered by the previous sections. These attack vectors mostly include attacks exploiting strongly aligned round functions, but also distinguishers only applicable to the permutation.

Truncated and impossible differential attacks.

A variant of differential cryptanalysis is the truncated differential cryptanalysis [48]. In the latter, the attacker does not fix the values of the differences, but specifies conditions between the differences of the branches of the round that should be satisfied, or fixes some differences at the input of the branches to zero. The attacker works with truncated differential characteristics, which are a collection/set of several differential characteristic.

Impossible differential cryptanalysis was introduced by Biham *et al.* [16] and Knudsen [47]. It exploits differentials that occur with probability zero.

Regarding our scheme, we do not expect these two attacks to outperform the attacks that are presented in section 7.4. As an example, a truncated differential with probability one covering one round can be used as starting point to present an impossible differential for two rounds:

$$\{(0,0,x)\} \xrightarrow{\text{prob. 1}} \{(x, RC4_\ell x, x)\} \neq \{(0,0,y)\} \xleftarrow{\text{prob. 1}} \{(y, RC4_\ell y, y)\},$$

where $(x, y) \neq 0$.

Zero-sum distinguishers.

Zero-sum distinguishers [20] are so-called inside-out distinguishers on the permutation. With this type of distinguishers, an attacker crafts an initial structure that is placed somewhere in the middle of the permutation, and computes forward to the input and backward output of the permutation. An attacker can compute a set of input and output values that sums to zero by carefully selecting and employing the initial structure, and choosing the algebraic degree of the round function and the inverse of the round function.

To prevent the use of such distinguishers against our round function, we need at least $2s$ rounds. We are fairly confident that this distinguishing property is unlikely to extend towards distinguisher of the used modes. For instance, zero-sum distinguisher on *full* Keccak-p [20] are well known. On the contrary, attacks on constructions that use 12 rounds of the Keccak permutation, and that exploit this property are not known. This is half the number of rounds.

Boomerang and differential-linear distinguishers.

Boomerang [62] and differential-linear [50] distinguishers, and their variants, rely on chaining two good differential/linear trails. As studied in section 7.4, we have at least one active multiplication per three rounds in our trail. Those distinguishers are then rather unlikely. However, even if an attacker can find good differentials, or linear hulls, a differential-linear/boomerang distinguisher can only cover up to six rounds.

Zero-correlation attacks.

As their name suggest, these attacks exploit linear hulls with a zero correlation [18]. In general, those linear hulls are found by a miss-in-the-middle approach. For example, we need to find two trails that propagate some deterministic properties, and then to combine them, in order to ensure that the property cannot be fulfilled. However, our permutations have at least nine rounds, and based on the results of our differential and linear analysis, we assume that finding impossible differentials or zero-correlation linear hulls is infeasible.

More attacks exploiting strong alignment.

As a cipher working natively on larger field elements, Ciminion could be considered to be strongly aligned. Many more attack vectors exist that exploit strong alignment on ciphers, especially for AES (see [41]). However, we conjecture that such attacks become quickly infeasible, because the security level is tied to the size of the field element, and there is a huge number of rounds compared to the number of field elements that form the state.

F Security analysis – data limit $2^{s/2}$

For applications like MPC, given a security level of s bits, the data is limited to $2^{s/2}$ (namely, related to the birthday bound) instead of 2^s. For this reason, we analyze in this section the number of rounds that is required to provide security, if limited data is available to the attacker. We focus our attention on algebraic attacks, which are the most powerful ones against our cipher. In the following, we demonstrate that a lower number of rounds, w.r.t. the ones given previously, is sufficient to provide security. Additionally, we verify that the chosen number of rounds guarantees security against statistical attacks.

Interpolation attack.

The amount of data that is available to the attacker highly impacts the interpolation attack. Indeed, this attack can be set up if the number of texts is higher than or equal to the number of monomials that defines the polynomial. Since the number of monomials is related to the degree of the polynomial, a lower number of rounds (w.r.t. the one given previously) is sufficient to prevent this attack. In particular, assuming that the polynomial that describes the scheme is dense, since the degree grows as 2^r after r rounds, approximately $s/2$ rounds are sufficient to prevent the attack (for data limited to $2^{s/2}$). We arbitrarily decided to increase the number of rounds for p_C to $2/3 \cdot s$ to provide some extra security against this attack.

Gröbner basis attack.

The Gröbner basis attack that is described in the previous sections, requires a few number of texts that are much lower than $2^{s/2}$. Hence, the analysis that is presented in subsection 7.4.4, can also be applied in this case. The number of rounds for p_C must thus be higher than $s/2$, while the number of rounds for p_E must be equal to $\left\lceil \frac{s+19}{12} + 1.5 \right\rceil$.

Statistical attacks.

Since the efficiency of statistical attacks depends on the amount of data that is available to the attacker, a lower number of rounds w.r.t. the standard one is still sufficient to guarantee security. We arbitrarily decided in this case to keep the same number of rounds as in the standard scenario, namely at least six rounds.

Conclusion.

We conjecture that $2/3 \cdot (s+6)$ rounds for p_C, and $\left\lceil \frac{s+19}{12} + 1.5 \right\rceil$ rounds for p_E, are sufficient to provide a security level of s bits, if the amount of data available to the attacker is limited to $2^{s/2}$. Due to argument analogous the ones given in section 7.4, this number of rounds provides security against statistical attacks, and higher-order differential attack.

G Related works: MPC costs for several ciphers published in the literature

G.1 Related works

MiMC.

MiMC [3] is a scheme that has been proposed over $\mathbb{F}_q$, where q is either a prime p, or a power of 2 $q = 2^n$, where $\gcd(p-1, 3) = 1$ or n odd. The round function of the block cipher is defined as

$$R_i(x) = x^3 \oplus k \oplus c_i \qquad \text{or} \qquad R_i(x) = x^3 + k + c_i\,,$$

for a round constant c_i, and a master key k. In $\mathbb{F}_q^{\,t}$, the cipher MiMC can be used in CTR-mode.

The number of rounds is equal to $\lceil \log_3(p) \rceil$, or $\lceil n \cdot \log_3(2) \rceil$. In MPC application, the cost to evaluate a text in $\mathbb{F}^t$ is thus given by

$$\begin{aligned} \#\text{ multiplications:} &\qquad 2t \cdot \lceil \log_3(p) \rceil\,, \\ (\#\text{ online}, \#\text{ offline}) \text{ rounds:} &\qquad (\lceil \log_3(p) \rceil, 0)\,, \end{aligned}$$

for both the binary and the prime case (it is sufficient to replace p with 2^n). We refer to [42] for a detailed explanation about the possibility to evaluate $x \to x^3$ with a single communication round. Moreover, evaluating $x \to x^3$ requires two multiplications in MPC applications in the binary case.

We make some observations. First of all, in a "classical" application, the number of multiplications in the binary case can be divided by two, since $x \to x^2$ does not require any multiplication. Secondly, for the Boolean case only, a new attack on full MiMC has been presented recently [35]. The latter attack combines a distinguisher based on higher-order differential technique (that can cover up to $\lceil (n-1) \cdot \log_3(2) \rceil - 1$ rounds) with an interpolation technique. That makes it possible to find the secret key. Since the data cost of such attack is half of the full code-book, it does not apply in this context, because we are working with a PRF. Indeed, we only consider attacks whose complexities are below the birthday bound.

GMiMC$_{erf}$.

GMiMC$_{erf}$ [2] is a scheme from GMiMC family over $\mathbb{F}_p^{\,t}$. The round function is defined as

$$(x_1, x_2, \ldots, x_t) \to (x_2 + (x_1 + k^{(i)})^3, x_3 + (x_1 + k^{(i)})^3, \ldots, x_t + (x_1 + k^{(i)})^3, x_1).$$

The detail regarding the key schedule is explained in [2]. We note that the round keys can be precomputed, thus they do not influence the cost in MPC applications. Under the assumption that $p \gg t$, and for a security level of $\log_2(p)$ bits, the number of rounds is given by[10]

$$\max\left\{2 + 2 \cdot (t + t^2), \lceil 2 \cdot \log_3(p) \rceil + 2t\right\}.$$

More precisely, the designers deduced that $2 + (t + t^2)/2$ are sufficient to prevent differential attacks, under the assumption that there is a differential trail for $t + t^2$ rounds with prob. $4 \cdot p^{-2}$. However, in [15], the authors demonstrated that such differential trail has a probability equal to $2 \cdot p^{-1}$. Consequently, they proved the existence of differential distinguishers that can cover the full cipher (and even more). For this reason, we adapted the number of rounds as suggested by the authors of [2] (in a private communication[11]).

With the same application as in MiMC, the MPC cost using GMiMC_{erf} is given by:

$$\begin{aligned} \#\text{ multiplications:} &\quad 2 \cdot \max\left\{2 + 2(t + t^2), \lceil 2 \cdot \log_3(p) \rceil + 2t\right\}, \\ (\#\text{ online}, \#\text{ offline) rounds:} &\quad \left(\max\left\{2 + 2(t + t^2), \lceil 2 \cdot \log_3(p) \rceil + 2t\right\}, 0\right). \end{aligned}$$

Vision.

Vision [5] is an AES-like scheme that works over $\mathbb{F}_{2^n}{}^t$ for any $n \geq 3$, and $t \geq 2$. The round function is composed of two sub-rounds. It is defined as the following: $R(\cdot) = R_2 \circ R_1(\cdot)$, with

$$R_1(\cdot) = k \oplus M \times [B \circ S(\cdot)], \qquad \text{and} \qquad R_2(\cdot) = k' \oplus M \times \left[B^{-1} \circ S(\cdot)\right].$$

The S-box layer $S(\cdot)$ is defined as the concatenation of S-boxes that work at word level, and that are defined as $x \mapsto 1/x$ (where $1/0 := 0$). In addition, $B(\cdot)$ is defined as the concatenation of invertible linearized polynomials that work at a word level, and that are defined as $B(x) = x^4 \oplus b_2 \cdot x^2 \oplus b_3 \cdot x \oplus b_4$. In addition, M is a $t \times t$ MDS matrix. For a security level of n bits, the number of rounds is equal to $\max\left\{10, 2\left\lceil \frac{n+t+8}{8t} \right\rceil, 2\left\lceil \frac{2n}{(t+1)(n-4)} \right\rceil\right\}$.

Concerning the number of multiplications, the only non-linear operation is the inverse. In a MPC application, the inversion step can be evaluated by using the technique of Bar-Ilan and Beaver [10]. In brief, given x, y, z s.t. $x/y = z$ (where $y \neq 0$), the idea is to manipulate the equality $x = y \cdot z$ rather than $x/y = z$. This procedure requires two communication rounds, and works for all non-zero elements $x \in \mathbb{F}_{2^n}$. In scenarios where the shared value is unlikely to be zero (i.e., if the field is large enough), this technique can be used directly. Ignoring the zero test, the total cost of this method is one communication round. In the latter, it is possible to merge a multiplication, and an opening call. As presented in [5, App. E.2 of Version: 20190520:100450], $B(\cdot)$ and $B^{-1}(\cdot)$ require respectively two and three multiplications in

[10] The Gröbner basis attack does not outperform the interpolation attack under the assumption $p \gg t$.

[11] The goal is to guarantee that each differential trail has probability lower than $p^{-2 \cdot t}$ for a security level of $\log_2(p)$ bits.

MPC. Like previously, to give an overview, instead of using $B^{-1}(x) = y$, the idea is to work with $y = B(x)$ (remember that $B(\cdot)$ is semi-linear in the sense that $B(x \oplus y) = B(x) \oplus B(y)$).

It results that the cost in MPC protocol to working over $\mathbb{F}^t$ (for a security level of n bits) is given by

$$\begin{aligned}
\#\text{ multiplications:} &\quad 7t \cdot \max\left\{10, 2\left\lceil \frac{n+t+8}{8t} \right\rceil, 2\left\lceil \frac{2n}{(t+1)(n-4)} \right\rceil\right\},\\
\#\text{ online rounds:} &\quad 5 \cdot \max\left\{10, 2\left\lceil \frac{n+t+8}{8t} \right\rceil, 2\left\lceil \frac{2n}{(t+1)(n-4)} \right\rceil\right\},\\
\#\text{ offline rounds:} &\quad 2 \cdot \max\left\{10, 2\left\lceil \frac{n+t+8}{8t} \right\rceil, 2\left\lceil \frac{2n}{(t+1)(n-4)} \right\rceil\right\}.
\end{aligned}$$

We refer to [5, Version: 20190520:100450] for all details about the number of online/offline rounds. In this section, we solely recall that such numbers are independent of the number of S-boxes computed in parallel.

In comparison, the number of multiplications in a "classical setting" is much higher. In particular, using a square-and-multiply strategy, $x \mapsto x^{-1} = x^{2^n-2}$ requires $n-2$ multiplications and $n-1$ squarings (see [5, App. F of Version: 20190520:100450]). In this case, the total number of multiplications is higher, and is given by $2t \cdot (n-2) \cdot \max\left\{10, 2\left\lceil \frac{n+t+8}{8t} \right\rceil, 2\left\lceil \frac{2n}{(t+1)(n-4)} \right\rceil\right\} \in \mathcal{O}(n \cdot t)$.

Rescue.

Rescue [5] is an AES-like scheme that works over $\mathbb{F}_p^t$ for a prime $p \geq 3$, and $t \geq 2$. Let $\alpha \geq 3$ be the lowest integer s.t. $\gcd(p-1, \alpha) = 1$. The round function that is composed of two sub-rounds $R(\cdot) = R_2 \circ R_1(\cdot)$ is defined as follows:

$$R_1(\cdot) = k + M \times S^{-1}(\cdot)\,, \qquad \text{and} \qquad R_2(\cdot) = k' + M \times S(\cdot)\,.$$

The S-box layer $S(\cdot)$ is defined as the concatenation of S-boxes that work at word level, where as $x \mapsto x^\alpha$ (and S-box$^{-1}(x) = x^{1/\alpha}$). M is a $t \times t$ MDS matrix. For a security level of $\approx \log_2(p)$ bits and $\alpha = 3$, the number of rounds is equal to $\max\left\{10, 2 \cdot \left\lceil \frac{\log_2(p)+2}{4t} \right\rceil; 2 \cdot \left\lceil \frac{2\log_2(p)}{(t+1)\cdot(\log_2(p)-1)} \right\rceil\right\}$.

The only non-linear operations of Rescue to consider are the α, and inverse-α power maps. For any arbitrary large β, $x \mapsto x^\beta$ can be computed by adapting the exponentiation technique that was introduced

by Damgård et al. [30]. This technique requires $\lceil \log_2 \beta \rceil + 2$ multiplications. Hence, the cost in a MPC protocol is given by

$$\begin{aligned}
&\#\text{ multiplications:} \\
&\quad t \cdot (4\lceil \log_2 \alpha \rceil + 4) \cdot \max\left\{5; \left\lceil \frac{\log_2(p)+2}{4t} \right\rceil; \left\lceil \frac{2\log_2(p)}{(t+1)\cdot(\log_2(p)-1)} \right\rceil\right\}, \\
&\#\text{ online rounds:} \quad 4 \cdot \max\left\{5; \left\lceil \frac{\log_2(p)+2}{4t} \right\rceil; \left\lceil \frac{2\log_2(p)}{(t+1)\cdot(\log_2(p)-1)} \right\rceil\right\}, \\
&\#\text{ offline rounds:} \\
&\quad (2\lceil \log_2 \alpha \rceil + 2) \cdot \max\left\{5; \left\lceil \frac{\log_2(p)+2}{4t} \right\rceil; \left\lceil \frac{2\log_2(p)}{(t+1)\cdot(\log_2(p)-1)} \right\rceil\right\},
\end{aligned}$$

where we refer to [5, Version: 20190520:100450] for more details about the number of online/offline rounds.

In comparison, the number of multiplications is higher in a "classical setting". In particular, x^α requires approximately $\lceil \log_2 \alpha \rceil$ multiplications, while $x^{1/\alpha}$ requires approximately $\lceil \log_2 \alpha \rceil + \lceil \log_2 p \rceil$ multiplications (see [5, App. F of Version: 20190520:100450]). For this reason, by applying a square-and-multiply strategy, the number of multiplications is higher, and approximately given by $2t \cdot \left(2\lceil \log_2 \alpha \rceil + \lceil \log_2 p \rceil\right) \cdot \max\left\{5, \left\lceil \frac{\log_2(p)+2}{4t} \right\rceil; \left\lceil \frac{2\log_2(p)}{(t+1)\cdot(\log_2(p)-1)} \right\rceil\right\} \in \mathcal{O}(\log_2(p) \cdot t)$.

G.2 About fork-like ciphers

Our design is based on a modified version of Farfalle that can be viewed as a generalization of the "ForkCipher" design [7]. A ForkCipher design is instantiated by keyed permutations $P_k, \hat{P}_k, \tilde{P}_k$ over $\mathbb{F}$, which are in general defined as the concatenation of a certain number of rounds of a given cipher. In this design, every input $x \in \mathbb{F}$ is mapped to

$$\mathrm{ForkP}(x) = (\hat{P}_k \circ P_k(x), \tilde{P}_k \circ P_k(x)) \in \mathbb{F}^2.$$

However, unlike our design, a Fork-like cipher is invertible. That is to say, given $\mathrm{ForkP}(x) = (\hat{y}, \tilde{y})$, the following operations are possible:

$$\begin{aligned}
\text{Decryption:} \quad & x = P_k^{-1} \circ \hat{P}_k^{-1}(\hat{y}) = P_k^{-1} \circ \tilde{P}_k^{-1}(\tilde{y}), \\
\text{Inversion:} \quad & \tilde{P}_k \circ \hat{P}_k^{-1}(\hat{y}) = \tilde{y} \text{ and } \hat{P}_k \circ \tilde{P}_k^{-1}(\tilde{y}) = \hat{y}.
\end{aligned}$$

None of them is possible in our design, because the input of the middle phase is obtained by adding the outputs of the compression phase, and because of the truncation applied on the output words.

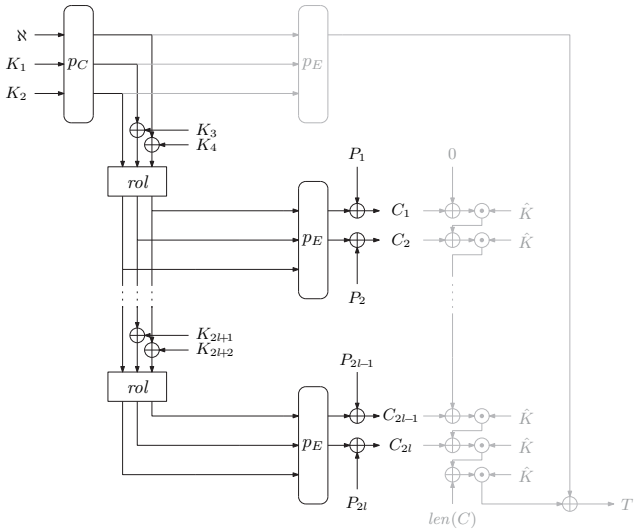


Figure 7.10: Authenticated encryption with Ciminion plus Wegman-Carter MAC over $\mathbb{F}_{2^n}$. The construction is similar over $\mathbb{F}_p$ ($\oplus$ is replaced by +, the addition modulo p).

H Towards an authenticated encryption scheme

A potential direction that can be explored when authenticated encryption is needed, is to pair our encryption scheme with a Wegman-Carter MAC [23]. This makes it possible to process one ciphertext element with one field multiplication as shown in Figure 7.10.

To be more specific, to augment Ciminion with the authentication of ciphertexts, a similar approach to GHASH that is part of GCM [52], can be used. In further detail, our construction uses $T = uhash(\hat{K}, C) + prf(\mathcal{N})$, where $uhash(\hat{K}, C) = C_1\hat{K}^{2l+1} + C_2\hat{K}^{2l} + \cdots C_{2l}\hat{K}^2 + len(C)\hat{K}$ with $\hat{K}$ as secret key element, $len(C)$ the number of field elements in C, and $prf(\mathcal{N})$ an instantiation that is part of our Farfalle-like construction (Figure 7.3).

The security of the resulting Wegman-Carter MAC depends on the security of the used instance of $prf(\mathcal{N})$, and the ϵ-almost-Δ-universality of the universal hash function $uhash(\hat{K}, C)$ that is employed. We have $P(uhash(\hat{K}, C) + uhash(\hat{K}, C') = A) \leq \epsilon$ for any constant A over a uniformly random choice of $\hat{K}$. Following McGrew and Viega [52] for our choice of $uhash(\hat{K}, C)$, we have $\epsilon \leq (2l+1)2^{-n}$ for $\mathbb{F}_{2^n}$, or $\epsilon \leq (2l+1)/p$ for $\mathbb{F}_p$, where $2l$ is the maximal number of elements per call to authenticated encryption or decryption. Like mentioned by Procter [56], an adversary then has an advantage that is at most $\beta\epsilon$ plus the advantage in breaking $prf(\mathcal{N})$ for creating a forgery. In this case, β is the total number of queries that are made to the authenticated encryption or decryption. Hence, for maintaining a sufficient level of security, the maximum length of messages, the maximum number of messages, and the verification attempts have to be limited.

Despite the latter restriction and other unfavorable properties [57] of this style of authentication, we think that the efficiency benefits provided by Wegman-Carter-style MACs in scenarios where finite field multiplication is the dominant cost factor, reasonably counterbalance its downsides.

I Algorithms

Algorithm 6: Encryption and decryption, where the finite field is either $\mathbb{F}_{2^n}$, or $\mathbb{F}_p$.

Encryption

Require: key $K \in \{\mathbb{F}\}^{2\lceil o/2 \rceil}$,
nonce $\mathcal{N} \in \mathbb{F}$,
plaintext $P \in \{\mathbb{F}\}^o$
Ensure: ciphertext $C \in \{\mathbb{F}\}^o$

Processing Nonce
$S_1 \| S_2 \| S_3 \leftarrow \mathcal{N} \| K_1 \| K_2$
$S_1 \| S_2 \| S_3 \leftarrow p_C(S_1 \| S_2 \| S_3)$
Encrypting Plaintext
for $i = 1, \ldots, \lceil o/2 \rceil$ **do**
$O_1 \| O_2 \| O_3 \leftarrow p_E(S_1 \| S_2 \| S_3)$
$C_{2i-1} \leftarrow O_1 + P_{2i-1}$
if $i < \lceil o/2 \rceil$ OR $\lceil o/2 \rceil = o/2$ **then**
$C_{2i} \leftarrow O_2 + P_{2i}$
end if
if $i < \lceil o/2 \rceil$ **then**
$S_2 \leftarrow S_2 + K_{i+1}$
$S_3 \leftarrow S_3 + K_{i+2}$
$S_1 \| S_2 \| S_3 \leftarrow rol(S_1 \| S_2 \| S_3)$
end if
end forreturn $C_1 \| \ldots \| C_o$

Decryption

Require: key $K \in \{\mathbb{F}\}^{2\lceil o/2 \rceil}$,
nonce $\mathcal{N} \in \mathbb{F}$,
ciphertext $C \in \{\mathbb{F}\}^o$

Ensure: plaintext $P \in \{\mathbb{F}\}^o$

Processing Nonce
$S_1 \| S_2 \| S_3 \leftarrow \mathcal{N} \| K_1 \| K_2$
$S_1 \| S_2 \| S_3 \leftarrow p_C(S_1 \| S_2 \| S_3)$
Decrypting Ciphertext
for $i = 1, \ldots, \lceil o/2 \rceil$ **do**
$O_1 \| O_2 \| O_3 \leftarrow p_E(S_1 \| S_2 \| S_3)$
$P_{2i-1} \leftarrow O_1 + C_{2i-1}$
if $i < \lceil o/2 \rceil$ OR $\lceil o/2 \rceil = o/2$ **then**
$P_{2i} \leftarrow O_2 + C_{2i}$
end if
if $i < \lceil o/2 \rceil$ **then**
$S_2 \leftarrow S_2 + K_{i+1}$
$S_3 \leftarrow S_3 + K_{i+2}$
$S_1 \| S_2 \| S_3 \leftarrow rol(S_1 \| S_2 \| S_3)$
end if
end forreturn $P_1 \| \ldots \| P_o$

Algorithm 7: Generation of key elements, where the finite field is either $\mathbb{F}_{2^n}$ or $\mathbb{F}_p$.

Generation of Key Elements

Require: master key $MK \in \{\mathbb{F}\}^2$,
$IV_H \in \mathbb{F}$
Ensure: key elements $K \in \{\mathbb{F}\}^o$

$S_1 \| S_2 \| S_3 \leftarrow IV_H \| MK_1 \| MK_2$
for $i = 1, \ldots, o$ **do**
$S_1 \| S_2 \| S_3 \leftarrow p_C(S_1 \| S_2 \| S_3)$
$K_i \leftarrow S_1$
end forreturn $K_1 \| \ldots \| K_o$

Table 7.8: Number of monomials of a certain degree, where all inputs are considered as variables in $\mathbb{F}_{2^n}$. All round constants $RC4_i$ equal 1.

	Output	Degree																												
Round	Variable	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	
	max	1	3	6	10	15	21	28	36	45	55	66	78	91	105	120	136	153	171	190	210	231	253	276	300	325	351	378	406	
	a	1	1	1																										
1	b	1	3	1																										
	c	1	2	1																										
	a	1	3	4	2	1																								
2	b	1	3	4	2	1																								
	c	1	3	4	2	1																								
	a	1	3	6	6	6	2	2	0	1																				
3	b	1	3	6	6	6	2	2	0	1																				
	c	1	3	6	6	6	2	2	0	1																				
	a	1	3	6	8	12	10	6	4	6	2	2	0	2	0	0	0	1												
4	b	1	3	6	8	12	10	6	4	6	2	2	0	2	0	0	0	1												
	c	1	3	6	8	12	10	6	4	6	2	2	0	2	0	0	0	1												
	a	1	3	6	8	12	14	10	12	12	12	10	4	6	4	4	0	6	2	2	0	2	0	0	0	2	0	0	0	
5	b	1	3	6	8	12	14	10	12	12	12	10	4	6	4	4	0	6	2	2	0	2	0	0	0	2	0	0	0	
	c	1	3	6	8	12	14	10	12	12	12	10	4	6	4	4	0	6	2	2	0	2	0	0	0	2	0	0	0	
	a	1	3	6	8	12	14	14	16	12	24	14	20	10	12	12	8	12	12	12	4	10	4	4	0	6	4	4	0	
6	b	1	3	6	8	12	14	14	16	12	24	14	20	10	12	12	8	12	12	12	4	10	4	4	0	6	4	4	0	
	c	1	3	6	8	12	14	14	16	12	24	14	20	10	12	12	8	12	12	12	4	10	4	4	0	6	4	4	0	

Algorithm 8: Permutation p_N and rolling function *rol*, where the finite field is either $\mathbb{F}_{2^n}$ or $\mathbb{F}_p$.

Rolling Function *rol*

Require: $\iota_a \in \mathbb{F}$,
$\iota_b \in \mathbb{F}$,
$\iota_c \in \mathbb{F}$
Ensure: $\omega_a \in \mathbb{F}$,
$\omega_b \in \mathbb{F}$,
$\omega_c \in \mathbb{F}$

$\iota_c \leftarrow \iota_c + \iota_a \cdot \iota_b$
$\omega_a \leftarrow \iota_c$
$\omega_c \leftarrow \iota_b$
$\omega_b \leftarrow \iota_a$

return $\omega_a, \omega_b, \omega_c$

Permutation p_N

Require: $a \in \mathbb{F}$,
$b \in \mathbb{F}$,
$c \in \mathbb{F}$
Ensure: $a \in \mathbb{F}$,
$b \in \mathbb{F}$,
$c \in \mathbb{F}$

for $i = 1, \ldots, N$ **do**
$c \leftarrow c + a \cdot b$
$b \leftarrow b + c$
$a \leftarrow a + RC4_{i+E-N} \cdot b$
$d \leftarrow a + RC1_{i+E-N}$
$a \leftarrow c + RC3_{i+E-N}$
$c \leftarrow b + RC2_{i+E-N}$
$b \leftarrow d$
end forreturn a, b, c

References

1. M. R. Albrecht, C. Cid, L. Grassi, D. Khovratovich, R. Lüftenegger, C. Rechberger, and M. Schofnegger. "Algebraic Cryptanalysis of STARK-Friendly Designs: Application to MARVELlous and MiMC". In: *ASIACRYPT*. Vol. 11923. LNCS. Springer, 2019, pp. 371–397.
2. M. R. Albrecht, L. Grassi, L. Perrin, S. Ramacher, C. Rechberger, D. Rotaru, A. Roy, and M. Schofnegger. "Feistel Structures for MPC, and More". In: *ESORICS*. Vol. 11736. LNCS. Springer, 2019, pp. 151–171.
3. M. R. Albrecht, L. Grassi, C. Rechberger, A. Roy, and T. Tiessen. "MiMC: Efficient Encryption and Cryptographic Hashing with Minimal Multiplicative Complexity". In: *ASIACRYPT*. Vol. 10031. LNCS. 2016, pp. 191–219.
4. M. R. Albrecht, C. Rechberger, T. Schneider, T. Tiessen, and M. Zohner. "Ciphers for MPC and FHE". In: *EUROCRYPT*. Vol. 9056. LNCS. 2015, pp. 430–454.
5. A. Aly, T. Ashur, E. Ben-Sasson, S. Dhooghe, and A. Szepieniec. *Design of Symmetric-Key Primitives for Advanced Cryptographic Protocols*. Cryptology ePrint Archive, Report 2019/426. 2019.
6. A. Aly, T. Ashur, E. Ben-Sasson, S. Dhooghe, and A. Szepieniec. "Design of Symmetric-Key Primitives for Advanced Cryptographic Protocols". *IACR Trans. Symmetric Cryptol.* 2020:3, 2020, pp. 1–45.
7. E. Andreeva, V. Lallemand, A. Purnal, R. Reyhanitabar, A. Roy, and D. Vizár. "Forkcipher: A New Primitive for Authenticated Encryption of Very Short Messages". In: *ASIACRYPT*. Vol. 11922. LNCS. Springer, 2019, pp. 153–182.
8. T. Ashur and S. Dhooghe. *MARVELlous: a STARK-Friendly Family of Cryptographic Primitives*. Cryptology ePrint Archive, Report 2018/1098. 2018.
9. T. Baignères, J. Stern, and S. Vaudenay. "Linear Cryptanalysis of Non Binary Ciphers". In: *SAC*. 2007, pp. 184–211.
10. J. Bar-Ilan and D. Beaver. "Non-Cryptographic Fault-Tolerant Computing in Constant Number of Rounds of Interaction". In: *ACM Symposium*. ACM, 1989, pp. 201–209.
11. M. Bardet, J. Faugère, and B. Salvy. "On the complexity of the F5 Gröbner basis algorithm". *J. Symb. Comput.* 70, 2015, pp. 49–70.
12. G. Bertoni, J. Daemen, S. Hoffert, M. Peeters, G. Van Assche, and R. Van Keer. "Farfalle: parallel permutation-based cryptography". *IACR Trans. Symmetric Cryptol.* 2017:4, 2017, pp. 1–38.
13. G. Bertoni, J. Daemen, M. Peeters, and G. Van Assche. *The Keccak SHA-3 submission (Version 3.0)*. 2011.
14. G. Bertoni, J. Daemen, M. Peeters, and G. Van Assche. *Sponge functions*. Ecrypt Hash Workshop 2007. 2007.

15. T. Beyne, A. Canteaut, I. Dinur, M. Eichlseder, G. Leander, G. Leurent, M. Naya-Plasencia, L. Perrin, Y. Sasaki, Y. Todo, and F. Wiemer. "Out of Oddity – New Cryptanalytic Techniques against Symmetric Primitives Optimized for Integrity Proof Systems". In: *CRYPTO*. Vol. 12172. LNCS. Springer, 2020, pp. 299–328.

16. E. Biham, A. Biryukov, and A. Shamir. "Cryptanalysis of Skipjack Reduced to 31 Rounds Using Impossible Differentials". In: *EUROCRYPT*. Vol. 1592. LNCS. Springer, 1999, pp. 12–23.

17. E. Biham and A. Shamir. "Differential Cryptanalysis of DES-like Cryptosystems". In: *Advances in Cryptology - CRYPTO '90, 10th Annual International Cryptology Conference, Santa Barbara, California, USA, August 11-15, 1990, Proceedings*. Ed. by A. Menezes and S. A. Vanstone. Vol. 537. Lecture Notes in Computer Science. Springer, 1990, pp. 2–21. DOI: 10.1007/3-540-38424-3_1. URL: https://doi.org/10.1007/3-540-38424-3%5C_1.

18. A. Bogdanov and M. Wang. "Zero Correlation Linear Cryptanalysis with Reduced Data Complexity". In: *FSE*. Vol. 7549. LNCS. Springer, 2012, pp. 29–48.

19. X. Bonnetain. *Collisions on Feistel-MiMC and univariate GMiMC*. Cryptology ePrint Archive, Report 2019/951. 2019.

20. C. Boura, A. Canteaut, and C. De Cannière. "Higher-Order Differential Properties of Keccak and *Luffa*". In: *FSE*. Vol. 6733. LNCS. Springer, 2011, pp. 252–269.

21. B. Buchberger. "A theoretical basis for the reduction of polynomials to canonical forms". *SIGSAM Bull.* 10:3, 1976, pp. 19–29.

22. A. Canteaut, S. Carpov, C. Fontaine, T. Lepoint, M. Naya-Plasencia, P. Paillier, and R. Sirdey. "Stream Ciphers: A Practical Solution for Efficient Homomorphic-Ciphertext Compression". *J. Cryptology* 31:3, 2018, pp. 885–916.

23. L. Carter and M. N. Wegman. "Universal Classes of Hash Functions (Extended Abstract)". In: *STOC*. ACM, 1977, pp. 106–112.

24. J. Daemen. *Cipher and hash function design, strategies based on linear and differential cryptanalysis, PhD Thesis*. K.U.Leuven, 1995.

25. J. Daemen, R. Govaerts, and J. Vandewalle. "Correlation Matrices". In: *Fast Software Encryption: Second International Workshop. Leuven, Belgium, 14-16 December 1994, Proceedings*. Ed. by B. Preneel. Vol. 1008. Lecture Notes in Computer Science. Springer, 1994, pp. 275–285. DOI: 10.1007/3-540-60590-8_21. URL: https://doi.org/10.1007/3-540-60590-8%5C_21.

26. J. Daemen, S. Hoffert, G. Van Assche, and R. Van Keer. "The design of Xoodoo and Xoofff". *IACR Trans. Symmetric Cryptol.* 2018:4, 2018, pp. 1–38. DOI: 10.13154/tosc.v2018.i4.1-38.

27. J. Daemen and V. Rijmen. "The Block Cipher Rijndael". In: *CARDIS*. Vol. 1820. LNCS. 1998, pp. 277–284.

28. J. Daemen and V. Rijmen. *The Design of Rijndael: AES - The Advanced Encryption Standard*. Information Security and Cryptography. Springer, 2002.

29. J. Daemen and V. Rijmen. “The Design of Rijndael: The Advanced Encryption Standard (AES)”. In: Springer, 2020. Chap. Correlation Analysis in GF(2n), pp. 181–194.

30. I. Damgård, N. Fazio, and A. Nicolosi. “Non-interactive Zero-Knowledge from Homomorphic Encryption”. In: *TCC*. Vol. 3876. LNCS. 2006, pp. 41–59.

31. I. Dinur, D. Kales, A. Promitzer, S. Ramacher, and C. Rechberger. “Linear Equivalence of Block Ciphers with Partial Non-Linear Layers: Application to LowMC”. In: *EUROCRYPT*. Vol. 11476. LNCS. Springer, 2019, pp. 343–372.

32. I. Dinur, Y. Liu, W. Meier, and Q. Wang. “Optimized Interpolation Attacks on LowMC”. In: *ASIACRYPT*. Vol. 9453. LNCS. Springer, 2015, pp. 535–560.

33. C. Dobraunig, M. Eichlseder, L. Grassi, V. Lallemand, G. Leander, E. List, F. Mendel, and C. Rechberger. “Rasta: A Cipher with Low ANDdepth and Few ANDs per Bit”. In: *CRYPTO*. Vol. 10991. LNCS. 2018, pp. 662–692.

34. S. Duval, V. Lallemand, and Y. Rotella. “Cryptanalysis of the FLIP Family of Stream Ciphers”. In: *CRYPTO*. Vol. 9814. LNCS. Springer, 2016, pp. 457–475.

35. M. Eichlseder, L. Grassi, R. Lüftenegger, M. Øygarden, C. Rechberger, M. Schofnegger, and Q. Wang. “An Algebraic Attack on Ciphers with Low-Degree Round Functions: Application to Full MiMC”. In: *ASIACRYPT*. Vol. 12491. LNCS. Springer, 2020, pp. 477–506.

36. J.-C. Faugère. “A new efficient algorithm for computing Gröbner bases without reduction to zero F5”. In: *ISSAC*. ACM, 2002, pp. 75–83.

37. J. Faugère, P. M. Gianni, D. Lazard, and T. Mora. “Efficient Computation of Zero-Dimensional Gröbner Bases by Change of Ordering”. *J. Symb. Comput.* 16:4, 1993, pp. 329–344.

38. G. Genovese. “Improving the algorithms of Berlekamp and Niederreiter for factoring polynomials over finite fields”. *J. Symb. Comput.* 42:1-2, 2007, pp. 159–177.

39. L. Grassi, D. Khovratovich, C. Rechberger, A. Roy, and M. Schofnegger. “Poseidon: A New Hash Function for Zero-Knowledge Proof Systems”. In: *USENIX Security 21*. USENIX Association, 2021.

40. L. Grassi, R. Lüftenegger, C. Rechberger, D. Rotaru, and M. Schofnegger. “On a Generalization of Substitution-Permutation Networks: The HADES Design Strategy”. In: *EUROCRYPT*. Vol. 12106. LNCS. 2020, pp. 674–704.

41. L. Grassi, C. Rechberger, and S. Rønjom. “A New Structural-Differential Property of 5-Round AES”. In: *EUROCRYPT*. Vol. 10211. LNCS. 2017, pp. 289–317.

42. L. Grassi, C. Rechberger, D. Rotaru, P. Scholl, and N. P. Smart. “MPC-Friendly Symmetric Key Primitives”. In: *CCS*. ACM, 2016, pp. 430–443.

43. L. Grassi, C. Rechberger, and M. Schofnegger. *Weak Linear Layers in Word-Oriented Partial SPN and HADES-Like Ciphers*. Cryptology ePrint Archive, Report 2020/500. 2020.

44. V. Grosso, F. Standaert, and S. Faust. "Masking vs. multiparty computation: how large is the gap for AES?" *J. Cryptographic Engineering* 4:1, 2014, pp. 47–57.

45. Y. Ishai, A. Sahai, and D. A. Wagner. "Private Circuits: Securing Hardware against Probing Attacks". In: *CRYPTO*. Vol. 2729. LNCS. Springer, 2003, pp. 463–481.

46. T. Jakobsen and L. R. Knudsen. "The Interpolation Attack on Block Ciphers". In: *FSE*. Vol. 1267. LNCS. Springer, 1997, pp. 28–40.

47. L. R. Knudsen. *DEAL – A 128-bit Block Cipher*. 1998.

48. L. R. Knudsen. "Truncated and Higher Order Differentials". In: *FSE 1994*. Ed. by B. Preneel. Vol. 1008. Lecture Notes in Computer Science. Springer, 1994, pp. 196–211. DOI: 10.1007/3-540-60590-8_16. URL: https://doi.org/10.1007/3-540-60590-8%5C_16.

49. X. Lai. "Higher Order Derivatives and Differential Cryptanalysis". In: *Communications and Cryptography: Two Sides of One Tapestry*. Springer US, 1994, pp. 227–233.

50. S. K. Langford and M. E. Hellman. "Differential-Linear Cryptanalysis". In: *CRYPTO*. Vol. 839. LNCS. Springer, 1994, pp. 17–25.

51. M. Matsui. "Linear Cryptanalysis Method for DES Cipher". In: *Advances in Cryptology - EUROCRYPT '93, Proceedings*. Ed. by T. Helleseth. DOI: 10.1007/3-540-48285-7_33.

52. D. A. McGrew and J. Viega. *The Security and Performance of the Galois/Counter Mode of Operation (Full Version)*. Cryptology ePrint Archive, Report 2004/193. 2004.

53. P. Méaux, A. Journault, F. Standaert, and C. Carlet. "Towards Stream Ciphers for Efficient FHE with Low-Noise Ciphertexts". In: *EUROCRYPT*. Vol. 9665. LNCS. Springer, 2016, pp. 311–343.

54. NIST. *FIPS PUB 202: SHA-3 Standard: Permutation-Based Hash and Extendable-Output Functions*. 2015.

55. K. Nyberg and L. R. Knudsen. "Provable Security Against a Differential Attack". *J. Cryptology* 8:1, 1995, pp. 27–37.

56. G. Procter. *A Security Analysis of the Composition of ChaCha20 and Poly1305*. Cryptology ePrint Archive, Report 2014/613. 2014.

57. G. Procter and C. Cid. "On Weak Keys and Forgery Attacks Against Polynomial-Based MAC Schemes". *J. Cryptol.* 28:4, 2015, pp. 769–795.

58. C. Rechberger, H. Soleimany, and T. Tiessen. "Cryptanalysis of Low-Data Instances of Full LowMCv2". *IACR Trans. Symmetric Cryptol.* 2018:3, 2018, pp. 163–181.

59. V. Rijmen, J. Daemen, B. Preneel, A. Bosselaers, and E. De Win. "The Cipher SHARK". In: *FSE*. Vol. 1039. LNCS. 1996, pp. 99–111.

60. T. Simon, L. Batina, J. Daemen, V. Grosso, P. M. C. Massolino, K. Papagiannopoulos, F. Regazzoni, and N. Samwel. "Friet: An Authenticated Encryption Scheme with Built-in Fault Detection". In: *EUROCRYPT*. Vol. 12105. LNCS. 2020, pp. 581–611.

61. T. Toffoli. “Reversible Computing”. In: *ICALP*. Vol. 85. LNCS. Springer, 1980, pp. 632–644.

62. D. A. Wagner. “The Boomerang Attack”. In: *FSE*. Vol. 1636. LNCS. Springer, 1999, pp. 156–170.

8 Propagation Properties of a Non-linear Mapping Based on Squaring in Odd Characteristic

Joan Daemen[1], Daniël Kuijsters[1], Silvia Mella[1], Denise Verbakel[1]

1 – Radboud University, The Netherlands

My contributions. This chapter is based on work accepted at BFA 2023. I was responsible for writing most of the text and made a contribution by determining the exact values of the correlations of the squaring mapping.

Abstract. Many modern cryptographic primitives for hashing and (authenticated) encryption make use of constructions that are instantiated with an iterated cryptographic permutation that operates on a fixed-width state consisting of an array of bits. Often, such permutations are the repeated application of a relatively simple round function consisting of a linear layer and a non-linear layer. These constructions do not require that the underlying function is a permutation and they can plausibly be based on a non-invertible transformation. Recently, Grassi proposed the use of non-invertible mappings operating on arrays of digits that are elements of a finite field of odd characteristic for so-called MPC-/FHE-/ZK-friendly symmetric cryptographic primitives. In this work, we consider a mapping that we call γ that has a simple expression and is based on squaring. We discuss, for the first time, the differential and linear propagation properties of γ and observe that these follow the same rules up to a relabeling of the digits. This is an intriguing property that, as far as we know, only exists for γ and the binary mapping χ_3 that is used in the cryptographic permutation Xoodoo. Moreover, we study the implications of its non-invertibility on differentials with zero output difference and on biases at the output of the γ mapping and show that they are as small as they can possibly be.

8.1 Introduction

The round functions in cryptographic permutations of the type Substitution-Permutation Networks (SPN) consist of a non-linear layer and a linear layer. These layers are chosen and combined so that there is no exploitable differential propagation from input to output or exploitable correlations between input and output when used in the context of a construction like the sponge or duplex construction [6],

Farfalle [5] or Even-Mansour [16]. The relevant properties of these mappings over binary fields have been studied extensively, leading to solid designs. However, in the last years there has been a growing interest in similar functions operating on arrays of digits that are elements of a field of odd characteristic. For instance, Kölbl et al. designed a ternary cryptographic hash function called Troika [20]. Other examples are the symmetric primitives defined over $\mathbb{F}_p^n$ like MiMC [2], GMiMC [1], Poseidon [17], Ciminion [15], and many others. These are designed to be efficient in the context of Secure Multi-Party Computation (MPC), Fully Homomorphic Encryption (FHE), and Zero-Knowledge proofs (ZK).

There are interesting differences between fields $\mathbb{F}_{2^d}$ of characteristic 2 and those of odd characteristic that we will denote by $\mathbb{F}_q$. For instance, addition and subtraction are the same in $\mathbb{F}_{2^d}$, but this is not the case in $\mathbb{F}_q$. In $\mathbb{F}_{2^d}$, squaring is a linear operation, whereas in $\mathbb{F}_q$ squaring is a non-linear operation. In $\mathbb{F}_2$, correlations between input and output bits have values that are rational and range from -1 to 1, but in $\mathbb{F}_p$, correlations are complex numbers inside the closed unit disk.

This work investigates a mapping over $\mathbb{F}_q^n$ that was recently proposed by Grassi [18] and that we call γ. This is the mapping defined over $\mathbb{F}_q^n$ by $\gamma_i(x) = x_i + x_{i+1}^2$ for $i \in \mathbb{Z}/n\mathbb{Z}$ and for all $x \in \mathbb{F}_q^n$.

The paper is organized as follows. Section 8.2 deals with commonly used notation and conventions that we follow. In Section 8.3 we recall the basic notions from differential cryptanalysis. An overview of correlation analysis is presented in Section 8.4. In Section 8.5 we apply this existing theory to the squaring transformation and derive its DP and LP values. Based on the squaring transformation, we motivate the choice for γ in Section 8.6. The main contribution of this paper lies in Section 8.7 and Section 8.8, where we study the differential and linear propagation properties of γ, both in the forward and backward direction. Our results are useful in determining the maximum probabilities of differentials and differential trails over transformations making use of γ in their round function, as in computer-assisted trail search [12]. Moreover, as the differential and linear propagation properties of γ follow the same rules, our results are also useful to study the correlations of linear approximations and linear trails. In Section 8.9 we study the collision probability and bias of linear combinations of output digits of γ. Finally, we conclude in Section 8.10.

8.2 Notation and conventions

We denote by $\mathbb{F}_q$ the finite field of *odd* characteristic p, i.e., q is equal to p^d for some odd prime p and positive integer $d > 0$. Let $\mathbb{F}_q^n$ be the vector space of dimension n over the finite field $\mathbb{F}_q$. Given two vectors $x, y \in \mathbb{F}_q^n$, we denote their vector subtraction by $x - y$, i.e., $x - y = x + (-1)y$. A vector $x \in \mathbb{F}_q^n$ is indexed by the set $\mathbb{Z}/n\mathbb{Z}$. We denote its ith coordinate by x_i and call it a *digit*. The dot product between x and y is defined as $x^\top y = \sum_{i=0}^{n-1} x_i y_i$. We write e_i for the vector with all digits equal to 0, except for the digit that is indexed by i, which is equal to 1. The linear span of a set of vectors $S \subseteq \mathbb{F}_q^n$ is denoted by $\mathrm{Span}(S)$. A digit is said to be *active* if it is non-zero. The Hamming weight $\mathrm{HW}(x)$ of a vector $x \in \mathbb{F}_q^n$ is the number of active digits in the vector.

Let $z \in \mathrm{C}$ be a complex number. We denote its absolute value as $|z|$. We write $\overline{z}$ for its complex conjugate.

Let F be a field, then we write F^* for its multiplicative group $F \setminus \{0\}$.

8.3 Differential analysis

First published by Biham and Shamir in [9], *differential cryptanalysis* is a chosen-plaintext attack that exploits the non-uniformity of the distribution of differences at the output of a transformation when it is applied to pairs of inputs with a fixed difference.

Any successful theory of cryptanalysis needs to address the problem of secret key translation. Differential cryptanalysis deals with this problem by considering differences, which are invariant under translation. Let $x \in \mathbb{F}_q^n$ and $x^* \in \mathbb{F}_q^n$ be inputs of a transformation $\alpha\colon \mathbb{F}_q^n \to \mathbb{F}_q^n$ and let their difference be $a = x^* - x$. Likewise, let $y \in \mathbb{F}_q^n$ and $y^* \in \mathbb{F}_q^n$ be outputs of α and let their difference be $b = y^* - y$. The (ordered) pair $(a, b) \in \mathbb{F}_q^n \times \mathbb{F}_q^n$ containing the input and output difference is called a *differential* over α. The differential $(0, 0)$ is called *trivial*. The *differential probability (DP)* of a differential (a, b) over the transformation α is defined as

$$\mathrm{DP}_\alpha(a, b) = q^{-n} \left| \{x \in \mathbb{F}_q^n : \alpha(x + a) - \alpha(x) = b\} \right| .$$

If $\mathrm{DP}_\alpha(a, b) > 0$, we say that a and b are *compatible* differences over α. For compatible differences a and b, we define the weight of a differential (a, b) over α as

$$\mathrm{w}_\alpha(a, b) = -\log_q(\mathrm{DP}_\alpha(a, b)) .$$

A non-trivial differential (a, b) over α can only lead to a distinguisher if $\mathrm{DP}_\alpha(a, b)$ differs significantly from q^{-n}, which is the expected DP of any non-trivial differential over a randomly selected transformation of $\mathbb{F}_q^n$.

8.4 Correlation analysis

Correlation analysis of cryptographic primitives effectively is Fourier analysis on finite abelian groups. As such, the theory is well-understood and this section serves as a recap. The ideas that we present here are based on the works of Daemen [13], Baignères et al. [3], and Daemen and Rijmen [14]. Many of the proofs can be found in the book by Hou [19].

8.4.1 Characters

Let $(G, +)$ be a finite abelian group and let e be the (finite) exponent of G, i.e., the smallest integer n such that $na = 0$ for all $a \in G$.

A *character* of G is a homomorphism from G into the subgroup of $\mathbb{C}^*$ consisting of the eth roots of unity. The set of characters of G is denoted by $\hat{G}$ and it forms a group under the multiplication defined

by $(\chi\chi')(a) = \chi(a)\chi'(a)$ for all $a \in G$ and $\chi, \chi' \in \hat{G}$. The groups G and $\hat{G}$ are isomorphic, but this isomorphism is not canonical.

For a fixed isomorphism between G and $\hat{G}$ and for each $a \in G$, we write χ_a for the image of a under this isomorphism. In particular, the character χ_0 that is defined by $\chi_0(a) = 1$ for all $a \in G$ is called the *trivial character* and it is the identity element of the group $\hat{G}$.

Now, let $(G, +, \cdot)$ be the commutative ring that is obtained by introducing a multiplicative structure on G. This is always possible by the fundamental theorem of finite abelian groups. A character $\chi \in \hat{G}$ is called a *generating character* for G if $\chi_a(b) = \chi(ab)$ for all $a, b \in G$. If a commutative ring has a generating character for its additive group, then $\chi_a(b) = \chi(ab) = \chi(ba) = \chi_b(a)$. In the case that G is the direct sum of n copies of a commutative ring R and if R has a generating character, say ϕ, then we obtain a generating character χ for G by setting $\chi(a_1, \ldots, a_n) = \phi(a_1) \cdots \phi(a_n)$. It holds that $\chi_a(b) = \chi(ab) = \phi(a^\top b)$, where the multiplication in G is defined component-wise.

As an example, consider G equal to $\mathbb{F}_q$ and put $\omega = e^{2\pi i/p}$. Let $\mathrm{Tr}\colon \mathbb{F}_q \to \mathbb{F}_p$ be the absolute trace function that is defined by $\mathrm{Tr}(x) = \sum_{i=0}^{d-1} x^{p^i}$. This is a linear mapping. Each $u \in \mathbb{F}_q$ defines a generating character χ_u for $\mathbb{F}_q$ that is defined by

$$\chi_u(x) = \omega^{\mathrm{Tr}(ux)}, \qquad x \in \mathbb{F}_q .$$

As a second example, consider G equal to $\mathbb{F}_q^n$, which is a direct sum of n copies of $\mathbb{F}_q$. Hence, each $u \in \mathbb{F}_q^n$ gives a generating character χ_u for $\mathbb{F}_q^n$ that is defined by

$$\chi_u(x) = \omega^{\mathrm{Tr}(u^\top x)}, \qquad x \in \mathbb{F}_q^n .$$

8.4.2 The Fourier transform

Consider the set $L^2(G)$ of functions $f\colon G \to \mathrm{C}$. Fix an ordering of the element of G, e.g., $G = \{a_0, \ldots, a_{n-1}\}$. We write $v_f = (f(a_0), \ldots, f(a_{n-1}))$ for the finite sequence of the output values of f. By identifying a function f with the vector $v_f \in \mathrm{C}^{|G|}$, $L^2(G)$ can be seen as a finite-dimensional complex inner product space with inner product

$$\langle f, g \rangle = \sum_{a \in G} f(a)\overline{g(a)}, \qquad f, g \in L^2(G) .$$

For any $f \in L^2(G)$, the inner product induces a norm by setting

$$\|f\| = \langle f, f \rangle^{\frac{1}{2}} .$$

The standard basis of $L^2(G)$ is formed by the set of Dirac delta functions $\{\delta_a \in L^2(G) : a \in G\}$, which are defined by

$$\delta_a(b) = \begin{cases} 1 & \text{if } a = b\,, \\ 0 & \text{if } a \neq b\,. \end{cases}$$

In the context of correlation analysis, the solution to the problem of secret key translation lies in changing the basis of $L^2(G)$ to the set of characters of G. For any $a, b \in G$, the corresponding characters satisfy $\langle \chi_a, \chi_b \rangle = |G| \delta_a(b)$. By normalizing the characters, we obtain an orthonormal basis

$$\Phi_G = \{\phi_a : a \in G\}\,,$$

where $\phi_a = |G|^{-\frac{1}{2}} \chi_a$. By projecting f onto Φ_G, we find that

$$f = \sum_{a \in G} \langle f, \phi_a \rangle \phi_a\,.$$

The operator $\mathcal{F} \colon L^2(G) \to L^2(G)$ that is defined by $\mathcal{F}(f)(a) = \langle f, \phi_a \rangle$ for all $a \in G$ is called the *Fourier transform*. By identifying a function f with v_f, the Fourier transform is best described as assigning to f its coordinates in the normalized character basis. The *Plancherel theorem* asserts that the Fourier transform is unitary, i.e., we have

$$\langle \mathcal{F}(f), \mathcal{F}(g) \rangle = \langle f, g \rangle, \qquad f, g \in L^2(G)\,.$$

Let us return to the question of how to address the problem of secret key translation. Let $b \in G$. We define the translation operator $T_b \colon L^2(G) \to L^2(G)$ by $(T_b f)(a) = f(a + b)$ for all $a \in G$. Moreover, we define the modulation operator $M_b \colon L^2(G) \to L^2(G)$ by $(M_b f)(a) = \phi_b(a) f(a)$ for all $a \in G$. The big insight is that translation turns into modulation when changing from the standard basis to the normalized character basis, i.e.,

$$T_b = \mathcal{F}^{-1} \circ M_b \circ \mathcal{F}, \qquad b \in G\,.$$

Let H be a finite abelian group and let $F \colon G \to H$ be a mapping between G and H. We want a representation of F in $L^2(G)$. To that end, let χ be any character of H. We take as representation the function $\chi \circ F \in L^2(G)$.

8.4.3 Correlation

We now specialize to the case that G and H are each equal to the vector space $\mathbb{F}_q^n$ over the finite field $\mathbb{F}_q$.

Let $\alpha\colon \mathbb{F}_q^n \to \mathbb{F}_q^n$ be a transformation of $\mathbb{F}_q^n$. We consider pairs $(u, v) \in \mathbb{F}_q^n \times \mathbb{F}_q^n$ that we call *linear approximations* of α. We refer to u as the *output mask* and to v as the *input mask*. The linear approximation $(0, 0)$ is called *trivial*. The *correlation* of the linear approximation is defined as

$$\mathrm{C}_\alpha(u, v) = q^{-\frac{n}{2}} \mathcal{F}(\chi_u \circ \alpha)(v) .$$

We call the masks u and v *compatible* over α if $\mathrm{C}_\alpha(u, v)$ is nonzero. In general, correlations are complex numbers. The *linear potential (LP)* is a real number and related to a correlation by

$$\mathrm{LP}_\alpha(u, v) = \mathrm{C}_\alpha(u, v)\overline{\mathrm{C}_\alpha(u, v)} .$$

If u and v are compatible over α, then we can define the *weight* of the linear approximation (u, v) as

$$\mathrm{w}_\alpha(u, v) = -\log_q(\mathrm{LP}_\alpha(u, v)) .$$

8.5 The squaring transformation

The squaring transformation $\beta\colon \mathbb{F}_q \to \mathbb{F}_q$ is defined by $x \mapsto x^2$ for all $x \in \mathbb{F}_q$. Because we study the case of odd characteristic, β is non-linear. We show that β has the property that the maximal DP over all non-trivial differentials is q^{-1}, which is the smallest possible value. A similar property holds for the maximal LP over all non-trivial linear approximations. In other words, we show that β is a *bent* polynomial [11]. Note that this is an improvement from the case of characteristic 2, for which these values are both equal to $2q^{-1}$ and are obtained by, respectively, almost perfect nonlinear and bent functions [10].

First, by applying Theorem 5.33 from [21], we obtain that the correlation of any linear approximation $(u, v) \in \mathbb{F}_q \times \mathbb{F}_q$ with $u \neq 0$ of β is equal to

$$\begin{aligned}
\mathrm{C}_\beta(u, v) &= q^{-\frac{1}{2}} \mathcal{F}(\chi_u \circ \beta)(v) \\
&= q^{-1} \sum_{x \in \mathbb{F}_q} \chi_1(ux^2 - vx) \\
&= \begin{cases} q^{-\frac{1}{2}} (-1)^{d-1} \chi_1(-v^2(4u)^{-1})\eta(u) & \text{if } p \equiv 1 \pmod 4 , \\ q^{-\frac{1}{2}} (-1)^{d-1} i^d \chi_1(-v^2(4u)^{-1})\eta(u) & \text{if } p \equiv 3 \pmod 4 , \end{cases}
\end{aligned}$$

where $\eta(u) = 1$ if u is a square in $\mathbb{F}_q$ and -1 otherwise. It follows that for all $u, v \in \mathbb{F}_q$ with $u \neq 0$ we have $\mathrm{LP}_\beta(u, v) = q^{-1}$. In particular, choosing v equal to zero shows that any linear combination of output digits of β is imbalanced, i.e., the distribution of this linear combination is non-uniform. If u is 0, then for all nonzero $v \in \mathbb{F}_q$ we have $\mathrm{LP}_\beta(0, v) = 0$, and $\mathrm{LP}_\beta(0, 0) = 1$.

Second, consider the equation that relates the input $x \in \mathbb{F}_q$, the input difference $a \in \mathbb{F}_q$, and the output difference $b \in \mathbb{F}_q$, i.e.,

$$\begin{aligned} b &= \beta(x + a) - \beta(x) \\ &= (x + a)^2 - x^2 \\ &= x^2 + 2ax + a^2 - x^2 \\ &= 2ax + a^2 \,. \end{aligned}$$

Assuming that $a \neq 0$ and because the characteristic of $\mathbb{F}_q$ is odd, we can solve for x to find that $x = (2a)^{-1}(b - a^2)$. Hence, there is exactly one solution to this equation. Dividing by the domain size, q, then shows that $\mathrm{DP}_\beta(a, b) = q^{-1}$. In particular, any nonzero input difference can propagate to a zero output difference. If a is 0, then for all nonzero $b \in \mathbb{F}_q$, we have $\mathrm{DP}_\beta(0, b) = 0$ and $\mathrm{DP}_\beta(0, 0) = 1$.

We summarize these properties to make the symmetry between the differential and linear case apparent:

- For all $a, u \in (\mathbb{F}_q)^*$ and $b, v \in \mathbb{F}_q$, we have $\mathrm{DP}_\beta(a, b) = \mathrm{LP}_\beta(u, v) = q^{-1}$;
- For all $b, v \in (\mathbb{F}_q)^*$, we have $\mathrm{DP}_\beta(0, b) = \mathrm{LP}_\beta(0, v) = 0$;
- We have $\mathrm{DP}_\beta(0, 0) = \mathrm{LP}_\beta(0, 0) = 1$.

8.6 The γ mapping

Some modern block cipher modes, like GCM [23], CTR and OFB [22], do not use the inverse block cipher. Similarly, constructions like sponge [7], duplex [6], and Farfalle [5], which are generally based on permutations, do not use their inverse. Therefore, in such constructions permutations can be replaced by transformations. An example is the GLUON family of lightweight hash functions [4], which makes use of the sponge construction on top of a non-invertible map.

A cryptographic transformation can be used as long as collisions and imbalances in the output cannot be exploited. This can be tackled by either ensuring that such imbalance is very small or by limiting the attacker's access to the input and output of the transformation by construction. For instance, in the sponge and duplex constructions the attacker has control of only the outer part of the state and not of its inner part. Therefore, if a collision requires a difference in the inner part of the state at the input of the transformation, the attacker cannot inject it with input messages. Similarly, the attacker has no visibility of the inner bits or digits of any output mask. As another example, whitening keys can be added at input and output, like in Farfalle [5], Even-Mansour [16], and Elephant [8].

We consider the problem of building a non-invertible mapping based on squaring that can be used as non-linear layer in the round function of cryptographic transformations. When such transformations are used in constructions that are usually instantiated with permutations, the non-invertibility of the mapping should be difficult to exploit.

By definition, such a non-linear layer has pairs of distinct inputs that are mapped to the same output, i.e., collisions. A naive idea would be to apply β to each digit of the state independently. The problem with this approach is that each collision for β is trivially extended to a collision for the entire non-linear layer, giving rise to differentials with DP as high as q^{-1}. They are easy to exploit as the adversary needs access to only a single input digit to generate a local collision. Similarly, any bias in the output of β is trivially present in the output of the non-linear layer, giving rise to linear approximations with LP as high as q^{-1}. They are easy to exploit as the adversary needs access to only a single output digit to exploit them. The measure of both is inversely proportional to the order of the field. Hence, unless the order of the field is very large, this leads to unacceptable weaknesses in the cryptographic transformation.

Compared to the above, the non-linear layer in the round function of a cryptographic transformation should have lower DP and LP and there should not exist local properties that can be extended to global properties. We achieve this by making the DP of differentials of the form $(a, 0)$ and the LP of linear approximations of the form $(u, 0)$ small, i.e., equal to the inverse of the domain size. Moreover, any differential over or linear approximation of the non-linear layer requires access to every digit of the state.

The work by Grassi [18] presents an analysis of a number of mappings based on β that minimize the probability of a collision in their output. We consider one of these mappings and call it γ. Concretely, the mapping $\gamma\colon \mathbb{F}_q^n \to \mathbb{F}_q^n$ is defined, for all $x \in \mathbb{F}_q^n$, by

$$\gamma_i(x) = x_i + x_{i+1}^2, \qquad i \in \mathbb{Z}/n\mathbb{Z}\,.$$

The remainder of this text is concerned with an analysis of the differential and linear propagation properties of γ.

8.7 Differential propagation properties of γ

Let $(a, b) \in \mathbb{F}_q^n \times \mathbb{F}_q^n$ be a differential over γ and let $x \in \mathbb{F}_q^n$ be an input of γ. The equations that relate the input difference a and the output difference b are of the form

$$b_i = a_i + a_{i+1}^2 + 2a_{i+1}x_{i+1}, \qquad i \in \mathbb{Z}/n\mathbb{Z}\,. \tag{8.1}$$

We consider two cases in the analysis of these equations. In the first case, we fix the input difference a and give a description of the set of compatible output differences b. From this, we are able to deduce that $\mathrm{DP}_\gamma(a, b)$ depends only on a and whether b is compatible with a or not.

In the second, reverse case, we fix the output difference b and present an algorithm for the computation of the set of compatible input differences a. We then derive an expression of the so-called minimum reverse weight of this set. All these results can be directly applied to perform computer-aided trail search, as described in [12], in cryptographic transformations instantiated with γ as the non-linear layer.

8.7.1 Forward propagation from a given input difference

We observe that for an input difference a, the equations of Equation (8.1) are linear in the digits of x. We make this explicit by writing them as a matrix equation of the form

$$\begin{pmatrix} b_0 \\ b_1 \\ b_2 \\ \vdots \\ b_{n-2} \\ b_{n-1} \end{pmatrix} = \begin{pmatrix} a_0 + a_1^2 \\ a_1 + a_2^2 \\ a_2 + a_3^2 \\ \vdots \\ a_{n-2} + a_{n-1}^2 \\ a_{n-1} + a_0^2 \end{pmatrix} + \begin{pmatrix} 0 & 2a_1 & 0 & 0 & \cdots & 0 & 0 \\ 0 & 0 & 2a_2 & 0 & \cdots & 0 & 0 \\ 0 & 0 & 0 & 2a_3 & \cdots & 0 & 0 \\ \vdots & & & & & & \\ 0 & 0 & 0 & 0 & \cdots & 0 & 2a_{n-1} \\ 2a_0 & 0 & 0 & 0 & \cdots & 0 & 0 \end{pmatrix} \begin{pmatrix} x_0 \\ x_1 \\ x_2 \\ \vdots \\ x_{n-2} \\ x_{n-1} \end{pmatrix}.$$

Hence, the set of compatible output vectors b, which we denote by $\mathscr{A}(a)$, forms an affine subspace of $\mathbb{F}_q^n$. By affine subspace we mean the following. Let W be a linear subspace of $\mathbb{F}_q^n$ and let $u \in \mathbb{F}_q^n$. The coset $u + W = \{u + w : w \in W\}$ is called an affine subspace of $\mathbb{F}_q^n$ and u is called an offset. The affine subspace $\mathscr{A}(a)$ can be described by

$$\mathscr{A}(a) = \gamma(a) + \mathrm{Span}\{2a_i e_{i-1} : i \in \mathbb{Z}/n\mathbb{Z}\}\,.$$

Two cosets $u + W$ and $v + W$ are equal if and only if $u - v \in W$. Therefore, we may add any linear combination of the basis vectors to the offset without it changing the affine subspace that is defined. Moreover, we may scale the basis vectors by any nonzero constant. Hence, a description of $\mathscr{A}(a)$ in which the offset has minimal Hamming weight is given by

$$\mathscr{A}(a) = a' + \mathrm{Span}\{e_i : i \in \mathbb{Z}/n\mathbb{Z} \text{ and } a_{i+1} \neq 0\}\,,$$

where

$$a'_i = \begin{cases} a_i & \text{if } a_{i+1} = 0\,, \\ 0 & \text{if } a_{i+1} \neq 0\,. \end{cases}$$

Clearly, the dimension of $\mathscr{A}(a)$, which is defined as the dimension of the associated vector space, is equal to the Hamming weight of a.

We are now ready to give a complete characterization of the distribution of differentials over γ.

Proposition 38. *Let $(a, b) \in \mathbb{F}_q^n \times \mathbb{F}_q^n$ be a differential over γ. Then b is compatible with a, i.e., $b \in \mathscr{A}(a)$, if and only if, for all $i \in \mathbb{Z}/n\mathbb{Z}$, we have $b_i = a_i$ or $a_{i+1} \neq 0$, in which case b_i can take on any value. Hence,*

$$\mathrm{DP}_\gamma(a, b) = \begin{cases} q^{-\mathrm{HW}(a)} & \text{if } b \in \mathscr{A}(a)\,, \\ 0 & \text{if } b \notin \mathscr{A}(a)\,. \end{cases}$$

In other words, the DP of a valid differential, and thus its differential weight, is a constant that depends only on the input difference.

8.7.2 Backward propagation from a given output difference

For a given output difference b, the compatible input differences do not form an affine space. However, we will show in this section how to efficiently generate all compatible input differences a with $\mathrm{w}_\gamma(a, b) \leq W$ for some weight limit W. To this end, we introduce the concept of compatible activity pattern. Given a vector $x \in \mathbb{F}_q^n$, its activity pattern $\tilde{x}$ is a vector in $\mathbb{F}_q^n$ with $\tilde{x}_i$ equal to 1 if $x_i \neq 0$ and 0 otherwise.

Definition 63. *An activity pattern is compatible with b if there exists an input difference a that is compatible with b and for which $\tilde{a}$ equals this activity pattern.*

The generation of all compatible input differences is done in two phases: in the first phase, we generate the set of activity patterns compatible with b, and in the second phase, we determine for each compatible activity pattern the set of compatible input differences with that pattern.

We generate the compatible activity patterns in a recursive way in Algorithm 9, making use of the following proposition.

Proposition 39. *Given a differential (a, b) over γ, the following properties hold:*

1. *if $a_i = 0$ and $b_{i-1} = 0$ then $a_{i-1} = 0$;*
2. *if $a_i = 0$ and $b_{i-1} \neq 0$ then $a_{i-1} \neq 0$.*

Proof. The two properties immediately follow from Equation 8.1. We have

$$b_{i-1} = a_{i-1} + a_i^2 + 2a_i x_i \,,$$

and $a_i = 0$ implies $b_{i-1} = a_{i-1}$. □

In Algorithm 9, we start with an empty list of compatible activity patterns L (line 4) and a fully unspecified activity pattern $\tilde{a}$ (line 6). Then we specify whether $\tilde{a}_{n-1} = 0$ (line 6) or 1 (line 7) (and thus whether a_{n-1} is active or not) and based on this we incrementally determine the activity of all other digits from a_{n-2} to a_0 using the procedure buildActivity. In this procedure, when $\tilde{a}_i = 0$ we use Proposition 39 to determine whether $\tilde{a}_{i-1} = 1$ or 0, otherwise we consider both possibilities (lines 16 and 17). When a compatible activity pattern is fully determined (i.e., when $i = 0$ is reached) then it is added to list L (line 12).

Given an output difference b and a compatible input activity pattern $\tilde{a}$, we generate all compatible differences with activity $\tilde{a}$ in Algorithm 10, making use of the following proposition.

Proposition 40. *Given a differential (a, b) over γ, the following properties hold:*

1. *if $\tilde{a}_i = 0$, then $a_i = 0$;*

Algorithm 9 Generation of input activity patterns compatible with output difference b

```
1:  Input: difference b ∈ F_q^n at output of γ and limit weight W
2:  Output: list L of activity patterns ã compatible with b at input of γ
3:
4:  L ← empty
5:  ã ← ∗^n
6:  ã_{n−1} ← 0; buildActivity(n − 1, ã, b, W)
7:  ã_{n−1} ← 1; buildActivity(n − 1, ã, b, W)
8:
9:  procedure buildActivity(i, ã, b, W)
10:     if (HW(ã) > W) then return                ▷ HW is computed on the specified part of ã
11:     if (i = 0) then
12:         if (ã_{n−1} = 1 OR b̃_0 = ã_0) then add ã to L
13:         return
14:     end if
15:     ã' ← ã
16:     if (ã_i = 1 OR b̃_{i−1} = 1) then ã'_{i−1} ← 1; buildActivity(i − 1, ã', b, W)
17:     if (ã_i = 1 OR b̃_{i−1} = 0) then ã'_{i−1} ← 0; buildActivity(i − 1, ã', b, W)
18:     return
19: end procedure
```

2. *if* $\tilde{a}_i = 1$ *and* $\tilde{a}_{i+1} = 0$, *then* $a_i = b_i$;

3. *if* $\tilde{a}_i = 1$ *and* $\tilde{a}_{i+1} = 1$, *then* a_i *can be any value in* $\mathbb{F}_q$.

Proof. The first property follows from the definition of activity pattern. The other two properties immediately follow from Equation 8.1. □

In Algorithm 10, we start with an empty list of compatible input differences L (line 4) and a fully unspecified difference a (line 5). We use the symbol $*$ when the activity of a digit is unspecified. Then we incrementally determine the value of all digits from a_0 to a_{n-1} using the procedure buildDifference. In this procedure, we use Proposition 40 to determine whether $a_i = 1$ or 0 (lines 10-12 and 16-18). When a compatible difference is fully determined (i.e., when $i = n - 1$ is reached) then it is added to list L (line 10-12).

8.7.3 Computing the minimum reverse weight of an output difference

Given an output difference b, let $\Omega(b) = \{a \in \mathbb{F}_q^n : \mathrm{DP}_\gamma(a, b) > 0\}$ be the set of input differences that are compatible with b. The differentials (a, b) over γ with $a \in \Omega(b)$ can have different weights. Following [12], the *minimum reverse weight* of an output difference b is defined by

$$\mathrm{w}_\gamma^{\mathrm{rev}}(b) = \min_{a \in \Omega(b)} \mathrm{w}_\gamma(a, b) .$$

We notice that the minimum reverse weight of a difference b at the output of γ is fully determined by its activity pattern and its compatible activity patterns with minimum Hamming weight. In particular, it can be computed as in the following Proposition, which uses the notion of *run*.

Algorithm 10 Generation of input differences compatible with output difference b and with activity pattern $\tilde{a}$

Input: difference $b \in \mathbb{F}_q^n$ at output of γ and activity pattern $\tilde{a}$
Output: list L of input differences compatible with b at input of γ with activity pattern $\tilde{a}$

$L \leftarrow$ empty
$a \leftarrow *^n$
buildDifference$(0, a, \tilde{a}, b)$

procedure buildDifference$(i, a, \tilde{a}, b)$
    if $(i = n - 1)$ **then**
        if $(\tilde{a}_i = 0)$ **then** $a'_i \leftarrow 0$; add a to L
        elsif $(\tilde{a}_i = 1$ AND $\tilde{a}_0 = 0)$ **then** $a'_i \leftarrow b_i$; add a to L
        else for each $k \in \mathbb{F}_q$ **do** $a'_i \leftarrow k$; add a to L
        return
    end if
    $a' \leftarrow a$
    if $(\tilde{a}_i = 0)$ **then** $a'_i \leftarrow 0$; buildDifference$(i + 1, a', \tilde{a}, b)$
    elsif $(\tilde{a}_i = 1$ AND $\tilde{a}_{i+1} = 0)$ **then** $a'_i \leftarrow b_i$; buildDifference$(i + 1, a', \tilde{a}, b)$
    else for each $k \in \mathbb{F}_q$ **do** $a'_i \leftarrow k$; buildDifference$(i + 1, a', \tilde{a}, b)$
end procedure

Definition 64. *Given $x \in \mathbb{F}_q^n$, a run of length ℓ in x is a sequence of ℓ active digits preceded and followed by non-active digits, i.e., it satisfies $x_i, x_{i+1}, \ldots, x_{i+\ell-1} \neq 0$ and $x_{i-1} = x_{i+\ell} = 0$ for some $i \in \mathbb{Z}/n\mathbb{Z}$.*

Proposition 41. *Given a difference b at the output of γ composed by m runs of lengths ℓ_j, with $j = 0, \ldots, m-1$, then*

$$\mathrm{w}_\gamma^{rev}(b) = \sum_{j=0}^{m-1} \lceil \ell_j / 2 \rceil .$$

Proof. For a run starting in position i and of length ℓ in b, the digit $\tilde{a}_{i+\ell-1}$ must be 1. There can be at most a single zero digit in between two active digits in the sequence $\tilde{a}_i, \tilde{a}_{i+1}, \ldots, \tilde{a}_{i+\ell-1}$. It follows that for each run of length ℓ in b, a has at least $\ell/2$ active digits if ℓ is even and $(\ell + 1)/2$ if ℓ is odd. □

8.8 Linear propagation properties of γ

In this section we analyze the correlation properties of the mapping γ, starting with the correlation of linear approximations of γ.

Proposition 42. *Let $(u, v) \in \mathbb{F}_q^n \times \mathbb{F}_q^n$ be a linear approximation of γ. We have*

$$\mathrm{C}_\gamma(u, v) = \prod_{i=0}^{n-1} \mathrm{C}_\beta(u_i - v_i, u_{i-1}) .$$

Proof. If we rewrite the correlation of a linear approximation of γ, we obtain

$$\begin{aligned}
\mathrm{C}_\gamma(u,v) &= q^{-n} \sum_{x\in\mathbb{F}_q^n} \omega^{\mathrm{Tr}\left(u^\top\gamma(x)-v^\top x\right)} \\
&= q^{-n} \sum_{x\in\mathbb{F}_q^n} \omega^{\mathrm{Tr}\left(\sum_{i=0}^{n-1} u_i(x_i+x_{i+1}^2)-v_i x_i\right)} \\
&= q^{-n} \sum_{x\in\mathbb{F}_q^n} \omega^{\mathrm{Tr}\left(\sum_{i=0}^{n-1} (u_i-v_i)x_i+u_{i-1}x_i^2\right)} \\
&= q^{-n} \sum_{x\in\mathbb{F}_q^n} \omega^{\sum_{i=0}^{n-1}\mathrm{Tr}\left((u_i-v_i)x_i+u_{i-1}x_i^2\right)} \\
&= q^{-n} \sum_{x\in\mathbb{F}_q^n} \prod_{i=0}^{n-1} \omega^{\mathrm{Tr}\left((u_i-v_i)x_i+u_{i-1}x_i^2\right)} \\
&= \prod_{i=0}^{n-1} q^{-1} \sum_{y\in\mathbb{F}_q} \omega^{\mathrm{Tr}\left((u_i-v_i)y+u_{i-1}y^2\right)} \\
&= \prod_{i=0}^{n-1} \mathrm{C}_\beta(u_i-v_i, u_{i-1}) \,.
\end{aligned}$$

□

The resulting product from Proposition 42 is non-zero if each of the factors is non-zero. Note that the correlation is non-zero if u_{i-1} is non-zero, as was discussed in Section 8.5. Additionally, if u_{i-1} is non-zero, then $v_i - u_i$ has to be equal to zero to get a non-zero correlation. In this case it should thus hold that $v_i = u_i$. From this reasoning, we can give a complete characterization of the distribution of linear approximations of γ.

Proposition 43. *Let $(u,v) \in \mathbb{F}_q^n \times \mathbb{F}_q^n$ be a linear approximation of γ. Then u is compatible with v, if and only if, for all $i \in \mathbb{Z}/n\mathbb{Z}$, we have $v_i = u_i$ or $u_{i-1} \neq 0$, in which case v_i can take on any value. Hence,*

$$\mathrm{LP}_\gamma(u,v) = \begin{cases} q^{-\mathrm{HW}(u)} & \text{if } v \text{ is compatible with } u\,, \\ 0 & \text{if } v \text{ is not compatible with } u\,. \end{cases}$$

Observe that Proposition 38 and Proposition 43 are very much alike. Indeed, propagation of differences and propagation of masks over γ follow similar rules. First, output masks play the role of input differences and input masks that of output differences. Second, indices are reversed, i.e., index i in a mask corresponds to index $n-i-1$ in a difference, to account for this change in direction. The following proposition is an immediate consequence.

Proposition 44. *Let $\pi\colon \mathbb{F}_q^n \to \mathbb{F}_q^n$ be the mapping defined by $\pi_i(x) = x_{n-i-1}$ for all $i \in \mathbb{Z}/n\mathbb{Z}$. Let (u,v) be a linear approximation of γ. We have*

$$\mathrm{LP}_\gamma(u,v) = \mathrm{DP}_\gamma(\pi(u),\pi(v)) \,.$$

From this, it follows that we can extend the results obtained in Section 8.7 to masks. For a given output mask $u \in \mathbb{F}_q^n$, we can build the affine subspace with dimension $\mathrm{HW}(u)$ of compatible input masks over γ as in Section 8.7.1. Moreover, for a given input mask $v \in \mathbb{F}_q^n$, the output activity patterns compatible with input masks over γ can be found by applying Algorithm 9. Using the resulting activity pattern $\tilde{a}$ and the input mask v, all compatible output masks u can be obtained as described in Algorithm 10. Note that there can be several compatible output masks u for a given input mask v. Among them, there will be one realizing the minimum value of $\mathrm{w}(u, v)$. The *minimum reverse weight* of v is defined as

$$\mathrm{w}_\gamma^{\mathrm{rev}}(v) = \min_{u:\mathrm{LP}_\gamma(u,v)>0} \mathrm{w}_\gamma(u, v)$$

and is determined by the decomposition of v in a sequence of runs, as explained in Section 8.7.3.

8.9 On collision probability and bias

A collision in the output of γ occurs when γ maps a pair of different inputs $(x, y) \in \mathbb{F}_q^n \times \mathbb{F}_q^n$ to the same output value. Assuming randomly and uniformly selected pairs of inputs, the probability of a collision is given by

$$\mathrm{CP}(\gamma) = q^{-2n} |\{(x, y) \in \mathbb{F}_q^n \times \mathbb{F}_q^n : x \neq y \text{ and } \gamma(x) = \gamma(y)\}| .$$

Translating this into the language of differential analysis, we find that

$$\mathrm{CP}(\gamma) = q^{-n} \sum_{a \in \mathbb{F}_q^n \setminus \{0\}} \mathrm{DP}_\gamma(a, 0) .$$

Proposition 45. *Let $a \in \mathbb{F}_q^n \setminus \{0\}$. If $(a, 0)$ is a differential with $\mathrm{DP}_\gamma(a, 0) > 0$, then all digits of a are active and $\mathrm{DP}_\gamma(a, 0) = q^{-n}$.*

Proof. Let $a \in \mathbb{F}_q^n \setminus \{0\}$ be such that $\mathrm{DP}_\gamma(a, 0) > 0$. The input difference a is compatible with the output difference 0 if the latter is contained in the affine space $A(a)$. This is the case if and only if $a_i \neq 0$ for $i \in \mathbb{Z}/n\mathbb{Z}$. Hence, $\mathrm{DP}_\gamma(a, 0) = q^{-n}$ by Proposition 38. □

Clearly, there are $(q-1)^n$ input differences a for which this property holds. Therefore, we find that

$$\mathrm{CP}(\gamma) = (q-1)^n q^{-2n} .$$

Now, the collision probability of a function that is chosen randomly from the set of functions from $\mathbb{F}_q^n$ to $\mathbb{F}_q^n$ is equal to q^{-n}. Hence, the ratio between the collision probability of γ and that of a random function is equal to $(1 - q^{-1})^n$. If the order of the field is large, then this quantity approximates 1.

By symmetry, we obtain a similar result for the bias of any linear combination of output digits of γ.

Proposition 46. *Let $u \in \mathbb{F}_q^n \setminus \{0\}$. If $(u, 0)$ is a linear approximation with $\mathrm{LP}_\gamma(u, 0) > 0$, then all digits of u are active and $\mathrm{LP}_\gamma(u, 0) = q^{-n}$.*

Clearly, if either q or n is large, then these quantities are very small and it becomes difficult to exploit them in practice.

8.10 Conclusion

When searching for trails over an iterated cryptographic transformation as described in [12], a number of tools are required. These include an efficient method to compute the minimum reverse weight of a given difference (resp. mask), and an efficient method to build all compatible input differences (resp. output masks) over the non-linear layer for a given output difference (resp. input mask) and vice versa. In this work we provided such tools for a mapping based on squaring that can be used as non-linear layer in the construction of cryptographic transformations of $\mathbb{F}_q^n$. Interestingly, it turns out that for this mapping, masks and differences follow the same propagation rules. This means that for a cryptographic transformation that uses this mapping as the non-linear layer in its round function, one would need to only perform either differential or linear trail search while obtaining insights and bounds for both.

Acknowledgments. Joan Daemen and Daniël Kuijsters are supported by the European Research Council under the ERC advanced grant agreement under grant ERC-2017-ADG Nr. 788980 ESCADA.

Silvia Mella is supported by the European Commission through the ERC Starting Grant 805031 (EPOQUE).

References

1. M. R. Albrecht, L. Grassi, L. Perrin, S. Ramacher, C. Rechberger, D. Rotaru, A. Roy, and M. Schofnegger. "Feistel Structures for MPC, and More". In: *ESORICS*. Vol. 11736. LNCS. Springer, 2019, pp. 151–171.
2. M. R. Albrecht, L. Grassi, C. Rechberger, A. Roy, and T. Tiessen. "MiMC: Efficient Encryption and Cryptographic Hashing with Minimal Multiplicative Complexity". In: *ASIACRYPT*. Vol. 10031. LNCS. 2016, pp. 191–219.
3. T. Baignères, J. Stern, and S. Vaudenay. "Linear Cryptanalysis of Non Binary Ciphers". In: *Selected Areas in Cryptography*. Ed. by C. Adams, A. Miri, and M. Wiener. Springer Berlin Heidelberg, Heidelberg, Germany, 2007, pp. 184–211. DOI: 10.1007/978-3-540-77360-3_13. URL: https://doi.org/10.1007/978-3-540-77360-3_13.
4. T. P. Berger, J. D'Hayer, K. Marquet, M. Minier, and G. Thomas. "The GLUON Family: A Lightweight Hash Function Family Based on FCSRs". In: *Progress in Cryptology - AFRICACRYPT 2012 - 5th International Conference on Cryptology in Africa, Ifrance, Morocco, July 10-12, 2012. Proceedings*. Ed. by A. Mitrokotsa and S. Vaudenay. Vol. 7374. Lecture Notes in Computer Science. Springer, 2012, pp. 306–323. DOI: 10.1007/978-3-642-31410-0_19. URL: https://doi.org/10.1007/978-3-642-31410-0%5C_19.

5. G. Bertoni, J. Daemen, S. Hoffert, M. Peeters, G. Van Assche, and R. Van Keer. “Farfalle: parallel permutation-based cryptography”. *IACR Trans. Symmetric Cryptol.* 2017:4, 2017, pp. 1–38.

6. G. Bertoni, J. Daemen, M. Peeters, and G. V. Assche. *Duplexing the sponge: single-pass authenticated encryption and other applications*. Cryptology ePrint Archive, Paper 2011/499. https://eprint.iacr.org/2011/499. 2011. URL: https://eprint.iacr.org/2011/499.

7. G. Bertoni, J. Daemen, M. Peeters, and G. V. Assche. “On the Indifferentiability of the Sponge Construction”. In: *Advances in Cryptology - EUROCRYPT 2008, 27th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Istanbul, Turkey, April 13-17, 2008. Proceedings*. Ed. by N. P. Smart. Vol. 4965. Lecture Notes in Computer Science. Springer, 2008, pp. 181–197. DOI: 10.1007/978-3-540-78967-3_11. URL: https://doi.org/10.1007/978-3-540-78967-3%5C_11.

8. T. Beyne, Y. L. Chen, C. Dobraunig, and B. Mennink. “Dumbo, Jumbo, and Delirium: Parallel Authenticated Encryption for the Lightweight Circus”. *IACR Trans. Symmetric Cryptol.* 2020:S1, 2020, pp. 5–30. DOI: 10.13154/tosc.v2020.iS1.5-30. URL: https://doi.org/10.13154/tosc.v2020.iS1.5-30.

9. E. Biham and A. Shamir. “Differential Cryptanalysis of DES-like Cryptosystems”. In: *Advances in Cryptology - CRYPTO ’90, 10th Annual International Cryptology Conference, Santa Barbara, California, USA, August 11-15, 1990, Proceedings*. Ed. by A. Menezes and S. A. Vanstone. Vol. 537. Lecture Notes in Computer Science. Springer, 1990, pp. 2–21. DOI: 10.1007/3-540-38424-3_1. URL: https://doi.org/10.1007/3-540-38424-3%5C_1.

10. C. Carlet, ed. *Boolean Functions for Cryptography and Coding Theory*. Cambridge University Press, 2020. ISBN: 9781108606806. DOI: 10.1017/9781108606806. URL: https://doi.org/10.1017/9781108606806.

11. R. S. Coulter and R. W. Matthews. “Bent polynomials over finite fields”. *Bulletin of the Australian Mathematical Society* 56:3, 1997, pp. 429–437. DOI: 10.1017/S000497270003121X.

12. J. Daemen and G. V. Assche. “Differential Propagation Analysis of Keccak”. In: *Fast Software Encryption - 19th International Workshop, FSE 2012, Washington, DC, USA, March 19-21, 2012. Revised Selected Papers*. Ed. by A. Canteaut. Vol. 7549. Lecture Notes in Computer Science. Springer, 2012, pp. 422–441. DOI: 10.1007/978-3-642-34047-5_24. URL: https://doi.org/10.1007/978-3-642-34047-5%5C_24.

13. J. Daemen, R. Govaerts, and J. Vandewalle. “Correlation Matrices”. In: *Fast Software Encryption: Second International Workshop. Leuven, Belgium, 14-16 December 1994, Proceedings*. Ed. by B. Preneel. Vol. 1008. Lecture Notes in Computer Science. Springer, 1994, pp. 275–285. DOI: 10.1007/3-540-60590-8_21. URL: https://doi.org/10.1007/3-540-60590-8%5C_21.

14. J. Daemen and V. Rijmen. "Correlation Analysis in GF(2^n)". In: *The Design of Rijndael: The Advanced Encryption Standard (AES)*. Springer Berlin Heidelberg, Heidelberg, Germany, 2020, pp. 181–194. DOI: 10.1007/978-3-662-60769-5_12. URL: https://doi.org/10.1007/978-3-662-60769-5_12.

15. C. Dobraunig, L. Grassi, A. Guinet, and D. Kuijsters. *Ciminion: Symmetric Encryption Based on Toffoli-Gates over Large Finite Fields*. Advances in Cryptology - EUROCRYPT 2021.

16. S. Even and Y. Mansour. "A Construction of a Cipher from a Single Pseudorandom Permutation". *J. Cryptol.* 10:3, 1997, pp. 151–162. DOI: 10.1007/S001459900025. URL: https://doi.org/10.1007/s001459900025.

17. L. Grassi, D. Khovratovich, C. Rechberger, A. Roy, and M. Schofnegger. *Poseidon: A New Hash Function for Zero-Knowledge Proof Systems*. 30th USENIX Security Symposium. 2021.

18. L. Grassi. "Bounded Surjective Quadratic Functions over Fnp for MPC-/ZK-/FHE-Friendly Symmetric Primitives". *IACR Trans. Symmetric Cryptol.* 2023:2, 2023, pp. 94–131. DOI: 10.46586/TOSC.V2023.I2.94-131. URL: https://doi.org/10.46586/tosc.v2023.i2.94-131.

19. X.-d. Hou. *Lectures on Finite Fields*. American Mathematical Society, 2018.

20. S. Kölbl, E. Tischhauser, P. Derbez, and A. Bogdanov. "Troika: a ternary cryptographic hash function". *Designs, Codes and Cryptography* 88:1, 2019, pp. 91–117. DOI: 10.1007/s10623-019-00673-2. URL: https://doi.org/10.1007/s10623-019-00673-2.

21. R. Lidl and H. Niederreiter. *Finite Fields*. 2nd ed. Vol. 20. Cambridge University Press, Cambridge, United Kingdom, 1997.

22. N. I. of Standards and Technology. *NIST SP 800-38A Recommendation for Block Cipher Modes of Operation: Methods and Techniques*. November 2007. URL: https://csrc.nist.gov/pubs/sp/800/38/a/final.

23. N. I. of Standards and Technology. *NIST SP 800-38D Recommendation for Block Cipher Modes of Operation: Galois/Counter Mode (GCM) and GMAC*. November 2007. URL: https://csrc.nist.gov/pubs/sp/800/38/d/final.

Part III

Miscellany

This part contains a summary of the thesis in the Dutch language, a description of the research data management, and the curriculum vitae of the author.

9 Samenvatting

Symmetrische cryptografie stelt twee partijen in staat om vertrouwelijke en integere communicatie tot stand te brengen op basis van een gedeelde geheime sleutel. Vercijfering waarborgt vertrouwelijkheid; het toevoegen van een authenticatietag beschermt de integriteit. Afhankelijk van het dreigingsmodel richt de analyse van een symmetrische primitieve zich op het aantonen van onvoorwaardelijke veiligheid, dan wel op veiligheid onder de aanname van een aanvaller met beperkte middelen.

Dit proefschrift onderzoekt de rol van structuur en willekeur in het ontwerp en de analyse van symmetrische primitieven. *Structured Randomness*, ofwel gestructureerde willekeur, duidt op het streven om constructies te ontwerpen die, ondanks hun structuur, voor begrensde aanvallers niet te onderscheiden zijn van willekeurig gekozen functies.

Het eerste deel ontwikkelt een theoretisch kader, waarin algebraïsche structuren, kansmodellen en klassieke analysetechnieken, zoals differentiële, lineaire, integrale en algebraïsche crypto-analyse, aan bod komen. Op basis hiervan worden in het tweede deel vijf bijdragen gepresenteerd, elk met een gerichte aanvulling op de bestaande literatuur.

De eerste bijdrage formaliseert het bestaande begrip alignment; het onderscheid tussen *aligned* en *unaligned* primitieven ligt in de structurele groepering van bits. Er wordt een analysekader ontwikkeld voor het evalueren van de interactie tussen lineaire en niet-lineaire lagen met betrekking tot differentiële en lineaire propagatie. Een empirische vergelijking suggereert dat alignment aanleiding geeft tot specifieke vormen van clustering in activiteitspatronen en padstructuren.

De tweede bijdrage analyseert tweerondige varianten van de blokcijfer Skinny, waarbij lineaire benaderingen met absolute correlatie één worden aangetoond voor een aanzienlijk deel van de mogelijke ronde-*tweakeys*. Daarnaast wordt aangetoond hoe deze kwetsbaarheden vermeden hadden kunnen worden door alternatieve ontwerpkeuzes. Deze analyse onderstreept dat het niet voldoende is om op zichzelf staande veilige bouwblokken te gebruiken; het is de combinatie en compositie van deze bouwblokken die uiteindelijk de veiligheid van het geheel bepaalt.

De derde bijdrage introduceert Koala, een pseudowillekeurige functie met lage latentie, gebaseerd op de Kirby-constructie en een aangepaste variant van de Subterranean-permutatie. Het ontwerp is geoptimaliseerd voor ASIC-implementaties. Een voorlopige crypto-analyse toont een hoge weerstand tegen integrale, cube-, division property- en hogere-orde differentiële aanvallen. Daarnaast blijkt uit vergelijking dat Koala bestaande lage-latentie PRF's overtreft in termen van latentie en andere prestatiekenmerken.

De vierde bijdrage introduceert Ciminion, een cryptosysteem gebaseerd op Toffoli-poorten over eindige lichamen. Het ontwerp is specifiek geoptimaliseerd voor gebruik in privacybeschermende reken-

modellen, waaronder *multi-party computation* (MPC) en homomorfe vercijfering, met bijzondere aandacht voor het minimaliseren van het aantal vermenigvuldigingen.

De vijfde bijdrage analyseert een niet-lineaire transformatie gebaseerd op kwadrateren in eindige lichamen met oneven karakteristiek. Deze afbeelding vertoont symmetrische en gunstige propagatie-eigenschappen in zowel differentieel als lineair opzicht, wat haar geschikt maakt voor toepassingen in onder meer MPC en homomorfe vercijfering.

Gezamenlijk positioneren deze bijdragen zich op het raakvlak van theorie en praktijk, en dragen zij bij aan een verdiept begrip van de wijze waarop structurele eigenschappen de veiligheid, efficiëntie en toepasbaarheid van symmetrische cryptografie bepalen.

10 Research data management

Many of the results presented in this thesis are based on data generated through simulations involving cryptographic building blocks.

Throughout the project, data were securely stored and version controlled on Radboud University infrastructure. For long-term preservation, the final datasets and the programs used to generate them have been archived on Zenodo and are accessible via the following DOI:

`https://doi.org/10.5281/zenodo.14030442`

The data and programs are publicly available under the Creative Commons Zero v1.0 Universal (CC0 1.0) license and are accompanied by metadata that adhere to the FAIR principles (Findable, Accessible, Interoperable, and Reusable). This approach ensures compliance with the standards of the academic field and with Radboud University's guidelines regarding transparency, accessibility, and the responsible management of research data.

II Curriculum vitae

Daniël Kuijsters was born on November 18, 1989, in Waalwijk, the Netherlands. He graduated *cum laude* in 2012 with a Bachelor's degree in Computer Science and Engineering, with a minor in Applied Mathematics, from Eindhoven University of Technology.

He subsequently pursued a Master's degree in Industrial and Applied Mathematics at the same university, specializing in discrete mathematics, and graduated *cum laude* in 2017.

In 2019, he began his doctoral research at Radboud University Nijmegen, under the supervision of prof. dr. Joan Daemen. His research focused on symmetric cryptography, resulting in several publications in cryptographic venues.

Before commencing his doctoral research, Daniël spent two years at Compumatica. During his doctoral research, he completed internships at Intrinsic ID, Intel Labs, and Cloudflare, gaining additional experience in industry.

Radboud Universiteit