Master thesis
Computing Science
Cyber Security

Radboud University

Designing q-ary Transformations for Symmetric Cryptography

Author:
Denise Verbakel
s1018597

First supervisor/assessor:
Prof. J.J.C. Daemen (Joan)
j.daemen@cs.ru.nl

Second supervisor:
Dr. S. Mella (Silvia)
silvia.mella@ru.nl

Third supervisor:
ir. D.W.C. Kuijsters (Daniël)
d.kuijsters@cs.ru.nl

Second assessor:
Dr. B.J.M. Mennink (Bart)
b.mennink@cs.ru.nl

June 27, 2023

Abstract

Designing an efficient transformation $f\colon \mathbb{F}_q^n \to \mathbb{F}_q^n$ is not as obvious as it seems. It all starts with defining a round function, which consists of a non-linear layer and several linear layers, that has good propagation properties. These propagation properties are well-studied over (extensions of) binary fields, but not over (extensions of) prime fields. Combining this with the emerging trend of using primitives that handle such fields in symmetric cryptography for multi-party computation (MPC), fully homomorphic encryption (FHE) and zero-knowledge (ZK) environments, it is deemed useful and important to look into the generalizations of these concepts. In this research, we formalize the metrics belonging to the avalanche behavior over $\mathbb{F}_q$ with q an odd prime power for the first time. We also define differential cryptanalysis over the field $\mathbb{F}_q$. Linear cryptanalysis is generalized to $\mathbb{F}_p$ from the binary case. Besides this, we consider a family of transformations over $\mathbb{F}_q^n$ called $\mathcal{T}_q$ of which the simple yet efficient non-linear layer is defined as $y_i \leftarrow x_i + x_{i+1}^2$. The parameters used in the round function of $\mathcal{T}_q$ are determined by investigating the avalanche behavior of $\mathcal{T}_q$. We also look into the differential and linear propagation properties of this family of transformations and find an intriguing property that is believed to only occur in our non-linear layer and the mapping χ_3 of Xoodoo. Additionally, we elaborate upon a certain instance of $\mathcal{T}_q$: we show an efficient software implementation with dedicated arithmetic using counter-intuitive shift offsets in the shuffle layer.

Contents

Chapter 1

Introduction

Most primitives in symmetric cryptography use a permutation, or even a transformation, which is constructed using a round function. This round function is repeated for a certain number of rounds and, in turn, consists of a non-linear mapping and multiple linear mappings. It is important for a round function in such a permutation or transformation to be deemed strong enough against various attacks. Examples of three common measures to evaluate this property are the avalanche behavior (diffusion analysis), differential cryptanalysis and linear cryptanalysis. The goal of these analyses is to study the behavior of a certain cryptographic design with reference to these properties. In the ideal case, the avalanche behavior indicates that each output bit depends on all input bits, the differential cryptanalysis concludes that there is no exploitable differential propagation from input to output and the linear cryptanalysis results in not having any exploitable correlations between input and output present. These three analyses are well-studied for binary fields, but not in much depth for other fields, such as prime fields.

In symmetric cryptography there is an emerging trend of using primitives that handle (extensions of) prime field $\mathbb{F}_p$. These primitives are then used in multi-party computation (MPC), fully homomorphic encryption (FHE) and zero-knowledge (ZK) environments. Examples of such primitives are Troika (operating in $\mathbb{F}_3$) [32], MiMC (operating in $\mathbb{F}_p$ with p either prime or a power of 2) [1] and Poseidon (operating in $\mathbb{F}_p$ with p prime) [28]. The design rationale of these last two primitives briefly touches on the topic of differential and linear cryptanalysis, and for Troika a more comprehensive approach is taken with respect to differential cryptanalysis [15]. The other analysis, investigating the avalanche behavior, is not discussed for these or any other primitives.

It is considered to be useful and important to look into these concepts over $\mathbb{F}_q$, with $q = p^l$ an odd prime power (using prime p and natural number l), for future use in MPC, FHE and/or ZK environments. Therefore, we address the following research question in this thesis:

"How can we design an efficient q-ary transformation with good propagation properties?"

Note that we define a q-ary transformation $f : \mathbb{F}_q^n \to \mathbb{F}_q^n$ as an assignment of each element of $\mathbb{F}_q^n$ to some element in $\mathbb{F}_q^n$ [44].

This is particularly interesting because of the richer arithmetic implied by an odd prime power field: the standard operation of squaring is no longer a linear operation, but can potentially be a very efficient non-linear operation. We want to investigate the mapping $y_i \leftarrow x_i + x_{i+1}^2$ as partly introduced in [27] as the non-linear layer of a round function and see how the propagation properties of the full round function, i.e., the non-linear layer combined with some linear layers, perform.

1.1 Related Work

This thesis generalizes the concepts of the avalanche behavior as discussed in [21, 45, 49], the differential cryptanalysis as discussed in [10, 12, 14, 18, 45] and the linear cryptanalysis discussed in [4, 11, 14, 18, 21, 25, 34, 35, 36]. In particular, the method we follow to do differential and linear trail search was introduced in [20, 30]. As part of our non-linear layer γ, we use the map introduced in [27] that is based on squaring.

In Chapter 8, an implementation of a ternary transformation is investigated. This transformation was coded using insights on trit encoding and arithmetic in $\mathbb{F}_3$ as shown in, for instance, [13] and [31].

1.2 Our Contribution

In this work, we present the formalization of the avalanche behavior over the field $\mathbb{F}_q$ which is interesting and useful because no one has done this formally before. Then, to capitalize on the emerging trend of symmetric cryptography over (extensions of) primary fields, the differential cryptanalysis is defined for q-ary transformations and the linear cryptanalysis is generalized from the binary case to be applicable to p-ary transformations (with p a prime). Note that, in this case, linear cryptanalysis does not work in the range -1 to $+1$ as was the standard for binary primitives, but correlations become complex numbers in the unit circle using the root $\omega = e^{\frac{2\pi i}{p}}$.

Besides this, we want to investigate the promising non-linear mapping based on squaring, which we define as $y_i \leftarrow x_i + x_{i+1}^2$. Combining this non-linear mapping with some other linear operations to form our round function,

we define a family of q-ary transformations. This family will be called $\mathcal{T}_q$. To build an efficient set of transformations, we look at the avalanche behavior of $\mathcal{T}_q$ to determine which parameters to use in its round function. We also investigate the differential and linear propagation properties of it and find a dual relation of the DP and LP, which is believed to only occur in our non-linear layer γ and the mapping χ_3 of Xoodoo. Along this, we provide tools in order to perform both differential and linear trail search on $\mathcal{T}_q$. These tools are for finding the minimum reverse weight, showing that the Hamming weight represents the minimum direct weight and how to find compatible input and output differences and/or masks.

Moreover, we look at practical applications using this family of transformations. We also shed light on a particular transformation τ: we show how one could possibly implement this transformation in code (by using counter-intuitive shift offsets in the shuffle layer) and perform the aforementioned analyses on the instance τ.

1.3 Outline

This thesis consists of 9 chapters of which the introduction forms the first one. The next chapter, Chapter 2, discusses all the preliminaries needed to understand our conducted research. After this, Chapter 3 introduces the structure of the family of transformations called $\mathcal{T}_q$. In the following chapters we will investigate properties of $\mathcal{T}_q$: in Chapter 4 we look at the avalanche behavior, in Chapter 5 we study the differential propagation and in Chapter 6 we explore the linear propagation. Then, Chapter 7 discusses some practical applications for $\mathcal{T}_q$. Following on this, we provide a case study on $\mathcal{T}_3$ (transformations for which $q = 3$) in Chapter 8. Here, we show how to design and implement τ, which is an instance of this family of transformations. Finally, Chapter 9 concludes our thesis.

Chapter 2

Preliminaries

Before defining $\mathcal{T}_q$ and investigating its propagation properties, we need some background information about the research we will be conducting. We start with introducing the field $\mathbb{F}_q$ and its properties in Section 2.1. After this, we will cover what a vector space and an affine space is: see respectively Section 2.2 and Section 2.3. Then, we explain what a (q-ary) transformation is and how we can use such function: see Section 2.4. In Section 2.5, we will introduce three important functions used throughout this research. Lastly, the terminology and notation regarding complex numbers are given in Section 2.6.

2.1 Finite Field $\mathbb{F}_q$

Let $q = p^l$ be an odd prime power with p a prime and $l \geq 1$. Then, let $\mathbb{F}_q$ be a *finite field*, which is defined as a set with a finite number of elements "in which four operations (called addition, multiplication, subtraction, and division) can be defined so that, with the exception of division by zero, the sum, product, difference, and quotient of any two elements in the set is an element of the set" [26]. A finite field is a set that satisfies the field axioms [47] given in Table 1.

In this research, we will give examples for a specific field, namely the finite field with $p = 3$, $l = 1$ and thus $q = 3$, i.e., $\mathbb{F}_3$. Elements of this field are called *trits* [17]. Note that we call elements of $\mathbb{F}_2$ *bits* and, in general, elements of $\mathbb{F}_q$ (especially for $q \notin \{2, 3\}$) *digits*.

Note that if $l = 1$ and thus $q = p$, $\mathbb{F}_q$ is isomorphic to $(\mathbb{Z}/p\mathbb{Z}, +, \times)$, which means that addition and multiplication are performed modulo p. Subtraction and division can then be done by respectively using the additive and multiplicative inverse.

When $l > 1$, the elements of $\mathbb{F}_q$ can be represented as polynomials with a degree smaller than l and with coefficients in $\mathbb{F}_p$. Addition, multiplication, subtraction and division should now be performed modulo an irreducible

Table 1: The nine axioms for a finite field [47].

#	**Finite field axioms**
1.	Associative law of addition: $a + (b + c) = (a + b) + c$.
2.	Commutative law of addition: $a + b = b + a$.
3.	Existence of additive identity: $a + 0 = 0 + a = a$.
4.	Existence of additive inverse: $a + (-a) = (-a) + a = 0$.
5.	Associate law of multiplication: $a \cdot (b \cdot c) = (a \cdot b) \cdot c$.
6.	Commutative law of multiplication: $a \cdot b = b \cdot a$.
7.	Existence of multiplicative identity: $a \cdot 1 = 1 \cdot a = a$ where $1 \neq 0$.
8.	Existence of multiplicative inverse: $a \cdot a^{-1} = a^{-1} \cdot a = 1$ for $a \neq 0$.
9.	Distributive laws: $a \cdot (b + c) = a \cdot b + a \cdot c$ and $(b + c) \cdot a = b \cdot a + c \cdot a$.

polynomial $p(X) \in \mathbb{F}_q[X]$ of degree l. This is a polynomial that is only divisible by 1 and itself, i.e., there does not exist any $g(X), h(X) \in \mathbb{F}_q[X]$ such that $p(X) = g(X) \cdot h(X)$ [22]. So, to multiply two elements modulo such irreducible polynomial, one should perform two steps: take the product of the two polynomials and after this, take the remainder of the result after division by $p(X)$ [22]. As an example, we get the following result if we multiply the polynomial $X + X^2$ with $1 + X^2$ over $\mathbb{F}_{2^4}$ by using irreducible polynomial $p(X) = 1 + X + X^4$:

$$\begin{aligned}\left(X + X^2\right)\left(1 + X^2\right) &\equiv X + X^3 + X^2 + X^4 \pmod{p(X)} \\ &\equiv X + X^3 + X^2 + (-1 - X) \pmod{p(X)} \\ &\equiv X + X^3 + X^2 + 1 + X \pmod{p(X)} \\ &\equiv 1 + X^2 + X^3 \pmod{p(X)}.\end{aligned}$$

2.2 Vector Spaces

After having defined the finite field $\mathbb{F}_q$, we define $\mathbb{F}_q^n$ as a *vector space* of dimension n over the finite field $\mathbb{F}_q$ when it satisfies the axioms shown in Table 2. A vector space is a "collection V of objects, $\boldsymbol{u}$, $\boldsymbol{v}$, etc., called vectors, such that we are given a binary operation, +, which assigns to every pair of vectors $\boldsymbol{u}$ and $\boldsymbol{v}$ a third vector $\boldsymbol{u} + \boldsymbol{v}$" [5]. Likewise, a multiplication assigns to every scalar $t \in \mathbb{F}_p$ and every vector $\boldsymbol{v} \in \mathbb{F}_q$ another vector $t\boldsymbol{v} \in \mathbb{F}_q$ [5].

Table 2: The six axioms for a vector space [5].

#	**Vector space axioms**
1.	Associative law of addition: $(\boldsymbol{u} + \boldsymbol{v}) + \boldsymbol{w} = \boldsymbol{u} + (\boldsymbol{v} + \boldsymbol{w})$.
2.	Commutative law of addition: $\boldsymbol{u} + \boldsymbol{v} = \boldsymbol{v} + \boldsymbol{u}$.
3.	Existence of additive identity: there is a vector $\mathbf{0}$ such that $\mathbf{0} + \boldsymbol{v} = \boldsymbol{v}$ for all $\boldsymbol{v}$.
4.	Existence of additive inverse: for every $\boldsymbol{v}$ there is a $-\boldsymbol{v}$ such that $\boldsymbol{v} + (-\boldsymbol{v}) = \mathbf{0}$.
5.	'One' acts as multiplicative identity: $1\boldsymbol{v} = \boldsymbol{v}$ for every $\boldsymbol{v}$.
6.	Associative and distributive laws: for any scalars r and s and any vectors $\boldsymbol{u}$ and $\boldsymbol{v}$ it holds that $(rs)\boldsymbol{v} = r(s\boldsymbol{v})$, $(r + s)\boldsymbol{v} = r\boldsymbol{v} + s\boldsymbol{v}$ and $r(\boldsymbol{u} + \boldsymbol{v}) = r\boldsymbol{u} + r\boldsymbol{v}$.

2.2.1 Standard Basis e_i

We define a *standard basis* e_i of length n over the finite field $\mathbb{F}_q$ as a vector which has a single 1 on position i and 0 in all other positions. Note that one can also define the standard basis e_i^n as the standard basis vector in $\mathbb{F}_q^n$, where $e_{ij}^n = 1$ if $i = j$ (and 0 otherwise) [14].

2.2.2 Activity Patterns

Let $x \in \mathbb{F}_q^n$. The *activity pattern of* x is a vector in $\mathbb{F}_q^n$ whose i^{th} digit equals to 1 if $x_i \neq 0$ and is 0 otherwise [14]. If the i^{th} digit of the activity pattern equals to 1, the digit of x is called *active*. Likewise, if the i^{th} digit of the activity pattern equals to 0, the digit of x is called *passive*. If the i^{th} digit of x has not been assigned a value and is thus neither active nor passive, it is called *unspecified*. This is denoted with the value $*$.

To illustrate this concept, take $q = 3$ and $n = 5$. The vector $x \in \mathbb{F}_3^5$ can, for instance, take on the values 20210. The activity pattern of x is then equal to 10110.

2.2.3 Runs

After defining what active, passive and unspecified digits are, one can define a 1-run. We define a 1-run as a sequence of active digits preceded by at least one passive digit and followed by at least one passive digit. Note that a g-run can then be defined by having a sequence of active digits with a step size g instead of 1 in between them.

Again take $q = 3$ and $n = 5$ with $x = 20210$ (such that $x \in \mathbb{F}_3^5$). In this vector, two 1-runs are present: the single 2 at the beginning and the sequence 21 in the middle of x. Note that if we would look at, for instance, the 2-runs within x this would only yield one run: the sequence 122. This

is because the first and last value of x are also seen as neighboring values: runs can thus wrap around the length of x.

2.3 Affine Spaces

An *affine space* is a translation of a vector space as defined above. In other words, an affine space is a "set A consisting of points P, Q, etc., and an operation + which assigns to each $P \in A$ and each $\boldsymbol{v} \in V$ another point in A which is denoted by $P + \boldsymbol{v}$" [5]. Here, V is a vector space as defined in Section 2.2. Affine spaces follow the axioms as listed in Table 3.

Table 3: The four axioms for an affine space [5].

#	**Affine space axioms**
1.	Associative law: $(P + \boldsymbol{u}) + \boldsymbol{v} = P + (\boldsymbol{u} + \boldsymbol{v})$ for any $P \in A$ and $\boldsymbol{u}, \boldsymbol{v} \in V$.
2.	'Zero' acts as identity: $P + \mathbf{0} = P$ for any $P \in A$.
3.	Transitivity: given any two points $P, Q \in A$, there is a $\boldsymbol{v} \in V$ such that $P + \boldsymbol{v} = Q$.
4.	Faithfulness: if, for any P, the equality $P + \boldsymbol{u} = P + \boldsymbol{v}$ holds, then $\boldsymbol{u} = \boldsymbol{v}$.

Note that the axioms 3 and 4 indicate that if we are given two points P and Q, there exists a *unique* vector $\boldsymbol{v}$ such that $P + \boldsymbol{v} = Q$ [5].

2.4 Transformations

A *q-ary transformation* $f\colon \mathbb{F}_q^n \to \mathbb{F}_q^n$ is an assignment of each element of $\mathbb{F}_q^n$ to some element in $\mathbb{F}_q^n$ [44]. Note that a transformation does not need to be invertible, unlike a permutation $f'\colon \mathbb{F}_q^n \to \mathbb{F}_q^n$, which is an ordered arrangement of the elements of the set $\mathbb{F}_q^n$ [44].

In particular, we consider a *ternary transformation*. This is a transformation f defined using $q = 3$, i.e., we have that $f\colon \mathbb{F}_3^n \to \mathbb{F}_3^n$. A *binary transformation* can be defined similarly by using $q = 2$.

2.4.1 Iterated Transformations

A q-ary transformation f can be built by composing a number of lightweight *round functions* R_i. This can be denoted as $f[k] = \mathrm{R}_{k-1} \circ \cdots \circ \mathrm{R}_1 \circ \mathrm{R}_0$ for some $k \geq 0$ [14]. Note that $f[0] = \mathrm{id}$, where id represents the identity function [14]. We then talk about an *iterated transformation*. Transformations, iterated or not, can be useful in, for instance, a sponge or a duplex construction [6]: see Section 7 for more practical applications.

2.5 Functions

In this research, we will use some standard mathematical functions. Examples of these are the Hamming weight function, the Kronecker delta and the transpose function, which will be explained in the next sections.

2.5.1 Hamming Weight

Assuming a vector space $\mathbb{F}_q^n$, the *Hamming weight* of a vector $\boldsymbol{v}$, denoted as $\mathrm{HW}(\boldsymbol{v})$, is equal to the number of non-zero entries in $\boldsymbol{v}$ [3]. Note that, for binary vectors, this means that the Hamming weight is equal to the number of ones in it.

2.5.2 Kronecker Delta

The *Kronecker delta of two variables* $i, j \in \mathbb{Z}$ [33] is defined as follows:

$$\delta_{ij} = \begin{cases} 1 & \text{if } i = j\,; \\ 0 & \text{if } i \neq j\,. \end{cases}$$

Another form of the Kronecker delta is the so-called *discrete unit sample function* $\delta[x]$, which takes a value of one when $x = 0$ and a value of zero elsewhere [2]. In mathematical terms:

$$\delta[x] = \begin{cases} 1 & \text{if } x = 0\,; \\ 0 & \text{if } x \neq 0\,. \end{cases}$$

2.5.3 Transpose

Taking the transpose of a matrix $\boldsymbol{M}$ is an operation that switches the row and column indices of $\boldsymbol{M}$ [26]. We denote the transpose of $\boldsymbol{M}$ with $\boldsymbol{M}^\top$ and if the inverse of this transpose is well-defined, it is denoted as $\boldsymbol{M}^{-\top}$.

2.6 Complex Numbers

A *complex number* $z \in \mathbb{C}$ is a number in the complex plane of the form $z = a + bi$ where $a, b \in \mathbb{R}$ and i the imaginary unit satisfying $i^2 = -1$ [42]. Terminology related to complex numbers, such as modulus, real and imaginary part, complex conjugate etc. are listed in Table 4.

A complex number also has a *polar form*: $z = re^{i\phi} = r(\cos\phi + i\sin\phi)$ [42]. Note that r represents the absolute value, i.e., the modulus of z and ϕ the argument of z.

Table 4: Terminology and notation related to complex numbers [42].

Name	**Meaning**	**Notation**
Real axis	Set of real numbers	
Imaginary axis	Set of imaginary numbers	
Imaginary number	Real multiple of i	
Modulus of z	Length r of z	$\|z\| = \sqrt{a^2 + b^2}$
Argument of z	Angle ϕ of z	$\arg(z)$
Real part of z	x coordinate of z	$\mathrm{Re}(z)$
Imaginary part of z	y coordinate of z	$\mathrm{Im}(z)$
Complex conjugate of z	Reflection of z in the real axis	$\overline{z} = a - bi$

Chapter 3

Structure of the Family of Transformations $\mathcal{T}_q$

In order to investigate how we can design q-ary transformations with good propagation properties, we should first define a (set of) transformation(s). In this section, we will present a parameterized multidimensional state array s and define a parameterized family of q-ary transformations called $\mathcal{T}_q$. At the end of this chapter, we explain which conditions should be satisfied by the parameters in the round function to accomplish, for instance, invertibility.

3.1 Structure of the State Array

We will call the state on which $\mathcal{T}_q$ operates s, which is an element of $\mathbb{F}_q^n$. In this research, we will use two representations of this state: see Figure 1.

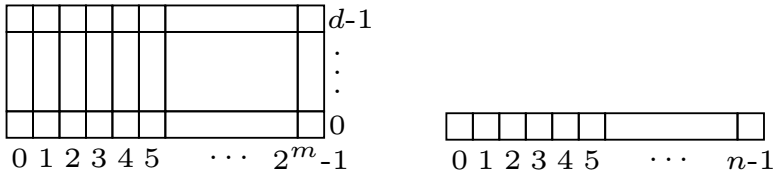


Figure 1: (a) The two-dimensional representation of state s with d rows and 2^m columns where $m \geq 0$. (b) The one-dimensional representation of state s with state width n. Note that $n = d \cdot 2^m$.

In representation (a), the state s is represented as a two-dimensional array. The number of rows is equal to d and the number of columns is a power of two, namely 2^m for $m \geq 0$. The state size n is then equal to $d \cdot 2^m$. The digits in this representation of state s are denoted by $s_{x,y}$ where $0 \leq x < 2^m$ and $0 \leq y < d$.

In representation (b), the state s is depicted as a one-dimensional array. In this case, the digits in the state s are referred to as s_i where $0 \leq i < n$.

There exists a mapping between the two-dimensional indexing and one-dimensional indexing. To go from one-dimensional to two-dimensional, the following formulas should be used:

$$x = i \bmod 2^m\,;$$
$$y = i \bmod d\,.$$

In the reverse direction, this formula can be applied:

$$i = x + ((y - x) \bmod d) \cdot 2^m\,.$$

This last equation results from using the Chinese Remainder Theorem in Garner's form [24]. An example of this state numbering for transformations over $\mathbb{F}_3$ using $d = 3$ and $m = 6$ is shown in Figure 2. Note that this particular indexing is used in the two-dimensional state to simplify the description of the round function: it expresses the working of the shuffle layer well and makes the explanation of it easier.

128	65	2	131	68	5	···	191
64	1	130	67	4	133	···	127
0	129	66	3	132	69	···	63

(a)

0	1	2	3	4	5	···	191

(b)

Figure 2: (a) The two-dimensional representation of state s with $d = 3$ rows and $2^m = 2^6 = 64$ columns. (b) The one-dimensional representation of state s with state width $n = 3 \cdot 64 = 192$.

3.2 Structure of the Round Function

Each round function R_i is built by composing several *step functions* [14]. In this research, the round function R that is iterated in $\mathcal{T}_q$ consists of three step functions, namely the theta (θ) step, the rho (ρ) step and the gamma (γ) step:

$$\mathrm{R} = \gamma \circ \rho \circ \theta\,.$$

Here, θ serves for mixing, ρ for dispersion and γ for non-linearity. A visualization of the round function R for $\mathcal{T}_q$ using $d = 3$ can be seen in Figure 3.

Note that normally a fourth step function is added: a round constant addition [14]. In this thesis, we do not treat such function as it affects our analyses minimally. The addition of this step function is left as future work.

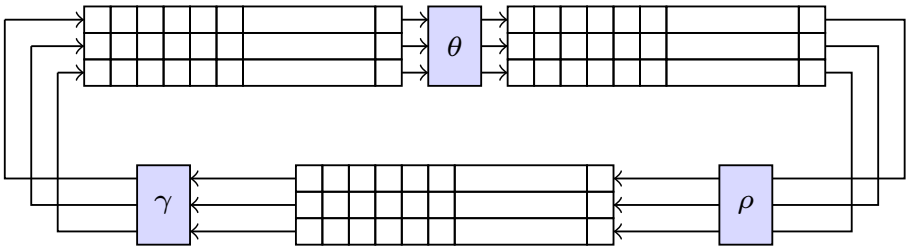


Figure 3: A schematic overview of round function R for $\mathcal{T}_q$ using $d = 3$ in the two-dimensional representation.

3.2.1 Mixing Layer θ

The first step of the round function R is θ and is visualized in Figure 4 for a specific instance. This step updates each state digit by performing additions of state digits. When using the two-dimensional representation, we can formulate this step mathematically as follows:

$$s_{x,y} \leftarrow \sum_{j=0}^{n-1} (c_j \cdot s_{x+j,y+j}) \ .$$

Likewise, using the one-dimensional representation, we get:

$$s_i \leftarrow \sum_{j=0}^{n-1} (c_j \cdot s_{i+j}) \ .$$

Note that, for all step functions, the indices x and y of $s_{x,y}$ are respectively taken modulo 2^m and d and the indices i of s_i are taken modulo n.

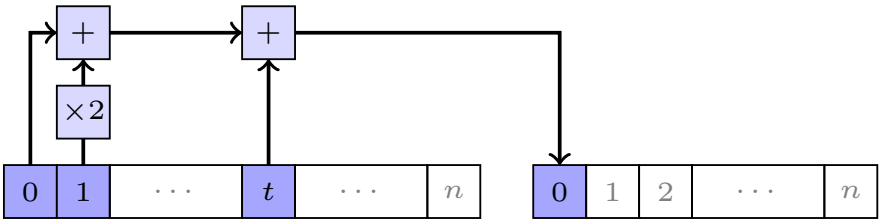


Figure 4: A schematic overview of the θ operation with $c_0 = 1$, $c_1 = 2$, $c_t = 1$ and all other $c_j = 0$ using the one-dimensional representation. Here, it is shown that the state digit s_0 is adapted by θ resulting in the state digit $\theta(s_0)$. In the figure the state digits s_i are denoted as i.

3.2.2 Shuffle Layer ρ

The second step of R is called ρ and is visualized in Figure 5 for $\mathcal{T}_q$ using $d = 3$. ρ makes sure that the state digits are rotated by $r_0, r_1, r_2, \ldots r_{d-1}$ positions to the left depending on their position in the state. Note that

r_0 is fixed to the value zero. For the two-dimensional representation this step function can be explained as follows: row 0 of the state is rotated by 0 positions, row 1 of the state is rotated by r_1 positions, row 2 of the state is rotated by r_2 positions and row $d-1$ of the state is rotated by r_{d-1} positions:

$$s_{x,y} \leftarrow s_{x+r_y,y} \,.$$

When looking at the one-dimensional representation, one can state that each digit with index $i \equiv 0 \pmod{d}$ is rotated by 0 positions, each digit with index $i \equiv 1 \pmod{d}$ is rotated by r_1 positions, each digit with index $i \equiv 2 \pmod{d}$ is rotated by r_2 positions and each digit with index $i \equiv d-1 \pmod{d}$ is rotated by r_{d-1} positions:

$$s_i \leftarrow s_{i+d \cdot r_{i \bmod d}} \,.$$

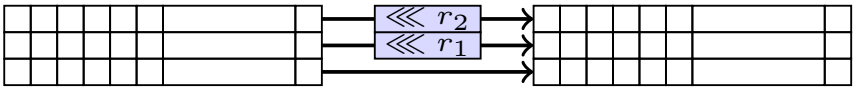


Figure 5: A schematic overview of the ρ operation for $\mathcal{T}_q$ using $d = 3$ in the two-dimensional representation. Here, it is shown that row 0 is not shifted, that row 1 is shifted with r_1 and that row 2 is shifted with r_2.

3.2.3 Non-Linear Layer γ

The last step of the round function R is the non-linear operation γ that takes a parameter g. This step is visualized in Figure 6. In this step, each state digit is updated by adding the square of another state digit. Using the two-dimensional representation, we can express γ mathematically as:

$$s_{x,y} \leftarrow s_{x,y} + (s_{x+g,y+g})^2 \,.$$

When using the one-dimensional representation, we end up with the following:

$$s_i \leftarrow s_i + (s_{i+g})^2 \,.$$

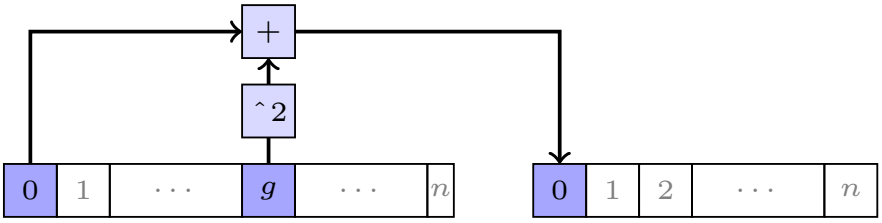


Figure 6: A schematic overview of the γ operation using the one-dimensional representation. Here, it is shown that the state digit s_0 is adapted by γ resulting in the state digit $\gamma(s_0)$. In the figure the state digits s_i are denoted as i.

3.3 Possible Parameters in the Round Function

In the previous sections, the step functions were defined using several parameters. Below we will discuss possible values for all of them.

3.3.1 Parameters c_j in θ

As we saw in the previous section, θ is a function operating on a variable number of state digits. In order to choose proper values for the parameters c_j, one should choose them in such a way that θ is invertible. If θ is not invertible, one can easily find collisions by making use of its kernel [29], which is undesirable. We can determine values for the parameters c_j such that θ is invertible by looking at the associated polynomial of the mixing layer.

The state s can be expressed by using the following polynomial:

$$S(X) = \sum_{i=0}^{n-1} s_i X^i \,.$$

So, instead of looking at the state as a one or two-dimensional array, we consider the state as a polynomial. In this case, addition is thus expressed as the addition of the polynomials. A cyclic shift over offset r_i of the state S is then expressed as the multiplication of S with the polynomial X^{r_i} modulo $X^n - 1$.

The associated polynomial of the mixing layer, which is the polynomial representation of the output state of θ, can be represented as

$$S \leftarrow S \cdot \sum_{j=0}^{n-1} \left(c_j \cdot X^j\right) \bmod X^n - 1 \,.$$

This is because the state S is updated with θ: $s_i \leftarrow \sum_{j=0}^{n-1} (c_j \cdot s_{i+j})$. What we can thus see here is that $s_i = s_{i+0}$ is represented as $X^0 = 1$ in the polynomial, s_{i+1} as $X^1 = X$, s_{i+2} as X^2 etc. The modulo $X^n - 1$ makes sure that the indices j stay within the bounds of 0 and $n - 1$. The inverse of the associated polynomial of θ can then logically be denoted as:

$$S \leftarrow S \cdot \left(\sum_{j=0}^{n-1} \left(c_j \cdot X^j\right)\right)^{-1} \bmod X^n - 1 \,.$$

In order for this inverse to exist, the associated polynomial should be coprime to the polynomial $X^n - 1$. The parameters c_j should thus be chosen such that this property holds.

3.3.2 Parameters r_i in ρ

The next step function, ρ, has d different parameters. The first parameter r_0, is fixed to the zero. Note that the parameter r_0 can be set to 0 because for any r_0 we can add r_0 to all other r_i and get the same state rotated by r_0. This thus means that we get an equivalent state, but need to test less values when looking at the avalanche behavior (as is done in Section 8.5).

The other parameters r_i do not have a restriction like the ones for θ, so these offsets can initially be picked at random as long as they are not equal to zero and $r_i \neq r_j$ for $i \neq j$ holds. If this property would hold, we expect bad propagation properties.

3.3.3 Parameter g in γ

As we saw before, γ operates on two state digits of which one of the two indices is fixed and the other one is determined by the parameter g. We want a parameter value for which it holds that g is coprime to n, i.e., for which it holds that $\gcd(g, n) = 1$. If g is not coprime, then γ performs its operations on separate sections of the state instead of on the full state width n.

To illustrate this, take, for instance, $g = 64$ and $n = 192$ such that $\gcd(64, 192) \neq 1$. Now, assume we want to apply γ to the state s and want to start with calculating s_0. We then have the following equations:

$$\begin{aligned} s_0 &= s_0 + (s_{64})^2 \ ; \\ s_{64} &= s_{64} + (s_{128})^2 \ ; \\ s_{128} &= s_{128} + (s_0)^2 \ . \end{aligned}$$

If this is the case, the number of collisions between states is much higher than when γ would perform its operations on the full state width. So, if we do not want this to happen, g should be coprime to the state size n.

Note that, at this point, γ is not invertible. One could make γ invertible by adding a hole to this step function, i.e., putting a constraint on, for instance, the first digit such that it is left unchanged during this non-linear step. γ would, in that case, look like the following:

$$s_{x,y} \leftarrow \begin{cases} s_{x,y} & \text{if } (x, y) = (0, 0) \, ; \\ s_{x,y} + (s_{x+g,y+g})^2 & \text{if } (x, y) \neq (0, 0) \, . \end{cases}$$

Or equivalently using the one-dimensional representation:

$$s_i \leftarrow \begin{cases} s_i & \text{if } i = 0 \, ; \\ s_i + (s_{i+g})^2 & \text{if } i \neq 0 \, . \end{cases}$$

Chapter 4

Avalanche Behavior

In this chapter we will look into the avalanche behavior of q-ary transformations. The avalanche behavior is about how single digit differences applied at the input to the state propagate through k rounds. Specifically, the avalanche behavior looks at how digits of the intermediate state after k rounds change if one of the input digits changes [49]. It thereby gives an estimate of how vulnerable a cryptographic transformation is against structural distinguishers such as impossible differentials, integral cryptanalysis or truncated differentials [45].

In general, the avalanche behavior is defined for binary transformations and permutations. In this chapter, we will define all definitions and algorithms related to the avalanche behavior for non-binary transformations, i.e., for q-ary transformations. Note that one can deduce the definitions and algorithms for the binary case from the given definitions and algorithms for the q-ary transformations.

4.1 Input and Output Differences

Before we go into the different aspects of the avalanche behavior, we start with explaining what input and output differences are. In order to do this, one should first define inputs and outputs of a q-ary transformation. Let $x \in \mathbb{F}_q^n$ and $x^* \in \mathbb{F}_q^n$ be inputs of the q-ary transformation $f\colon \mathbb{F}_q^n \to \mathbb{F}_q^n$. The *input difference* [14, 19] is then denoted by

$$b = x - x^* \,.$$

Likewise, let $y \in \mathbb{F}_q^n$ and $y^* \in \mathbb{F}_q^n$ be outputs of f for, respectively, the inputs x and x^*. Then, the *output difference* [14, 19] can be written as

$$a = y - y^* = f(x) - f(x^*) \,.$$

4.2 Avalanche Probability Matrix

We start with defining avalanche probability matrices. The definition of the avalanche probability matrices for q-ary transformations can be seen in Definition 1. This definition is generalized from the case considering binary transformations as described in [21, 45].

Definition 1. *Let $f\colon \mathbb{F}_q^n \to \mathbb{F}_q^n$ be a q-ary transformation and b an input difference. The* avalanche probability matrix $\boldsymbol{P}_{f,b} = (p_{i,j})$ for q-ary transformations *is defined as a matrix with dimensions $(n \times q)$ where component (i, j) is the probability that digit i of the output of f has difference j in the output due to the input difference b.*

Note that the avalanche probability matrix $\boldsymbol{P}_{f,b}$ describes $n \cdot q$ probabilities, namely the probability that the difference in digit i equals to $j \in \mathbb{F}_q$ when the input difference is b :

$$p_{i,j,b} = \frac{\left|\left\{x \in \mathbb{F}_q^n\colon f(x)_i - f(x-b)_i = j\right\}\right|}{q^n}\,.$$

The generation of the avalanche probability matrix for q-ary transformations is defined in Algorithm 1. This algorithm generalizes the generation of avalanche probability vectors as given in [21]. Note that Algorithm 1 only gives an approximation of the avalanche probability matrices: the more samples M are used, the more accurate the results.

Algorithm 1 Computation of the avalanche probability matrix $\boldsymbol{P}_{f,b}$.

Parameters: a q-ary transformation $f\colon \mathbb{F}_q^n \to \mathbb{F}_q^n$, an input difference b and a number of samples M.
Output: the avalanche probability matrix $\boldsymbol{P}_{f,b}$.

1: Initialize a two-dimensional array p of probabilities $p_{i,j}$ with the dimension $(n \times q)$ to all zeroes.
2: **for** M uniformly and randomly generated states x **do**
3: Compute $B = f(x) - f(x - b)$
4: **for** all state digit positions i **do**
5: **for** all $j \in \mathbb{F}_q$ **do**
6: **if** $j == B_i$ **then**
7: $p_{i,B_i} = p_{i,B_i} + 1/M$
8: **return** $\boldsymbol{P}_{f,b} = p$

As can be deduced from the algorithm above, the two-dimensional array p now thus represents all the $n \cdot q$ probabilities $p_{i,j,b}$. Algorithm 1 is rather detailed: we use all $n \cdot q$ probabilities whereas we could achieve the same result with only using $n \cdot (q - 1)$ probabilities. The final column (containing

n probabilities) can then be obtained by subtracting the sum of all existing columns from the all-ones vector. Also note that the check in line 6 does not have to be done for each value of j: if the equation is satisfied for a particular j, one can stop the inner loop.

In this research, we will compute the avalanche probability matrix for all input differences that have Hamming weight 1. The digit that attains a non-zero value in the input difference is called the *difference digit*. In general, the avalanche probability matrix is computed for all non-zero values of the difference digit. However, as the difference digit with the value x represents an equivalent difference as the difference digit with the value $-x$, there are only $\frac{q-1}{2}$ different values and thus only $\frac{q-1}{2}$ different avalanche probability matrices to compute.

After having computed all avalanche probability matrices, we compute the three metrics which we will describe in the next sections. We then report on the worst-case performance of the metrics over all the avalanche probability matrices. Note that this is a choice: one could also report on, for instance, the average results or on the divergence between different input differences.

4.3 Avalanche Dependence

After having defined the avalanche probability matrix we turn to the first avalanche metric: the avalanche dependence [21, 45]. This metric is denoted as D_{av} and is defined in Definition 2, which is a generalization of the avalanche dependence for binary transformations as defined in [21, 45].

Definition 2. *Let b be an input difference with Hamming Weight 1 and $p_{i,j,b}$ the probabilities described by $\boldsymbol{P}_{f,b}$. The* avalanche dependence for q-ary transformations *describes the number of output digits that may change due to b. The formula that describes this avalanche dependence is given as follows:*

$$D_{\text{av}}\left(\boldsymbol{P}_{f,b}\right) = n - \sum_{i=0}^{n-1} \delta \left[\sum_{j=1}^{q-1} p_{i,j,b} \right] .$$

An equivalent representation is:

$$D_{\text{av}}\left(\boldsymbol{P}_{f,b}\right) = n - \sum_{i=0}^{n-1} \delta \left[1 - p_{i,0,b} \right] .$$

As we report on the minimal value for D_{av}, we expect that the value for D_{av} over all input differences with Hamming weight 1 of a random transformation is equal to the width of the transformation, i.e., equal to n. This is expected because then all output digits are a function of all n input digits

[45]. At this point, the digits diffuse well throughout the random transformation [45]. Note that, in our experiments, we increase the number of rounds until the value for D_{av} is converged to n.

4.4 Avalanche Weight

The second avalanche metric is the avalanche weight [21, 45]. The avalanche weight is denoted as $\overline{w}_{\text{av}}$ and is defined in Definition 3. This definition is a generalization of the one for binary transformations given in [21, 45].

Definition 3. *Let b be an input difference with Hamming Weight 1 and $p_{i,j,b}$ the probabilities described by $\boldsymbol{P}_{f,b}$. The* avalanche weight for q-ary transformations *describes the expected number of digits that change due to b. The formula that describes this avalanche weight is given as follows:*

$$\overline{w}_{\text{av}}\left(\boldsymbol{P}_{f,b}\right) = \sum_{i=0}^{n-1}\sum_{j=1}^{q-1} p_{i,j,b}\,.$$

An equivalent representation is:

$$\overline{w}_{\text{av}}\left(\boldsymbol{P}_{f,b}\right) = \sum_{i=0}^{n-1}\left(1 - p_{i,0,b}\right)\,.$$

Just like we report on the minimal value for D_{av}, we also report on the minimal value for $\overline{w}_{\text{av}}$. Looking at the value for $\overline{w}_{\text{av}}$ over all input differences with Hamming weight 1 of a random transformation, we expect the value to be equal to $n \cdot \frac{q-1}{q}$. This is because all n output digits can change to one of the other $q-1$ values (if it does not keep its current value). This thus indicates that the output digits change with a probability of $\frac{q-1}{q}$ due to a change in the input difference [49]. In our experiments, we keep increasing the number of rounds until the value for $\overline{w}_{\text{av}}$ is roughly converged to $n \cdot \frac{q-1}{q}$.

4.5 Avalanche Entropy

The third avalanche metric is called the avalanche entropy [21, 45]. This metric is denoted as H_{av}, is defined in Definition 4 and generalizes the avalanche entropy for binary transformations as given in [21, 45].

Definition 4. *Let b be an input difference with Hamming Weight 1 and $p_{i,j,b}$ the probabilities described by $\boldsymbol{P}_{f,b}$. The* avalanche entropy for q-ary transformations *describes the uncertainty about whether output digits change due to b. The formula that describes this avalanche entropy is given as follows:*

$$H_{\text{av}}(\boldsymbol{P}_{f,b}) = \sum_{i=0}^{n-1}\sum_{j=0}^{q-1}\left(-p_{i,j,b}\log_q\left(p_{i,j,b}\right)\right)\,.$$

Note that we choose the log base q instead of log base 2 to make sure that the inner sum can take on a maximum value of 1 if no bias [48] is present, i.e., if no output digit has a higher chance of changing value than other output digits. This thus means that if we again take the minimum value, in this case for H_{av} over all input differences with Hamming weight 1 of a random transformation, we expect the value to be the width of the transformation, i.e., to be equal to n. We expect this because if there is no bias and thus if each of the n output differences occur equally likely, the inner sum will sum op to 1 and the outer sum will make the value of the avalanche entropy equal to n. Note that we increase the number of rounds in the experiments until the value for H_{av} is converged to n.

Chapter 5

Differential Propagation

In order to find the differential propagation properties of a transformation, one will perform differential cryptanalysis. This is "a method which analyzes the effect of particular differences in plaintext pairs on the differences of the resultant ciphertext pairs" [10]. Information about the differential propagation is valuable for estimating how vulnerable a cryptographic transformation is against exploitation of high-probability differentials [10, 12].

This chapter will treat differential cryptanalysis on q-ary transformations. First, we will explain the concepts of (round) differentials, differential probability, differential trails, differential trail cores and trail search. Then, we discuss the differential propagation properties through the non-linear layer of $\mathcal{T}_q$. Lastly, we will provide tools to perform a differential trail search on $\mathcal{T}_q$.

5.1 Differentials

As introduced in the previous chapter, we have the input difference $b = x - x^*$ and the output difference $a = y - y^* = f(x) - f(x^*)$. The (ordered) pair $(b, a) \in \mathbb{F}_q^n \times \mathbb{F}_q^n$ containing the input and output difference is called a differential [14, 19].

5.2 Differential Probability

Using such differential, one can define the differential probability: see Definition 5 [14, 19].

Definition 5. *Let $f \colon \mathbb{F}_q^n \to \mathbb{F}_q^n$ be a q-ary transformation, b an input difference of f and a an output difference of f. The* differential probability (DP) of a differential (b, a) over f *is defined as*

$$\mathrm{DP}_f(b, a) = \frac{\mathrm{N}_f(b, a)}{q^n}$$

with $\mathrm{N}_f(b,a)$ *denoting the number of values* $x \in \mathbb{F}_q^n$ *for which it holds that* $f(x) - f(x-b) = a$.

From this definition we can deduce that the following relation always holds for a fixed input difference b :

$$\sum_a \mathrm{DP}_f(b,a) = 1\,.$$

Another relation between DPs is as follows:

$$\mathrm{DP}_f(b,a) = \mathrm{DP}_f(-b,-a)\,.$$

Having introduced the differential probability, we will define the weight related to this DP as the (restriction) weight. The weight can only be derived from the differential probability if the input difference b and output difference a are compatible, i.e., when $\mathrm{DP}_f(b,a) > 0$ [14, 19]. The definition of this weight is shown in Definition 6 and is derived from the definitions in [14, 19].

Definition 6. *Let* $\mathrm{DP}_f(b,a)$ *be the DP of a differential* (b,a) *over a* q*-ary transformation* f*. The* (restriction) weight of a differential (b,a) over f *that satisfies* $\mathrm{DP}_f(b,a) > 0$ *is defined as*

$$\mathrm{w_r}(b,a) = -\log_q\left(\mathrm{DP}_f(b,a)\right)$$

which can be equivalently written as

$$q^{-\mathrm{w_r}(b,a)} = \mathrm{DP}_f(b,a)\,.$$

A question that arises is for instance: "How can we easily compute $\mathrm{DP}_f(b,a)$ and $\mathrm{w_r}(b,a)$, with f a q-ary transformation containing multiple iterations of the round function R, if the state size n (and thus implicitly q^n) gets too big to exhaustively compute by hand or by computer?". To solve this question, we will investigate the linear and non-linear layer separately.

We start off with the linear layer. The linear layer of $\mathcal{T}_q$ is defined as $\lambda = \rho \circ \theta$. Note that this linear layer λ can be generically expressed as $y = \boldsymbol{M}x$ where $y \in \mathbb{F}_q^n$ denotes the output, $x \in \mathbb{F}_q^n$ the input and $\boldsymbol{M} \in \mathbb{F}_q^{n\times n}$ some matrix (which is a linear transformation) [19]. Now, if we write out the output difference a algebraically, we obtain the result as shown in Equation 1.

$$\begin{aligned} a &= y - y^* \\ &= \lambda(x) - \lambda(x^*) \\ &= \boldsymbol{M}x - \boldsymbol{M}x^* \\ &= \boldsymbol{M}x - \boldsymbol{M}(x-b) \\ &= \boldsymbol{M}x - \boldsymbol{M}x + \boldsymbol{M}b \\ &= \boldsymbol{M}b\,. \end{aligned} \tag{1}$$

This thus means that the linear layer has the following differential probability:

$$\mathrm{DP}_\lambda(b,a) = \begin{cases} 1 & \text{if and only if } a = \boldsymbol{M}b = \lambda(b)\,; \\ 0 & \text{otherwise}\,. \end{cases}$$

Note that the derivation in Equation 1 also holds for affine mappings $y = \boldsymbol{M}x + c$ with some constant vector c.

As we can deduce from the result on $\mathrm{DP}_\lambda(b,a)$ above, we should focus on the differential propagation through the non-linear layer γ. This is a consequence of the fact that for linear layers the DP equals to 1. Normally, one can split the non-linear layer into smaller chunks, called S-boxes [14, 19]. However, our q-ary transformation does not contain any S-boxes, so this is inapplicable in our case. We will investigate the propagation properties of the non-linear layer in Section 5.6.

5.3 Round Differentials

Instead of looking at the differential probability of q-ary transformations, we can also look at the differentials over a round function R_i. Let q_i be the input difference of round R_i and q_{i+1} the output difference of round R_i and, at the same time, the input difference of round R_{i+1}. The (ordered) pair $(q_i, q_{i+1}) \in \mathbb{F}_q^n \times \mathbb{F}_q^n$ containing the input and output difference of round R_i is called a *round differential* [14].

5.4 Differential Trails

It can happen that $\mathrm{N}_{\mathrm{R}_i}(q_i, q_{i+1}) \geq 1$ for all i for a sequence of input and output differences $(q_0, q_1, \ldots, q_k) \in \left(\mathbb{F}_q^n\right)^{k+1}$. In other words, it could be possible that the number of values $x \in \mathbb{F}_q^n$ for which $\mathrm{R}_i(x) - \mathrm{R}_i(x - q_i) = q_{i+1}$ holds (for all $0 \leq i < k$) is bigger or equal to 1 for a sequence of input and output differences $(q_0, q_1, \ldots, q_k) \in \left(\mathbb{F}_q^n\right)^{k+1}$. In this case we denote this sequence as a k-round differential trail [14, 45]: see also Definition 7.

Definition 7. *A* k-round differential trail *is a sequence of* $k+1$ *difference patterns* $Q_k = (q_0, q_1, \ldots, q_k) \in \left(\mathbb{F}_q^n\right)^{k+1}$ *that satisfies* $\mathrm{DP}_{\mathrm{R}_i}(q_i, q_{i+1}) > 0$ *for* $0 \leq i < k$.

Note that if we write $Q \in (q_0, q_k)$, we indicate that Q starts in input difference q_0 and ends in output difference q_k.

If we remove the initial difference q_0 and the final difference q_k, we obtain a *differential trail core* $(q_1, q_2, \ldots, q_{k-1})$ [14]. This trail core defines a collection of differential trails with the same inner differences [14].

Just like a differential, a k-round differential trail has a differential probability. Its differential probability is "equal to the probability that input pair

$(x, x + q_0)$ with x uniformly random will exhibit the sequence of differences through the rounds" [45]. In other words, this DP indicates the probability that a pair with input difference q_0 has difference q_1 after one iteration of the round function, has difference q_2 after two iterations of the round function, etc. [19]. After k rounds the pair should end up with the output difference q_k. Definition 8 describes this concept mathematically as done in [14].

Definition 8. *Let $f\colon \mathbb{F}_q^n \to \mathbb{F}_q^n$ be a q-ary transformation and $Q_k = (q_0, q_1, \ldots, q_k)$ a k-round differential trail. The* differential probability (DP) of a k-round differential trail Q_k *is defined as*

$$\mathrm{DP}_f(Q_k) = \frac{\mathrm{N}_f(Q_k)}{q^n}$$

with $\mathrm{N}_f(Q_k)$ denoting the number of values $x \in \mathbb{F}_q^n$ for which it holds that $\mathrm{R}_i(x) - \mathrm{R}_i(x - q_i) = q_{i+1}$ for all $0 \leq i < k$.

Now, let f represent a q-ary transformation that contains k rounds of round function R. As we know that any ordered pair $(x, x+b)$ follows exactly one differential trail, we also know that the DP of a k-round differential equals to the sum of the differential probabilities of the trails in it [14, 19]:

$$\mathrm{DP}_f(q_0, q_k) = \sum_{Q_k \in (q_0, q_k)} \mathrm{DP}_f(Q_k)\,.$$

As the DP of a differential trail is in general hard to compute, one can use the concept of the expected differential probability to obtain an approximate value: see Definition 9 [14, 19]. Note that this concept is based on a statistical independence assumption [46].

Definition 9. *Let $f\colon \mathbb{F}_q^n \to \mathbb{F}_q^n$ be a q-ary transformation,* R *the round function belonging to f and $Q_k = (q_0, q_1, \ldots, q_k)$ a k-round differential trail. The* expected differential probability (EDP) of a k-round differential trail Q_k *is defined as*

$$\mathrm{EDP}_f(Q_k) = \prod_{i=0}^{k-1} \mathrm{DP}_{\mathrm{R}}\left(q_i, q_{i+1}\right)\,.$$

After having defined the (E)DP of a differential trail, one can also define the weight of it. The weight of a trail is similar to the weight of a differential [14, 19] and is easy to compute exactly: see Definition 10.

Definition 10. *Let $f\colon \mathbb{F}_q^n \to \mathbb{F}_q^n$ be a q-ary transformation, $Q_k = (q_0, q_1, \ldots, q_k)$ a k-round differential trail and $\mathrm{EDP}_f(Q_k)$ the EDP of k-round differential trail Q_k. The* (restriction) weight of a k-round differential trail Q_k *is defined as*

$$\mathrm{w_r}(Q_k) = -\log_q(\mathrm{EDP}_f(Q_k))$$

which can be equivalently written as

$$\mathrm{w_r}(Q_k) = \sum_{i=0}^{k-1} \mathrm{w_r}\left(q_i, q_{i+1}\right) .$$

If the weight is sufficiently small in the size of the state and meets some specific requirements, round differentials are (almost) independent [19] such that

$$\mathrm{DP}_f(Q_k) \approx \mathrm{EDP}_f(Q_k) = q^{-\mathrm{w_r}(Q_k)} .$$

Next to this relation with the expected DP, there are two other important relations we can link to the EDP. First of all, if there is almost no clustering [14], there is often a single dominating trail [19] such that

$$\mathrm{DP}_{\mathrm{R}^k}(q_0, q_k) \approx \mathrm{EDP}_f(Q_k) .$$

Secondly, when counting differential trails, a sequence $(q_0, q_1, q_2, \ldots, q_k)$ is considered as a proper trail only if $\mathrm{EDP}_f(Q_k) > 0$ [14, 19].

5.5 Trail Search

As stated in the beginning of the chapter, high-probability differentials can be exploited [10, 12]. If we encounter trails with low weight and thus with a high DP, we know that there automatically exists a differential with a high DP, which are potential exploitable weaknesses [38]. We thus aim to find the minimum weight, or similarly, the maximum DP, of trails over k rounds. In other words: we aim to find a lower bound on the trail weights over k rounds [30, 38].

As k increases and takes on a value of above 3, this usually already gets quite complex [38]. To still perform this analysis, something called a trail core tree search [38] can be performed. This technique has been applied to various other ciphers such as Keccak-p [39], Xoodoo [21] and Ascon [30] and will be explained in more detail in the next sections.

However, before we do, we should clarify some things. First of all, this trail search looks separately at the linear and non-linear layer of the round function. We split our round function again in the linear layer $\lambda = \rho \circ \theta$ and the non-linear layer γ. Besides this, the k-round differential trail Q_k is written down differently to represent this split: we use a^i to represent the i^{th} input of the linear layer and b^i to represent the i^{th} input of the non-linear layer. For Q_k we then write:

$$Q_k = \left(a^0, b^0, a^1, b^1, \ldots, b^{k-1}, a^k\right) .$$

5.5.1 Trail Extension in the Forward Direction

The first concept we need to understand before being able to do trail search is trail extension in the forward direction: extending a k-round trail $Q_k = \left(a^0, b^0, a^1, b^1, \ldots, b^{k-1}, a^k\right)$ by one round [38]. The result of this extension is a set of $k+1$-round trails $Q_{k+1} = \left(a^0, b^0, a^1, b^1, \ldots, b^{k-1}, a^k, b^k, a^{k+1}\right)$. As can be seen, for trail extension in the forward direction one should first compute $b^k = \lambda(a^k)$ and then build all compatible output differences over γ, i.e., build all a^{k+1} [38].

If we now want to say something about the weight of this newly created trail Q_{k+1}, one should keep in mind that the weight of the shorter original trail, Q_k, is known. This weight is namely equal to $\mathrm{w_r}(Q_k)$. The only thing to determine is the weight of the two newly added differences, namely $\mathrm{w_r}\left(b^k, a^{k+1}\right)$, such that we can define

$$\mathrm{w_r}(Q_{k+1}) = \mathrm{w_r}(Q_k) + \mathrm{w_r}\left(b^k, a^{k+1}\right)$$

as the weight of the extended trail. To find a trail with a DP as high as possible, we want to compute what the minimum possible weight is for a given transformation. This can be achieved by computing the minimum direct weight [38]: see Definition 11.

Definition 11. *Let* $\mathrm{w_r}\left(b^k, a^{k+1}\right)$ *be the weight of the differential* $\left(b^k, a^{k+1}\right)$. *The* minimum direct weight *is defined as*

$$\mathrm{w^{dir}}\left(b^k\right) := \min_{a^{k+1}} \mathrm{w_r}\left(b^k, a^{k+1}\right) .$$

5.5.2 Trail Extension in the Backward Direction

The second important concept is related to the previous notion: if one can extend a trail in the forward direction, one should also be able to extend a trail in the backward direction. More concretely, trail extension in the backward direction is about extending a k-round trail $Q_k = \left(a^1, b^1, \ldots, b^{k-1}, a^k, b^k, a^{k+1}\right)$ by one round [38]. The result is then the set of $k+1$-round trails $Q_{k+1} = \left(a^0, b^0, a^1, b^1, \ldots, b^{k-1}, a^k, b^k, a^{k+1}\right)$. To obtain this set, one should thus first build all compatible input differences over γ, i.e., build all b^0 and then compute $a^0 = \lambda^{-1}(b^0)$ [38].

If we want to say something about the weight, we can again define it as the sum of the weight of the k-round trail and the newly added weight: in this case of the differential $\left(b^0, a^1\right)$. Note that it is wrong to consider the differential of the two computed values – $\left(a^0, b^0\right)$ – as for the weight we need a differential over the non-linear layer γ instead of over the linear layer λ.

From this, we can thus conclude that the weight of the $k+1$-round differential trail is defined as

$$\mathrm{w_r}(Q_{k+1}) = \mathrm{w_r}\left(b^0, a^1\right) + \mathrm{w_r}(Q_k) .$$

To find a trail with a DP as high as possible, we should know what the minimum possible weight for this transformation is. This is described by the so-called minimum reverse weight [38] as shown in Definition 12.

Definition 12. *Let* $\mathrm{w_r}\left(b^0, a^1\right)$ *be the weight of the differential* $\left(b^0, a^1\right)$*. The* minimum reverse weight *is defined as*

$$\mathrm{w^{rev}}\left(a^1\right) := \min_{b^0} \mathrm{w_r}\left(b^0, a^1\right) .$$

5.5.3 Trail Cores

The last concept to understand before we can do a trail search is the notion of trail cores as introduced in Section 5.4. Using the notation of input and output differences on the level of the non-linear and linear layer, one can write a k-round differential trail core as $\left(a^1, b^1, \ldots, b^{k-1}\right)$ [14, 38]. If we combine these trail cores with the notions of minimum direct and minimum reverse weight, one can lower bound the weight of a k-round trail core by using the following formula [38]:

$$\mathrm{w_r}\left(Q_k\right) = \mathrm{w^{rev}}\left(a^1\right) + \sum_{i=2}^{k-1} \mathrm{w_r}\left(b^{i-1}, a^i\right) + \mathrm{w^{dir}}\left(b^{k-1}\right) .$$

5.5.4 Search Strategy Used in Trail Search

To find the lower bound on the trail weight, one could build all k-round trail cores up to some target weight T_k. If no trail core is found, then T_k is a non-tight bound on the weight of all k-round trails [38]. If a trail core is found, the minimum weight found defines a tight bound on the weight of all k-round trails [38].

However, building all k-round trail cores up to some target weight T_k is very inefficient and possibly even infeasible. To solve this issue, one can start with a k'-round trail core with $k' < k$ and extend this trail core using the extension methods explained in previous sections [38].

To give a small example of this technique [30, 38], we assume that our goal is to lower bound the weight of all 4-round trail cores using target weight T_4. A 4-round trail core Q_4 has the following weight:

$$\mathrm{w_r}(Q_4) = \mathrm{w^{rev}}(a_1) + \mathrm{w_r}\left(b^1, a^2\right) + \mathrm{w_r}\left(b^2, a^3\right) + \mathrm{w^{dir}}(b_3) .$$

As we know that $\mathrm{w_r}(Q_4) \leq T_4$, this implies that

$$\mathrm{w^{rev}}(a_1) + \mathrm{w_r}\left(b^1, a^2\right) \leq \frac{T_4}{2}$$

or

$$\mathrm{w_r}\left(b^2, a^3\right) + \mathrm{w^{dir}}(b_3) \leq \frac{T_4}{2} .$$

This thus indicates that we can build any 4-round trail core Q_4 with $\mathrm{w_r}(Q_4) \leq T_4$ by building all 2-round trail cores Q_2 with $\mathrm{w_r}(Q_2) \leq T_2 = \frac{T_4}{2}$ and after this extending them by 2 rounds in the forward and backward direction [38].

To generate a 2-round trail core, denoted by $Q_2 = (a^1, b^1)$, one should first note that Q_2 is fully determined by a^1 [38]. We can namely write $b^1 = \lambda(a^1)$ resulting in the trail core notation $Q_2 = (a^1, \lambda(a^1))$. From this, it follows that the weight of this trail core is also fully determined by a^1 [38]:

$$\mathrm{w_r}(Q_2) = \mathrm{w}^{\mathrm{rev}}(a_1) + \mathrm{w}^{\mathrm{dir}}(\lambda(a^1))\,.$$

The last thing we need to do in order to bound $\mathrm{w_r}(Q_2)$ by target weight T_2, is to build all possibilities for a^1 [38]. This can be done using a so-called tree-traversal approach as explained in [30].

5.6 Differential Propagation Properties of γ in $\mathcal{T}_q$

Before we look at the full non-linear layer γ, we will first investigate the DP of the squaring function $s(x) = x^2$. Let b be the input difference of input pair (x, x^*). The corresponding output difference a can then be written as:

$$\begin{aligned} a &= y - y^* \\ &= x^2 - (x^*)^2 \\ &= x^2 - (x-b)^2 \\ &= x^2 - x^2 - 2xb + b^2 \\ &= b^2 - 2xb\,. \end{aligned}$$

As we can see, the result is a linear equation. This equation shows us that for any output difference a there is only one input pair (x, x^*) with $x = (b^2 - a)(2b)^{-1}$. From this, it follows that the set of output differences a compatible over the squaring function $s(x) = x^2$ with a non-zero input difference b coincides with $\mathbb{F}_q$. Therefore, $\mathrm{DP}_s(b, a) = \frac{1}{q}$.

Now, the output difference of the non-linear layer γ can also be written out mathematically. We rewrite the output difference a with respect to digit i of the state in terms of input difference b and input x in Equation 2. Note that in this equation the input difference b is known, and thus that we can consider the differences b_i and b_{i+g} as constants.

$$
\begin{aligned}
a_i &= y_i - y_i^* \\
&= \gamma(x_i) - \gamma(x_i^*) \\
&= \left(x_i + (x_{i+g})^2\right) - \left(x_i^* + \left(x_{i+g}^*\right)^2\right) \\
&= x_i - x_i^* + (x_{i+g})^2 - \left(x_{i+g}^*\right)^2 \\
&= b_i + (x_{i+g})^2 - \left(x_{i+g}^*\right)^2 \\
&= b_i + \left(x_{i+g} - x_{i+g}^*\right)\left(x_{i+g} + x_{i+g}^*\right) \\
&= b_i + b_{i+g}\left(x_{i+g} + x_{i+g}^*\right) \qquad (2) \\
&= b_i + b_{i+g}\left(x_{i+g} - x_{i+g}^* + 2x_{i+g}^*\right) \\
&= b_i + b_{i+g}\left(b_{i+g} + 2x_{i+g}^*\right) \\
&= b_i + (b_{i+g})^2 + b_{i+g} \cdot 2x_{i+g}^* \\
&= b_i + (b_{i+g})^2 + b_{i+g} \cdot 2(x_{i+g} - b_{i+g}) \\
&= b_i + (b_{i+g})^2 + b_{i+g} \cdot 2x_{i+g} - 2\,(b_{i+g})^2 \\
&= b_i - (b_{i+g})^2 + b_{i+g} \cdot 2x_{i+g}\,.
\end{aligned}
$$

The result of Equation 2 thus means that we can consider the expression as a linear equation in input digit $2x_{i+g}$. We can consider two cases regarding b_{i+g} :

1. If $b_{i+g} = 0$, the digit value $2x_{i+g}$ can take on any value, i.e., $2x_{i+g} \in \mathbb{F}_q$, as long as $a_i = b_i$. This is the case as we know that $\gcd(g, 2) = 1$ holds: we namely know that 2 is invertible. Because 2 is invertible, the map $z \mapsto 2z$ becomes bijective. So, if z can take on any value, then so can $2z$.
2. If $b_{i+g} \neq 0$, the digit value $2x_{i+g}$ can only have one value as determined by the constants in the following equation:

$$
2x_{i+g} = \left(a_i - b_i + (b_{i+g})^2\right)(b_{i+g})^{-1}\,.
$$

From these two cases, we can conclude that Theorem 1 holds for the DP of a differential (b, a) over the non-linear layer γ.

Theorem 1. *Let $\gamma\colon \mathbb{F}_q^n \to \mathbb{F}_q^n$ be the non-linear layer of $\mathcal{T}_q$, b an input difference of γ and a an output difference of γ. The differential probability (DP) of a differential (b, a) over γ is equal to*

$$
\mathrm{DP}_\gamma(b, a) = q^{-\mathrm{HW}(b)}\,.
$$

Proof of Theorem 1. The case distinction has shown that $2x_{i+g}$ is fixed if $b_{i+g} \neq 0$, but that it can take on any value if $b_{i+g} = 0$. For the DP, it is important to know how many times an input can take on any value as we

can calculate the DP using this knowledge. The DP should namely be equal to the number of possible (unfixed) values, divided by all possible states:

$$\begin{aligned}\mathrm{DP}_\gamma(b,a) &= \frac{q^{\text{number of zero entries of } b}}{q^n} \\ &= \frac{q^{n-\mathrm{HW}(b)}}{q^n} \\ &= \frac{q^n \cdot q^{-\mathrm{HW}(b)}}{q^n} \\ &= q^{-\mathrm{HW}(b)}\,.\end{aligned}$$

Here, n represents the size of the state. □

As we can see in the derivation above, this relation does not only hold for q, but holds for any integer N as long as N is coprime to 2, i.e., as long as N is an odd number. Note that, at this point, we are not talking about a finite field anymore. The structure in which this will work is in a structure where the element 2 has an inverse, such as the ring $(\mathbb{Z}/N\mathbb{Z}, +, \times)$.

As we know that $\mathrm{DP}_\gamma(b,a) = q^{-\mathrm{w_r}(b,a)}$ should hold, we can answer the question posed before on how to compute $\mathrm{DP}_f(b,a)$ and $\mathrm{w_r}(b,a)$ easily. We can namely state that the weight of a differential is equal to the Hamming weight of the input difference:

$$\mathrm{w_r}(b,a) = \mathrm{HW}(b)\,. \tag{3}$$

5.6.1 Tools for Differential Trail Search on $\mathcal{T}_q$

We will provide tools that are useful to perform differential trail search on $\mathcal{T}_q$. However, the complete trail search is left as future work.

Before we provide the components for a trail search on $\mathcal{T}_q$, we note that we fixed the parameter g in the non-linear layer γ to $g = 1$. We did this to better see relations between the output and input differences of γ. Note that the results for $g = 1$ can logically be converted to any parameter value g: this is done using multiplicative shuffles as the only thing to be changed is the order of the state coordinates. An example of this is as follows. When using $g = 1$ in γ we can represent a 5-digit state by $s_0s_1s_2s_3s_4$. If we now want to convert this to a non-linear layer using $g = 2$, we reorder the state as $s_0s_2s_4s_1s_3$ to obtain similar results. The algorithms in this chapter are for simplicity thus defined with the parameter $g = 1$.

Compatible Output Differences Through γ

The compatible output differences $a \in \mathbb{F}_q^n$ over γ form an affine space for a certain input difference $b \in \mathbb{F}_q^n$ [18, 19, 38]. This can be shown if we consider

Equation 2. Replacing g with 1 in this equation, we can immediately deduce that our offset is $b_i - (b_{i+1})^2$. In matrix notation, we can write the result of Equation 2 as [18]:

$$\begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ \vdots \\ a_{n-2} \\ a_{n-1} \end{bmatrix} = b_i - (b_{i+1})^2 + \begin{bmatrix} 0 & b_1 & 0 & 0 & \cdots & 0 & 0 \\ 0 & 0 & b_2 & 0 & \cdots & 0 & 0 \\ 0 & 0 & 0 & b_3 & \cdots & 0 & 0 \\ \cdots & & & & & & \\ 0 & 0 & 0 & 0 & \cdots & 0 & b_{n-1} \\ b_0 & 0 & 0 & 0 & \cdots & 0 & 0 \end{bmatrix} \begin{bmatrix} 2x_0 \\ 2x_1 \\ 2x_2 \\ \vdots \\ 2x_{n-2} \\ 2x_{n-1} \end{bmatrix}.$$

The independent columns of this matrix form a basis [18]. Note that they are all already independent since each column has only one non-zero value that does not appear in any other column. The basis is thus given by column vectors $u_k = b_{k-1}e_{k-1}$, which are vectors with all zeros except b_{k-1} in position $k-1$ [18].

Having an affine space formed by the compatible output differences means one can find all compatible output differences by building an offset and a basis belonging to that affine space [5]. How to build this offset and basis is shown in Algorithm 2.

Algorithm 2 Generation of the offset and basis for the affine space that defines all compatible output differences of the input difference $b \in \mathbb{F}_q^n$.

Parameters: an input difference $b \in \mathbb{F}_q^n$.
Output: the offset P and basis V for the compatible output differences of input difference $b \in \mathbb{F}_q^n$.

1: Initialize an empty set V to hold all basis vectors v_i.
2: **for** state coordinate i **do**
3: $\quad P_i = b_i - (b_{i+1})^2$
4: $\quad$ **if** $b_i \neq 0$ **then**
5: $\quad\quad$ Add $b_i e_i$ to V.
6: **return** (P, V)

Hamming Weight as the Minimum Direct Weight

If we look at Definition 11, we see that the minimum direct weight takes the minimum restriction weight over all differentials (b^k, a^{k+1}). However, as was shown in Equation 3, the restriction weight of a differential (b, a) is equal to the Hamming weight of input difference b: $\mathrm{w_r}(b, a) = \mathrm{HW}(b)$. As a result, we essentially do not have a minimum direct weight: it is equal to the Hamming weight of the input difference.

Compatible Input Differences Through γ

The input differences compatible with a given output difference do not form an affine space. To find relations to determine the compatible input differences without an exhaustive search of all q^n possibilities, we make a toy example over $\mathbb{F}_3$ with state size $n \leq 8$.

In this small example, all possible input differences are generated. From these input differences, all output differences over γ are computed in a brute-force manner. After having generated all compatible output differences for each input difference, we make a list of the reversed structure: we make a list of all compatible input differences for a certain output difference. From this list, we could reveal a structure on how to compute all compatible input differences belonging to a given output difference of any instance of $\mathcal{T}_3$, but also of $\mathcal{T}_q$.

The method to find all compatible input differences of output difference a consists of two steps. First, the compatible input activity patterns belonging to a are determined. How to do this is described in Algorithm 3.

As can be seen in Algorithm 3, a tree traversal approach is taken in order to obtain all compatible input activity patterns B. The first thing that is done is getting the activity pattern A of output difference a. Then, the input activity pattern B is initialized to all unspecified values. After this, it is specified whether B_{n-1} is active or passive. Following on this, for all other digits B_{n-2} to B_0, we determined whether the digits are active or passive based on these two rules:

1. If $B_i = 0$ and $A_{i-1} = 0$, then $B_{i-1} = 0$;
2. If $B_i = 0$ and $A_{i-1} \neq 0$, then $B_{i-1} \neq 0$.

Proof of statement 1. We prove statement 1 with a proof by contradiction. Assume that $B_i = 0$, $A_{i-1} = 0$ and $B_{i-1} \neq 0$. As $B_i = 0$ we know that $b_i = x_i - x_i^* = 0$, which yields $x_i = x_i^*$. Likewise, as $B_{i-1} \neq 0$, we know that $b_{i-1} = x_{i-1} - x_{i-1}^* \neq 0$, which yields $x_{i-1} \neq x_{i-1}^*$. Because an application of γ results in $y_{i-1} = x_{i-1} + (x_i)^2$ and $y_{i-1}^* = x_{i-1}^* + (x_i^*)^2$, we know that $y_{i-1} \neq y_{i-1}^*$. This is because in both equations the same value for the square is added, but the first term differs. Since $y_{i-1} \neq y_{i-1}^*$, it follows that $a_{i-1} = y_{i-1} - y_{i-1}^* \neq 0$ and thus that $A_{i-1} \neq 0$. But we assumed that $A_{i-1} = 0$: a contradiction. □

Proof of statement 2. We prove statement 2 with a proof by contradiction. Assume that $B_i = 0$, $A_{i-1} \neq 0$ and $B_{i-1} = 0$. As $B_i = 0$ we know that $b_i = x_i - x_i^* = 0$, which yields $x_i = x_i^*$. Likewise, as $B_{i-1} = 0$, we know that $b_{i-1} = x_{i-1} - x_{i-1}^* = 0$, which yields $x_{i-1} = x_{i-1}^*$. Because an application of γ results in $y_{i-1} = x_{i-1} + (x_i)^2$ and $y_{i-1}^* = x_{i-1}^* + (x_i^*)^2$, we know that $y_{i-1} = y_{i-1}^*$. This is because both equations essentially perform the same addition of a square. Since $y_{i-1} = y_{i-1}^*$, it follows that $a_{i-1} = y_{i-1} - y_{i-1}^* = 0$

Algorithm 3 Generation of all compatible input activity patterns of the output difference $a \in \mathbb{F}_q^n$.

Parameters: an output difference $a \in \mathbb{F}_q^n$ and bound for the weight W.
Output: list L containing the compatible input activity patterns of output difference $a \in \mathbb{F}_q^n$.

Build the activity pattern A of output difference a.
Initialize an empty list L to hold compatible input activity patterns.
Initialize the activity pattern B of input difference b to all $*$.
$B_{n-1} = 1$
`buildB`$(n-1,\ B,\ A,\ W)$
$B_{n-1} = 0$
`buildB`$(n-1,\ B,\ A,\ W)$

where `buildB`(i, B, A, W), with i the remaining unspecified digits of B, B the activity pattern of input difference b, A the activity pattern of output difference a and W the bound for the weight, is defined as:
    if $\mathrm{HW}(B) > W$ **then return**
    if $i = 0$ **then**
        if $B_{n-1} = 1$ **or** $A_0 = B_0$ **then**
            Add B to L.
        return
    $B' = B$
    if $B_i = 1$ **or** $A_{i-1} = 1$ **then**
        $B'_{i-1} = 1$
        `buildB`$(i-1, B', A, W)$
    if $B_i = 1$ **or** $A_{i-1} = 0$ **then**
        $B'_{i-1} = 0$
        `buildB`$(i-1, B', A, W)$
    return

and thus that $A_{i-1} = 0$. But we assumed that $A_{i-1} \neq 0$: a contradiction. $\square$

Note that these rules are not used explicitly in Algorithm 3. We use the following inferences from these statements:

1. If $B_i = 1$, the rules give no restriction on the value for B_{i-1}. This thus means that we should investigate both possibilities: B_{i-1} can be passive ($B_{i-1} = 0$) but can also be active ($B_{i-1} = 1$);
2. If $A_{i-1} = 1$ and $B_i = 0$, we trigger the second rule: this means that B_{i-1} should be active ($B_{i-1} = 1$);
3. If $A_{i-1} = 0$ and $B_i = 0$, we trigger the first rule: this means that B_{i-1} should be passive ($B_{i-1} = 0$);

The second step is to deduce the compatible input differences from the compatible input activity patterns resulting from Algorithm 3. This is done as follows. Given an input activity pattern B, one leaves the passive coordinates fixed to 0, as passive coordinates represent zero values. Then, we should change an active coordinate B_i to a_i if $B_{i+1} = 0$. This is because if $B_{i+1} = 0$ and $B_i \neq 0$, we get that $x_{i+1} = x^*_{i+1}$ and $x_i \neq x^*_i$, which yields $a_i = y_i - y^*_i = x_i + (x_{i+1})^2 - \left(x^*_i + \left(x^*_{i+1}\right)^2\right) = x_i - x^*_i = b_i$. Following the same reasoning, the last step is changing all other active coordinates B_i to any non-zero value if $B_{i+1} \neq 0$.

Minimum Reverse Weight

As we can generate all compatible input differences belonging to a certain output difference, this means that an algorithm that deduces the minimum reverse weight can also be formulated: see Algorithm 4.

As we can see in Algorithm 4, the minimum reverse weight is a bit more complicated than the minimum direct weight. The minimum reverse weight is defined as a sum over the length l of each 1-run. We namely add:

- $\frac{l}{2}$ if l is an even number;
- $\frac{l+1}{2}$ if l is an odd number.

We can define the minimum reverse weight like this because of the rules defined above: if $B_i = 0$ and $A_{i-1} = 0$, then $B_{i-1} = 0$ and if $B_i = 0$ and $A_{i-1} \neq 0$, then $B_{i-1} \neq 0$. The first rule basically says that if we have a zero on position $i - 1$ in the output difference a, the input difference b can never have a non-zero value on position $i - 1$ and a zero on position i. Similarly, the second rule states that if we have a non-zero digit on position $i - 1$ in output difference a, the input difference b can never have zero values on both positions i and $i - 1$.

When we look at a 1-run of even length, we notice that the minimal way to comply with these rules is to have alternating zero and non-zero values in

Algorithm 4 Computation of the minimum reverse weight of the output difference $a \in \mathbb{F}_q^n$.

Parameters: an output difference $a \in \mathbb{F}_q^n$.
Output: the minimum reverse weight of $a \in \mathbb{F}_q^n$.

```
1: Make a list L to hold all 1-runs of output difference a.
2: if L ≠ ∅ do
3:     Initialize the minimum reverse weight w to 0.
4:     for r ∈ L do
5:         l = length(r)
6:         if l%2 = 0 do
7:             w = w + l/2
8:         else
9:             w = w + (l+1)/2
10:    return w
11: else
12:    return 0
```

the input difference. If a zero value on position i would change to a non-zero value, it still follows the rules, but we incremented the weight by one (so it is not minimal anymore). The other way around, if a non-zero value on position i changes to a zero value, we have zero values on both positions i and $i-1$, which is not possible. This thus indicates that if the length of a 1-run is even, the minimum reverse weight of this part is equal to $\frac{l}{2}$.

The same holds a 1-run of odd length. However, one should note that the alternating sequence of zero and non-zero values should always start with a non-zero value. If this is not the case, the first rule will not be satisfied. Therefore, if the length of a 1-run is odd, the minimum reverse weight of this part is equal to $\frac{l+1}{2}$.

Note that if an output difference is the all-zero state, no runs are found. In this case, the compatible input with the lowest weight would be the all-zero state again, which is why, in Algorithm 4, 0 is returned in case no runs are found.

5.6.2 Non-invertibility of γ

A non-zero input difference b can lead to a zero-output difference a (and thereby causing a collision) if 0 is its offset in the affine space. This can only happen if for all positions it holds that both the positions b_i and b_{i+1} are active, i.e., if the input difference b has no coordinates equal to zero. There are $(q-1)^n$ of such input differences, for which $\mathrm{DP}_\gamma(b, 0) = q^{-n}$.

We can quantify the non-invertibility of γ by looking at the collision probability. We define the collision probability of a mapping as the proba-

bility that we randomly choose two different inputs and have their outputs collide. We know that a random transformation $f\colon \mathbb{F}_q^n \to \mathbb{F}_q^n$ has collision probability q^{-n} since this is the probability that two chosen inputs end up in the same image. For the non-linear layer γ, the collision probability is equal to the number of colliding pairs divided by the total number of pairs. In mathematical notation:

$$\frac{(q-1)^n}{\binom{q^n}{2}} = \frac{2(q-1)^n}{q^n(q^n-1)} \approx \frac{2(q-1)^n}{q^{2n}}\,.$$

Therefore, if we look at the differential propagation, the non-invertibility of γ is no problem.

Chapter 6

Linear Propagation

Linear cryptanalysis of primitives that operate on arrays of elements of $\mathbb{F}_2$ is "a known plaintext attack in which the attacker studies probabilistic linear relations (called linear approximations) between parity bits of the plaintext, the ciphertext, and the secret key" [11]. Information about the linear propagation is valuable for estimating how vulnerable a cryptographic transformation is against exploitation of high correlations (in absolute sense) between linear combinations of input digits and linear combinations of output digits [14, 36].

This chapter will discuss the generalization of binary linear cryptanalysis to transformations that are defined over $\mathbb{F}_p$. The generalization for q-ary transformations (by making use of the trace function [23, 34]) is left as future work. We will first explain the concepts of correlation and linear approximations. After this, we discuss the linear propagation properties through the linear and non-linear layer of $\mathcal{T}_q$. Then, the concepts of linear potential, round linear approximations, linear trails, linear trail cores and trail search will be elaborated upon. Lastly, we will provide tools to perform a linear trail search on $\mathcal{T}_q$.

6.1 Correlation

Instead of considering all q-ary transformations, we will look at transformations from $\mathbb{F}_p^n$ to $\mathbb{F}_p^n$ for which thus $q = p^l = p^1 = p$ with p a prime. Having such transformations, we define new functions $f\colon \mathbb{F}_p^n \to \mathbb{F}_p$. The correlation between two such functions $f, g\colon \mathbb{F}_p^n \to \mathbb{F}_p$ will not be defined in the range of -1 to $+1$ as was the case for primitives that operate on arrays of elements of $\mathbb{F}_2$. If we generalize the setting of binary linear cryptanalysis to functions over $\mathbb{F}_p$, correlations become complex numbers on the unit circle. These complex numbers can be seen as vectors in $\mathbb{R}^2$ and thus, the correlation is related to the scaling factor in the projected vector [42].

Specifically, we can define the functions $\hat{f}\colon \mathbb{F}_p^n \to S$ where $S = \{z \in \mathbb{C}\colon |z| = 1\}$ is the unit circle in the complex plane [25, 35]. The functions $\hat{f}$ make use of the p^{th} root of unity, say, $\omega = e^{\frac{2\pi i}{p}}$ and can be expressed as:

$$\hat{f}(x) = \omega^{f(x)}\,.$$

Note that the image of $\hat{f}$ is not equal to all complex numbers, but only equal to the p p^{th} roots of unity lying on the unit circle S.

The set of functions $\mathbb{F}_p^n \to \mathbb{C}$ is an inner product space [29], which means that one can define an inner product between two such functions $\hat{f}$ and $\hat{g}$. The inner product of $\hat{f}$ and $\hat{g}$ is denoted by $\langle \hat{f}, \hat{g} \rangle$ and is defined as:

$$\langle \hat{f}, \hat{g} \rangle = \sum_{x \in \mathbb{F}_p^n} \hat{f}(x)\overline{\hat{g}(x)}$$

where $\overline{z}$ is the complex conjugate as introduced in Section 2.6. Having an inner product, it defines the norm, which is denoted by $||\hat{f}||$ and expressed as:

$$||\hat{f}|| = \sqrt{\langle \hat{f}, \hat{f} \rangle}\,.$$

The distance between two functions $\hat{f}$ and $\hat{g}$ can be calculated as $||\hat{f} - \hat{g}||$. Here, $\hat{f} - \hat{g}$ is given by $(\hat{f} - \hat{g})(x) = \hat{f}(x) - \hat{g}(x)$.

Note that the normalized inner product will function as our correlation. Therefore, we will now define the correlation between two functions $f, g\colon \mathbb{F}_p^n \to \mathbb{F}_p$ as described in Definition 13.

Definition 13. *Let $f, g\colon \mathbb{F}_p^n \to \mathbb{F}_p$ be functions over $\mathbb{F}_p$ from n digits to 1 digit. The* (complex) correlation between the functions f and g *is defined as*

$$\mathrm{C}(f, g) = \frac{1}{p^n} \cdot \sum_{x \in \mathbb{F}_p^n} \hat{f}(x)\overline{\hat{g}(x)}\,.$$

To make the correlation a bit more concrete, one should note that an inner product space is automatically a vector space (that has an inner product), which means that it has many bases $b \in B$. This B thus contains linear (basis) functions, which are orthogonal to each other [5]. We know that if B is the set of orthogonal basis vectors b of a vector space, one can express each vector v of that vector space as follows [29]:

$$v = \sum_{b \in B} \frac{\langle b, v \rangle}{||b||^2} \cdot b\,.$$

The coefficients of this expression are equal to the correlations, i.e., the correlation between v and a basis vector b is equal to $\frac{\langle b, v \rangle}{||b||^2}$. As one can see, these correlations are thus scaled inner products.

6.2 Linear Approximations

After giving an explanation on correlations, we will define linear approximations and their correlations. Let $f\colon \mathbb{F}_p^n \to \mathbb{F}_p^n$ be a transformation operating on n digits in $\mathbb{F}_p$. Take $u \in \mathbb{F}_p^n$ and $v \in \mathbb{F}_p^n$. The (ordered) pair $(u, v) \in \mathbb{F}_p^n \times \mathbb{F}_p^n$ is called a linear approximation, with u the input mask and v the output mask [14].

We can perform an operation on the transformation $f\colon \mathbb{F}_p^n \to \mathbb{F}_p^n$ to obtain functions $f_v\colon \mathbb{F}_p^n \to \mathbb{F}_p$. This can be done by choosing an output mask $v \in \mathbb{F}_p^n$ and after this linearly combining coordinates. Therefore, the functions f_v with input $x \in \mathbb{F}_p^n$ are defined as:

$$f_v(x) = v^\top f(x)\,.$$

The correlation of a linear approximation can then be defined by the correlation between the functions $u^\top x$ and $v^\top f(x)$ [25] as follows:

$$C_f(u, v) = \frac{1}{p^n} \cdot \sum_{x \in \mathbb{F}_p^n} \omega^{u^\top x - v^\top f(x)}\,. \tag{4}$$

Note that $u^\top x$ defines a linear function of the input digits and that $v^\top f(x)$ defines a linear function of the output digits for $x \in \mathbb{F}_p^n$ [18]. If $v^\top f(x)$ is viewed as a function of the input digits, it is not linear in case f is non-linear.

As there are attacks known which use correlations with high absolute value, the goal is to make sure that there are no such correlations between linear functions of the input and linear functions of the output of transformations [14, 36].

6.2.1 Linear Mask Propagation Through a Linear Layer

Recall that a linear layer λ can be expressed as $y = \boldsymbol{M}x$ where $y \in \mathbb{F}_p^n$ denotes the output, $x \in \mathbb{F}_p^n$ the input and $\boldsymbol{M} \in \mathbb{F}_p^{n \times n}$ some matrix (which is a linear transformation) [19]. Taking the output mask $v \in \mathbb{F}_p^n$ one can deduce the following relation regarding the linear layer [18]:

$$\begin{aligned} v^\top y &= v^\top \boldsymbol{M} x \\ &= \left(\boldsymbol{M}^\top v\right)^\top x\,. \end{aligned}$$

What we can see in this relation is that if $u = \boldsymbol{M}^\top v$, we get the equation $v^\top \lambda(x) = u^\top x$ when denoting $y = \lambda(x)$. But, if these two expressions are equal, we see in Equation 4 that $\mathrm{C}_\lambda(u, v)$ equals to 1.

A linear layer could potentially also include constant addition, which influences the correlation. Let $c\colon \mathbb{F}_p \to \mathbb{F}_p$ be given by $c(x) = x + k$. Again,

taking the output mask $v \in \mathbb{F}_p$ one can deduce [18]:

$$\begin{aligned} v^\top c(x) &= v^\top (x + k) \\ &= v^\top x + v^\top k \,. \end{aligned}$$

Now, if $u = v$, we get that $v^\top c(x) = u^\top x + u^\top k$. The correlation of constant addition, $\mathrm{C}_c(u, v)$, is thus equal to $\omega^{-v^\top k}$. Note that this is a correlation with absolute value 1 but with a different argument.

6.2.2 Linear Mask Propagation Through γ of $\mathcal{T}_q$

The non-linear layer γ performs four operations on each state digit. These four steps are:

1. Duplicate the state digit s_i once, such that two copies are present;
2. Square state digit s_{i+g} to obtain $(s_{i+g})^2$;
3. Perform an addition to obtain $s_i + (s_{i+g})^2$;
4. Save this result to state digit s_i.

To determine the correlation of a linear approximation over the non-linear layer, we should know the correlation of duplication, the correlation of squaring and the correlation of addition.

We first find the correlation of duplication. Let $d\colon \mathbb{F}_p \to \mathbb{F}_p \times \mathbb{F}_p$ be given by $d(x) = (x, x)$. Taking output mask $(v_1, v_2) \in (\mathbb{F}_p)^2$ yields:

$$\begin{aligned} (v_1, v_2)^\top d(x) &= (v_1, v_2)^\top (x, x) \\ &= v_1^\top x + v_2^\top x \\ &= (v_1^\top + v_2^\top) x \\ &= (v_1 + v_2)^\top x \,. \end{aligned}$$

If $u = v_1 + v_2$, we get the equation $(v_1, v_2)^\top d(x) = u^\top x$ and see that the correlation of duplication, $\mathrm{C}_d(u, (v_1, v_2))$, equals to 1.

Then, we determine the correlation of addition. Let $a\colon \mathbb{F}_p \times \mathbb{F}_p \to \mathbb{F}_p$ be given by $a(x, y) = x + y$. Taking output mask $v \in \mathbb{F}_p$, we get:

$$\begin{aligned} v^\top a(x, y) &= v^\top (x + y) \\ &= v^\top x + v^\top y \\ &= (v, v)^\top (x, y) \,. \end{aligned}$$

If $u_1 = u_2 = v$, we get the equation $v^\top a(x, y) = (u_1, u_2)^\top (x, y)$ and see that the correlation of addition, $\mathrm{C}_a((u_1, u_2), v)$, equals to 1.

Now, the only thing left to do is to determine the correlation of squaring. The inner product of the correlation is defined in Theorem 5.33 of [35] for $f(x) = a_2x^2 + a_1x + a_0 \in \mathbb{F}_p[x]$ with $a_2 \neq 0$ as follows:

$$\sum_{c \in \mathbb{F}_p} \hat{f}(c) = \widehat{\left(a_0 - a_1^2(4a_2)^{-1}\right)} \cdot \eta(a_2) \cdot G(\eta, \hat{}) \,.$$

Here, η represents the so-called Legendre symbol and G the Gaussian sum [35]. As the squaring function is defined as $f(x) = x^2$, we know that $a_2 = 1$, $a_1 = 0$ and $a_0 = 0$. We then get the following expression:

$$\hat{0} \cdot \eta(1) \cdot G(\eta, \hat{}) = \eta(1) \cdot G(\eta, \hat{}) \,.$$

The Legendre symbol $\eta(c) = \left(\frac{c}{p}\right)$ with $c \in \mathbb{F}_p^*$, expresses whether or not a value is a (non-)quadratic residue modulo p [35]. Note that if $c = 0$, the value stays 0. This indicates that the value $\eta(1)$ in our expression always equals to 1: $c = 1$ is always a (non-zero) quadratic residue modulo p. This yields the following expression for the inner product of the correlation:

$$1 \cdot G(\eta, \hat{}) = G(\eta, \hat{}) \,.$$

As stated before, the remaining term, $G(\eta, \hat{}) = \sum_{c \in \mathbb{F}_p^*} \psi(c)\hat{c}$, represents the Gaussian sum [35]. Here, ψ is the multiplicative character of $\mathbb{F}_p$, which means that ψ is a special group homomorphism from $\mathbb{F}_p$ to the multiplicative group of $\mathbb{F}_p^*$ [35]. For more information about characters, we refer to [35].

The exact value of this Gaussian sum can be expressed as is shown in Theorem 5.15 of [35]:

$$G(\eta, \hat{}) = \begin{cases} \sqrt{p} & \text{if } p \equiv 1 \pmod 4 \,; \\ i\sqrt{p} & \text{if } p \equiv 3 \pmod 4 \,. \end{cases}$$

To obtain the correlation, one should now get the scaled version of this sum:

$$\mathrm{C}_s(u, v) = \begin{cases} \frac{\sqrt{p}}{||b||^2} & \text{if } p \equiv 1 \pmod 4 \,; \\ \frac{i\sqrt{p}}{||b||^2} & \text{if } p \equiv 3 \pmod 4 \,. \end{cases}$$

Here, b represents the set of linear functions. However, the norms of all basis vectors are the same and is given by $||b||^2 = p$. Therefore, the correlation of squaring is equal to:

$$\mathrm{C}_s(u, v) = \begin{cases} \frac{\sqrt{p}}{p} & \text{if } p \equiv 1 \pmod 4 \,; \\ \frac{i\sqrt{p}}{p} & \text{if } p \equiv 3 \pmod 4 \,. \end{cases}$$

Then, if $a_2 = 0$ (so if the output mask v equals 0), the square is cancelled in the function and we have two cases to consider: either the input mask

$u = 0$ or $u \neq 0$. The only way to obtain a correlation equal to 1, is to set $u = 0$. This is because, by taking $u = 0$, the correlation between two complex vectors equals 1. Likewise, if $u \neq 0$, and we do not consider a correlation with the complex all-one vector, the correlation will be equal to 0.

To visualize why this particular correlation for squaring holds, we give an illustration of an example derivation over $\mathbb{F}_3$.

Illustration over $\mathbb{F}_3$ of the correlation of squaring

Let $\omega = e^{\frac{2\pi i}{3}}$ be a cube root over the complex numbers, such that raising it to the power of 3 equals to 1:

$$\begin{aligned}\omega^3 &= \left(e^{\frac{2\pi i}{3}}\right)^3 \\ &= e^{\frac{2\pi i}{3}\cdot 3} \\ &= e^{2\pi i} \\ &= -e^{\pi i} \\ &= -(-1) \\ &= 1\,.\end{aligned}$$

Note that this derivation uses Euler's identity [16], which states that $e^{\pi i}+1 = 0$. Now, consider Table 5. Here, we can see three linear functions and one non-linear function from one digit to another digit.

Table 5: Application of four linear functions on input x.

x	$x \mapsto 0$	$x \mapsto x$	$x \mapsto 2x$	$x \mapsto x^2$
0	0	0	0	0
1	0	1	2	1
2	0	2	1	1

In Table 5, each column belonging to a mapping will be called a value vector $v = (v_0, v_1, v_2)$. For example, the value vector of the mapping $x \mapsto x$ is equal to $v = (v_0, v_1, v_2) = (0, 1, 2)$. We define the function $\hat{f}(v) = (\omega^{v_0}, \omega^{v_1}, \omega^{v_2})$ to transform value vectors in $\mathbb{F}_3$ to value vectors in $\mathbb{C}$. If this function is applied to Table 5, this yields the results in Table 6.

Table 6: Application of $\hat{f}$ on the results of Table 5.

x	$\hat{f}((0,0,0))$	$\hat{f}((0,1,2))$	$\hat{f}((0,2,1))$	$\hat{f}((0,1,1))$
0	1	1	1	1
1	1	ω	ω^2	ω
2	1	ω^2	ω	ω

What can be deduced from Table 6 is that the three value vectors $\hat{f}((0,0,0))$, $\hat{f}((0,1,2))$ and $\hat{f}((0,2,1))$ function as orthogonal basis vectors for the belonging vector space. Their inner products namely all equal to 0:

$$\begin{aligned}
\langle \hat{f}((0,0,0)), \hat{f}((0,1,2)) \rangle &= \langle \hat{f}((0,0,0)), \hat{f}((0,2,1)) \rangle \\
&= \langle \hat{f}((0,1,2)), \hat{f}((0,0,0)) \rangle \\
&= \langle \hat{f}((0,2,1)), \hat{f}((0,0,0)) \rangle \\
&= 1 + \omega^2 + \omega \\
&= 0\,; \\
\langle \hat{f}((0,1,2)), \hat{f}((0,2,1)) \rangle &= \langle \hat{f}((0,2,1)), \hat{f}((0,1,2)) \rangle \\
&= 1 + \omega^2 + \omega^4 \\
&= 1 + \omega^2 + \omega \\
&= 0\,.
\end{aligned}$$

Why the addition $1 + \omega + \omega^2$ is equal to 0, is visualized in Figure 7 with the unit circle.

As stated before, we know that each vector v of a certain vector space can be expressed using the basis vectors $b \in B$:

$$v = \sum_{b \in B} \frac{\langle b, v \rangle}{||b||^2} \cdot b\,.$$

As it holds that $||\hat{f}((0,0,0))||^2 = ||\hat{f}((0,1,2))||^2 = ||\hat{f}((0,2,1))||^2 = 3$, we can write $v = \hat{f}((0,1,1))$ as shown in Equation 5.

One can now deduce the correlations between the mapping $x \mapsto x^2$ and the three mappings $x \mapsto 0$, $x \mapsto x$ and $x \mapsto 2x$ by looking at Equation 5. This is because the correlations are equal to the coefficients before respectively the value vectors $\hat{f}((0,0,0))$, $\hat{f}((0,1,2))$ and $\hat{f}((0,2,1))$. We thus get the following three correlations:

1. The correlation between $x \mapsto x^2$ and $x \mapsto 0$: $\frac{1}{\sqrt{3}} e^{-\frac{\pi i}{2}}$;
2. The correlation between $x \mapsto x^2$ and $x \mapsto x$: $\frac{1}{\sqrt{3}} e^{\frac{\pi i}{6}}$;
3. The correlation between $x \mapsto x^2$ and $x \mapsto 2x$: $\frac{1}{\sqrt{3}} e^{\frac{\pi i}{6}}$.

These correlations are visualized in Figure 8.

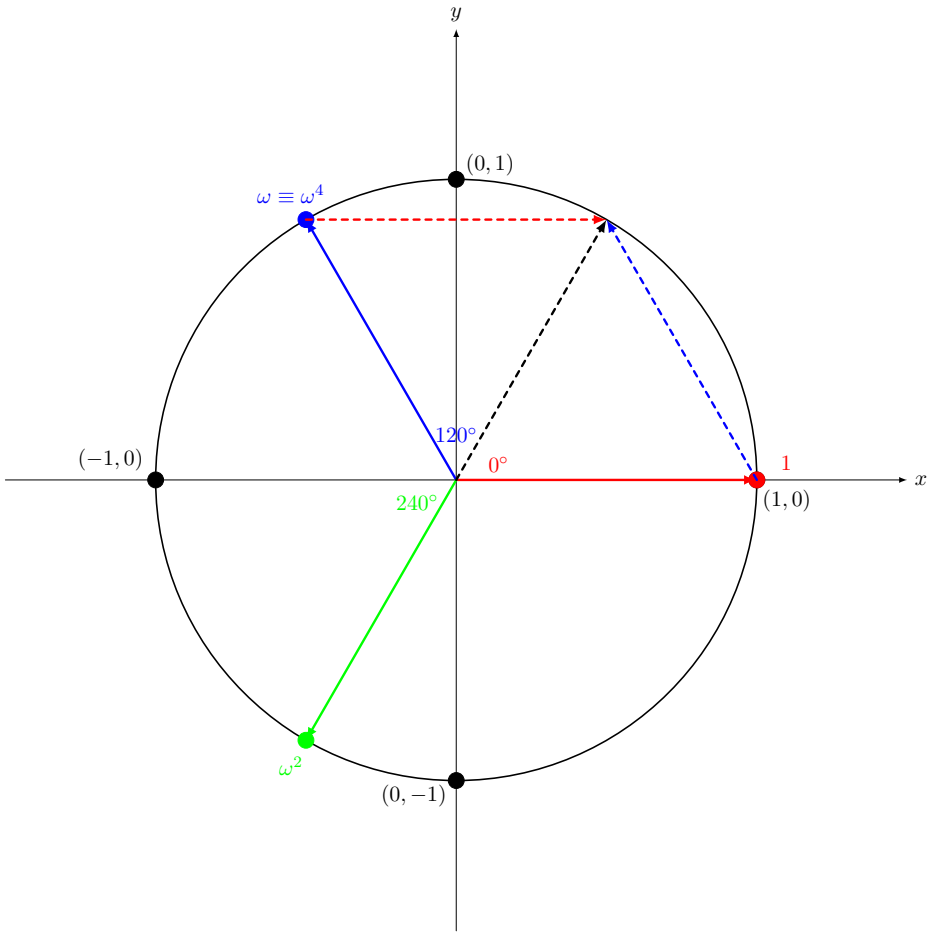


Figure 7: Visualization of basis vector addition (the addition of $1+\omega+\omega^2$) on the unit circle using $\omega = e^{\frac{2\pi i}{3}}$ with $\arg(\omega) = 120°$. The red vector represents the vector of 1, the blue vector represents the vector of ω and the green vector represents the vector of ω^2. Here, it is shown that first $1 + \omega$ is calculated. The result of this addition is shown as the black dashed vector. As the result of this addition is in the exact opposite direction of the green vector, the addition of all three terms yields 0.

$$
\begin{aligned}
\hat{f}((0,1,1)) = & \frac{1}{3} \cdot \langle \hat{f}((0,0,0)), \hat{f}((0,1,1)) \rangle \cdot \hat{f}((0,0,0)) + \\
& \frac{1}{3} \cdot \langle \hat{f}((0,1,2)), \hat{f}((0,1,1)) \rangle \cdot \hat{f}((0,1,2)) + \\
& \frac{1}{3} \cdot \langle \hat{f}((0,2,1)), \hat{f}((0,1,1)) \rangle \cdot \hat{f}((0,2,1)) \\
= & \frac{1}{3} \cdot \hat{f}((0,0,0)) \cdot \overline{\hat{f}((0,1,1))} \cdot \hat{f}((0,0,0)) + \\
& \frac{1}{3} \cdot \hat{f}((0,1,2)) \cdot \overline{\hat{f}((0,1,1))} \cdot \hat{f}((0,1,2)) + \\
& \frac{1}{3} \cdot \hat{f}((0,2,1)) \cdot \overline{\hat{f}((0,1,1))} \cdot \hat{f}((0,2,1)) \\
= & \frac{1}{3} \cdot (1,1,1) \cdot (1,\omega^2,\omega^2)^\top \cdot \hat{f}((0,0,0)) + \\
& \frac{1}{3} \cdot (1,\omega,\omega^2) \cdot (1,\omega^2,\omega^2)^\top \cdot \hat{f}((0,1,2)) + \\
& \frac{1}{3} \cdot (1,\omega^2,\omega) \cdot (1,\omega^2,\omega^2)^\top \cdot \hat{f}((0,2,1)) \\
= & \frac{1}{3} \cdot (1+\omega^2+\omega^2) \cdot \hat{f}((0,0,0)) + \\
& \frac{1}{3} \cdot (1+1+\omega) \cdot \hat{f}((0,1,2)) + \\
& \frac{1}{3} \cdot (1+\omega+1) \cdot \hat{f}((0,2,1)) \\
= & \frac{1}{3} \cdot \left(1 + \left(e^{\frac{2\pi i}{3}}\right)^2 + \left(e^{\frac{2\pi i}{3}}\right)^2\right) \cdot \hat{f}((0,0,0)) + \\
& \frac{1}{3} \cdot \left(2 + e^{\frac{2\pi i}{3}}\right) \cdot \left(\hat{f}((0,1,2)) + \hat{f}((0,2,1))\right) \\
= & \frac{1}{3} \cdot \left(1 + 2e^{\frac{-2\pi i}{3}}\right) \cdot \hat{f}((0,0,0)) + \\
& \frac{1}{3} \cdot \left(2 + e^{\frac{2\pi i}{3}}\right) \cdot \left(\hat{f}((0,1,2)) + \hat{f}((0,2,1))\right) \\
= & \frac{1}{3} \cdot \sqrt{3} e^{-\frac{\pi i}{2}} \cdot \hat{f}((0,0,0)) + \\
& \frac{1}{3} \cdot \sqrt{3} e^{\frac{\pi i}{6}} \cdot \left(\hat{f}((0,1,2)) + \hat{f}((0,2,1))\right) \\
= & \frac{1}{\sqrt{3}} e^{-\frac{\pi i}{2}} \cdot \hat{f}((0,0,0)) + \\
& \frac{1}{\sqrt{3}} e^{\frac{\pi i}{6}} \cdot \left(\hat{f}((0,1,2)) + \hat{f}((0,2,1))\right) .
\end{aligned}
\tag{5}
$$

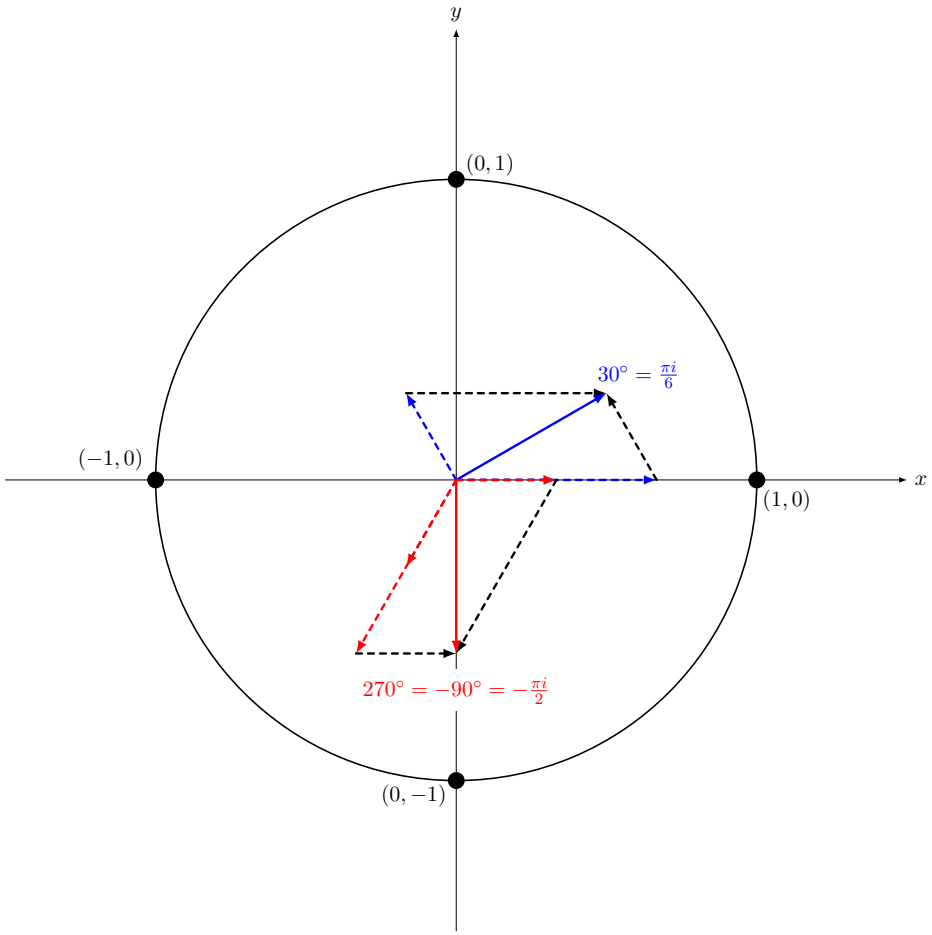


Figure 8: Visualization of the found correlations. The red vector depicts the correlation between the mappings $x \mapsto x^2$ and $x \mapsto 0$ and the blue vector shows the correlation between the mappings $x \mapsto x^2$ and $x \mapsto x$ and between the mappings $x \mapsto x^2$ and $x \mapsto 2x$.

6.3 Linear Potential

Using the correlation, one can specify the linear potential [14, 25]: see Definition 14.

Definition 14. *Let* $\mathrm{C}_f(u, v)$ *be the correlation between the input mask* u *and output mask* v *over a* p*-ary transformation* f*. The* linear potential (LP) of a linear approximation (u, v) over f *is defined as*

$$\mathrm{LP}_f(u, v) = \mathrm{C}_f(u, v)\overline{\mathrm{C}_f(u, v)} = |\mathrm{C}_f(u, v)|^2 .$$

Note that the linear potential expresses a real number instead of a complex number [14, 25].

After having introduced the linear potential, we define the weight belonging to this LP as the (correlation) weight [14]. This weight only exists if the linear potential is not equal to zero, i.e., when $\mathrm{LP}_f(u,v) \neq 0$ [14]: see Definition 15.

Definition 15. *Let* $\mathrm{LP}_f(u,v)$ *be the LP of a linear approximation* (u,v) *over a* p*-ary transformation* f*. The* (correlation) weight of a linear approximation (u,v) over f *that satisfies* $\mathrm{LP}_f(u,v) \neq 0$ *is defined as*

$$\mathrm{w_c}(u,v) = -\log_p\left(\mathrm{LP}_f(u,v)\right)$$

which can be equivalently written as

$$p^{-\mathrm{w_c}(u,v)} = \mathrm{LP}_f(u,v)\,.$$

One can see resemblance between this definition and Definition 6. This means that we can ask a similar question: "How can we easily compute $\mathrm{LP}_f(u,v)$ and $\mathrm{w_c}(u,v)$, with f a p-ary transformation containing multiple iterations of the round function R, if the state size n (and thus implicitly p^n) gets too big to exhaustively compute by hand or by computer?". We will answer this question by looking at the LP of the non-linear layer separately.

6.3.1 Relation Between Hamming Weight and LP

As we know the correlations of all of the separate operations that happen in γ, we also know their LP. Some of these linear potentials are also shown in [25]. In particular, the LP of duplication is equal to

$$\mathrm{LP}_d(u,(v_1,v_2)) = \begin{cases} 1 & \text{if } u = v_1 + v_2\,; \\ 0 & \text{otherwise}\,, \end{cases}$$

the LP of addition to

$$\mathrm{LP}_a((u_1,u_2),v) = \begin{cases} 1 & \text{if } u_1 = u_2 = v\,; \\ 0 & \text{otherwise}\,, \end{cases}$$

and the LP of squaring to

$$\mathrm{LP}_s(u,v) = \begin{cases} \frac{1}{p} & \text{if } v \neq 0\,; \\ 1 & \text{if } u = 0 \text{ and } v = 0\,; \\ 0 & \text{if } u \neq 0 \text{ and } v = 0\,. \end{cases}$$

We can now look at the linear potential of the entire non-linear layer. This layer is shown in Figure 9 as a circuit of operation blocks.

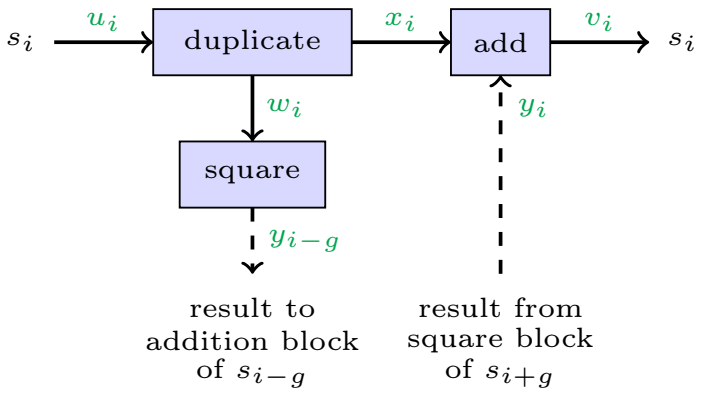


Figure 9: Schematic overview of the non-linear layer γ: $s_i = s_i + s_{i+g}^2$ for $0 \leq i < n$ using the operation blocks of duplication, addition and squaring. The green letters $*_i$ indicate index i of mask $*$.

Now, let the mask $u \in \mathbb{F}_p^n$ be defined at the input of the duplication block, the mask $w \in \mathbb{F}_p^n$ between the output of the duplication block and the input of the squaring block, the mask $x \in \mathbb{F}_p^n$ between the output of the duplication block and the input of the addition block, the mask $y \in \mathbb{F}_p^n$ between the output of the squaring block and the input of the addition block and the mask $v \in \mathbb{F}_p^n$ at the output of the addition block: see also Figure 9. We will consider four cases to say something about the LP of the full non-linear layer:

1. The mask element $v_i = 0$ and the mask element $v_{i-g} = 0$.
   As the $\mathrm{LP}_a((u_1, u_2), v)$ is 1 if and only if $u_1 = u_2 = v$ and we are given that $v_i = 0$, this yields that $x_i = y_i = v_i = 0$. Similarly, we obtain $x_{i-g} = y_{i-g} = v_{i-g} = 0$. Then, by $\mathrm{LP}_s(u, v)$, if $v = 0$, this LP is 1 if and only if $u = 0$. We know that $y_{i-g} = 0$, so $w_i = 0$. Lastly, as $\mathrm{LP}_d(u, (v_1, v_2))$ is 1 if and only if $u = v_1 + v_2$, we obtain that $u_i = x_i + w_i = 0 + 0 = 0$.
2. The mask element $v_i = 0$ and the mask element $v_{i-g} = 1$.
   Following the same reasoning as in case 1, we obtain: $x_i = y_i = v_i = 0$, $x_{i-g} = y_{i-g} = v_{i-g} = 1$, $w_i \in \mathbb{F}_p$ and $u_i = x_i + w_i \in \mathbb{F}_p$. Note that $w_i \in \mathbb{F}_p$ as $\mathrm{LP}_s(u, v)$ with $v \neq 0$ has an undetermined value for u.
3. The mask element $v_i = 1$ and the mask element $v_{i-g} = 0$.
   Following the same reasoning as in case 1, we obtain: $x_i = y_i = v_i = 1$, $x_{i-g} = y_{i-g} = v_{i-g} = 0$, $w_i = 0$ and $u_i = x_i + w_i = 1 + 0 = 1$.
4. The mask element $v_i = 1$ and the mask element $v_{i-g} = 1$.
   Following the same reasoning as in case 1 and 2, we obtain: $x_i = y_i = v_i = 1$, $x_{i-g} = y_{i-g} = v_{i-g} = 1$, $w_i \in \mathbb{F}_p$ and $u_i = x_i + w_i \in \mathbb{F}_p$.

What we can deduce from these cases is that if all coordinates of the output mask v are set to zero, the input mask u is also the all-zero mask, as expected.

Besides this, the only way to have coordinate i of the input mask u obtain any value in $\mathbb{F}_p$, the coordinate $i-g$ of the output mask v should be set to a non-zero value. This behavior is caused by the LP of squaring. In all other cases, i.e., when coordinate $i-g$ of the output mask v is set to zero, the value of coordinate i of the input mask u is fixed to a value $\mathbb{F}_p$.

This thus means that we found a similar relation for the linear potential as we did for the DP: the LP is equal to $p^{-(\text{number of non-zero entries in output mask } v)}$. The relation we can thus specify is as follows:

$$\begin{aligned} \mathrm{LP}_\gamma(u,v) &= p^{-\mathrm{HW}(v)} \\ &= p^{-\mathrm{w_c}(u,v)} \,. \end{aligned}$$

The correlation weight of a linear approximation (u,v) over p-ary transformations thus equals the Hamming weight of the output mask v :

$$\mathrm{w_c}(u,v) = \mathrm{HW}(v) \,.$$

The LP, just like the DP, can thus be more easily computed using the Hamming weight instead of exhaustively listing all possibilities.

6.3.2 Non-invertibility of γ

A non-zero output mask v can only be imbalanced (and thereby causing a collision) if 0 is its offset in the affine space. This can only happen if for all positions it holds that both the positions v_i and v_{i+1} are active, i.e., if the output mask v has no coordinates equal to zero. There are $(p-1)^n$ of such output masks, for which $\mathrm{LP}_\gamma(0,v) = p^{-n}$.

Again, we can quantify the non-invertibility of γ by looking at the collision probability. For the non-linear layer γ, the collision probability is equal to the number of colliding pairs divided by the total number of pairs and thus:

$$\frac{(p-1)^n}{\binom{p^n}{2}} = \frac{2(p-1)^n}{p^n(p^n-1)} \approx \frac{2(p-1)^n}{p^{2n}} \,.$$

Therefore, if we look at the linear propagation, the non-invertibility of γ is also no problem.

6.4 Round Linear Approximations

Just as there were round differentials defined over a round function R_i, one can also define linear approximations over it. Let q_i be the input mask of round R_i and q_{i+1} the output mask of round R_i and, at the same time, the input mask of round R_{i+1}. The (ordered) pair $(q_i, q_{i+1}) \in \mathbb{F}_p^n \times \mathbb{F}_p^n$ containing the input and output mask of round R_i is called a *round linear approximation* [14].

6.5 Linear Trails

The sequence we can build using round linear approximations is the so-called k-round linear trail [14]: see Definition 16.

Definition 16. *A* k-round linear trail *is a sequence of* $k+1$ *mask patterns* $Q_k = (q_0, q_1, \ldots, q_k) \in \left(\mathbb{F}_p^n\right)^{k+1}$ *that satisfies* $\mathrm{C}_{\mathrm{R}_i}(q_i, q_{i+1}) \neq 0$ *for* $0 \leq i < k$.

Note that if we write $Q \in (q_0, q_k)$, we indicate that Q starts in input mask q_0 and ends in output mask q_k.

If the initial mask q_0 and the final mask q_k is left out, one obtains the *linear trail core* $(q_1, q_2, \ldots, q_{k-1})$ [14]. This trail core holds a collection of linear trails with the same inner linear masks [14].

Next, we define the correlation contribution of a k-round linear trail [14]. This is shown in Definition 17.

Definition 17. *Let* $f \colon \mathbb{F}_p^n \to \mathbb{F}_p^n$ *be a* p*-ary transformation and* $Q_k = (q_0, q_1, \ldots, q_k)$ *a* k*-round linear trail. The* correlation contribution of a k-round linear trail Q_k *is defined as*

$$\mathrm{C}_f(Q_k) = \prod_{i=0}^{k-1} \mathrm{C}_{\mathrm{R}_i}\left(q_i, q_{i+1}\right) .$$

Using correlation matrices [18, 48], we can state that the correlation contribution of a k-round linear approximation equals to the sum of the correlation contributions of the trails in it [14]:

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

Finally, we introduce the (correlation) weight for linear trails [14]. This weight metric is defined in Definition 18.

Definition 18. *Let* $Q_k = (q_0, q_1, \ldots, q_k)$ *be a* k*-round linear trail. The* (correlation) weight of a k-round linear trail Q_k *is defined as*

$$\mathrm{w_c}(Q_k) = \sum_{i=0}^{k-1} \mathrm{w_c}\left(q_i, q_{i+1}\right) .$$

6.6 Trail Search

Like differentials with a high-probability can be exploited, linear approximations can be exploited if these have a high correlation [14, 36]. So, in other words, if we find a trail with a high correlation contribution, it has a low correlation weight and poses a potential weakness. Therefore, we again focus

on finding a lower bound on the trail weights over k rounds. This can be done with the same method as explained in Section 5.5.

Again, we slightly change the representation of a k-round trail. However, instead of looking at the input differences and figuring out which output differences are compatible, we investigate output masks and see which input masks belong to this output. Because of this, propagating through the linear layer involves the transpose of the corresponding matrix [21]. The transpose of the linear layer is equal to $\lambda^\top = (\rho \circ \theta)^\top = \theta^\top \circ \rho^\top$. Note that the non-linear layer stays the same. The k-round linear trail Q_k will then be represented as follows:

$$Q_k = \left(v^0, u^0, v^1, u^1, \ldots, u^{k-1}, v^k\right) .$$

Here, v^i is used to represent the i^{th} input of the transposed linear layer and u^i is used to represent the i^{th} input of the non-linear layer.

6.6.1 Trail Extension in the Forward and Backward Direction

Trail extension in both the forward and backward direction is very similar for linear trails as for differential trails as was also the case in [21]. This thus means that extension in the forward direction, i.e., extending a k-round linear trail $Q_k = \left(v^0, u^0, v^1, u^1, \ldots, u^{k-1}, v^k\right)$ by one round to the set of trails $Q_{k+1} = \left(v^0, u^0, v^1, u^1, \ldots, u^{k-1}, v^k, u^k, v^{k+1}\right)$, can be done by first computing $u^k = \lambda^\top(v^k)$ and afterwards building all compatible input masks v^{k+1} over γ. The minimum direct weight can therefore be defined as is stated in Definition 19.

Definition 19. *Let* $\mathrm{w_c}\left(u^k, v^{k+1}\right)$ *be the weight of the linear approximation* $\left(u^k, v^{k+1}\right)$. *The* minimum direct weight *is defined as*

$$\mathrm{w^{dir}}\left(u^k\right) := \min_{v^{k+1}} \mathrm{w_c}\left(u^k, v^{k+1}\right) .$$

Likewise, extension in the backward direction, i.e., extending a linear trail $Q_k = \left(v^1, u^1, \ldots, u^{k-1}, v^k, u^k, v^{k+1}\right)$ by one round to the set of trails $Q_{k+1} = \left(v^0, u^0, v^1, u^1, \ldots, u^{k-1}, v^k, u^k, v^{k+1}\right)$ can be achieved by first building all compatible output masks u^0 over γ and secondly computing $v^0 = (\lambda^\top)^{-1}(u^0)$. Then, the minimum reverse weight can be expressed as given in Definition 20.

Definition 20. *Let* $\mathrm{w_c}\left(u^0, v^1\right)$ *be the weight of the linear approximation* $\left(u^0, v^1\right)$. *The* minimum reverse weight *is defined as*

$$\mathrm{w^{rev}}\left(v^1\right) := \min_{u^0} \mathrm{w_c}\left(u^0, v^1\right) .$$

6.6.2 Search Strategy Using Trail Cores

Combining the definitions of the minimum direct weight, minimum reverse weight and linear trail cores, one can bound the weight of a linear trail core as follows:

$$\mathrm{w_c}\left(Q_k\right) = \mathrm{w^{rev}}\left(v^1\right) + \sum_{i=2}^{k-1} \mathrm{w_c}\left(u^{i-1}, v^i\right) + \mathrm{w^{dir}}\left(u^{k-1}\right) .$$

Having this expression, one can perform the same search strategy as explained in Section 5.5.4 and [30].

6.7 Tools for Linear Trail Search on $\mathcal{T}_q$

We have shown that the (dual) relations $\mathrm{DP}_s(b,a) = \frac{1}{q}$ and $\mathrm{LP}_s(u,v) = \frac{1}{p}$, but also $\mathrm{DP}_\gamma(b,a) = q^{-\mathrm{HW}(b)}$ and $\mathrm{LP}_\gamma(u,v) = p^{-\mathrm{HW}(v)}$ hold. Because of the similarity of the relations above, the masks propagate in a similar fashion over the non-linear layer γ as the differences do. To make this concrete, let $f_{-1}\colon \mathbb{F}_p^n \to \mathbb{F}_p^n$ be given by $f_{-1}(x) = y$ where $\forall i\colon y_{-i} = x_i$. Then, for $v = f_{-1}(b)$ and $u = f_{-1}(a)$, we get that $\mathrm{DP}_\gamma(b,a) = \mathrm{LP}_\gamma(u,v)$. One should recall that we are investigating in the direction of output to input masks instead of from input to output differences. Another important thing to note is that the indices should be reversed: the index i in an input and/or output mask corresponds to the index $-i$ in an input and/or output difference. This is because of the reverse direction and the specification of γ. If we namely go from input difference to output difference, we start with state digit s_i and obtain state digit s_{i+1}, but if we go from output mask to input mask, we start with state digit s_i and obtain state digit s_{i-1}.

Therefore, to generate all compatible input masks through γ, one can make use of the affine space as explained in Algorithm 2. To generate all compatible output masks through γ, Algorithm 3 should be followed. Likewise, to find the minimum direct weight, we can use the 1-runs as specified in Algorithm 4 and to find the minimum reverse weight, we simply take the Hamming weight.

The results of the linear trail search are, just like the ones for differential trail search, left as future work. By using the provided algorithms, one should be able to perform a successful linear trail search.

Chapter 7

Practical Applications of $\mathcal{T}_q$

After defining $\mathcal{T}_q$ and analyzing its diffusion and propagation properties, we will look into some use cases for it. In particular, one can use such transformations in sponge and duplex constructions, in encryption schemes such as Ciminion or even in authenticated encryption schemes such as Elephant. Designing concrete instances of $\mathcal{T}_q$ for these applications is left as future work.

7.1 Sponge Construction

A sponge construction is "a mode of operation, based on a fixed-length permutation (or transformation) and on a padding rule, which builds a function mapping variable-length input to variable-length output" [6]. Such sponge construction can be denoted by $Z = \textsc{Sponge}[f, \text{pad}, r](M, l)$ where f is either a permutation or a transformation operating on b digits, pad a certain padding rule, r the bitrate, M the message and l the output length [7]. An illustration of this construction can be seen in Figure 10. Note that the first r digits of the state are called the outer part and the last $b - r = c$ digits of the state are called the inner part [7]. Also note that the capacity c "determines the attainable security level of the construction" [7].

An instance of such sponge construction is called a sponge function, denoted with $\textsc{Sponge}[f, \text{pad}, r]$ [7]. As stated before, f is either a permutation or a transformation. This means that $\mathcal{T}_q$ operating on b digits could be used within a sponge function. The sponge construction can then be explained in three steps.

The first step is the initialization. In this step, all b digits of the state are initialized to zero [6, 7].

The second step is the absorbing phase. In this phase, the outer part of the state is added to the r-digit input message blocks, interleaved with applications of $\mathcal{T}_q$ [6, 7].

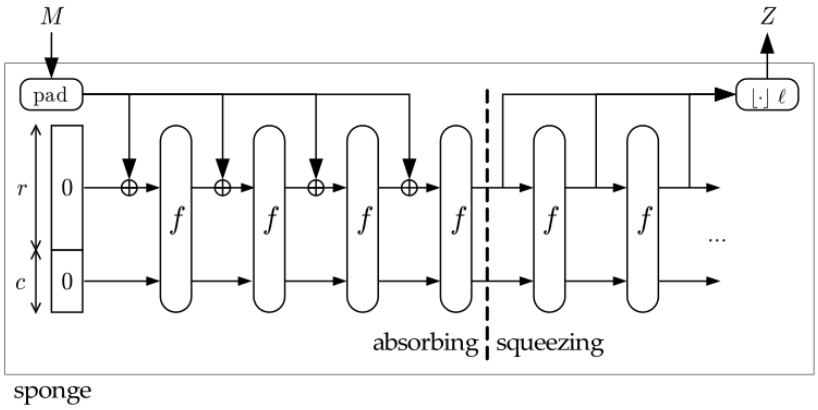


Figure 10: The sponge construction $Z = \text{SPONGE}[f, \text{pad}, r](M, l)$ [6].

After all message blocks are absorbed, the squeezing phase starts. In this final phase, the outer part of the state is returned as output blocks, interleaved with applications of $\mathcal{T}_q$ [6, 7]. The number of output blocks is determined by the parameter l given to the sponge construction: the output is truncated to, in total, l digits [7].

Note that, besides the original sponge structure, there also exist keyed sponges [40], in which $\mathcal{T}_q$ can be used as well. In keyed sponges, a key K is given as additional parameter. An example of using such key is in the initialization step: instead of initializing everything to zero, one could initialize the inner part of the sponge with K [40]. This construction can then be used for message authentication (by using a tag resulting from the sponge) or keystream generation (truncated to the length l) [40].

7.2 Duplex Construction

A cryptographic scheme that is closely related to the sponge construction and has an equivalent security is the duplex construction [6]. The duplex construction can be denoted by $Z = \text{DUPLEX}[f, \text{pad}, r](\sigma, l)$ where f is either a permutation or a transformation operating on b digits, pad a certain padding rule, r the bitrate, σ the input string/message and l the output length [7]. A visualization of a duplex construction can be seen in Figure 11. Note that $\sigma = (\sigma_0, \sigma_1, \ldots, \sigma_o)$, $Z = (Z_0, Z_1, \ldots, Z_o)$ and $l = (l_0, l_1, \ldots, l_o)$. This is because "unlike a sponge function that is stateless in between calls, the duplex construction results in an object that accepts calls that take an input string and return an output string that depends on all inputs received so far" [7].

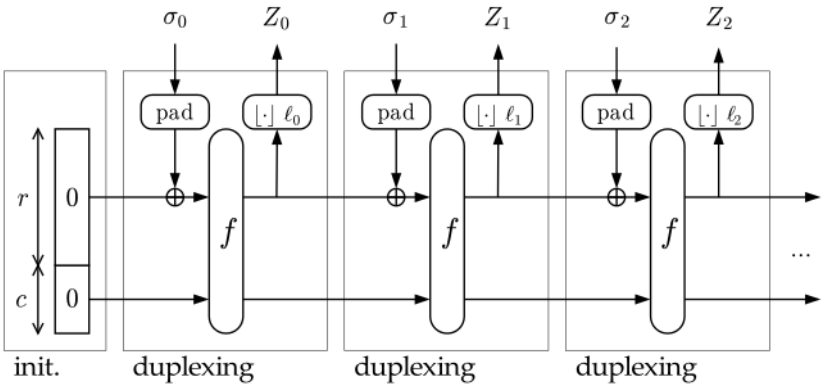


Figure 11: The duplex construction $Z = \text{DUPLEX}[f, \text{pad}, r](\sigma, l)$ [6].

An instance of a duplex construction is called a duplex object $D = \text{DUPLEX}[f, \text{pad}, r]$ and operates on b digits [7]. Again, as f is either a permutation or transformation, one could use $\mathcal{T}_q$ in this place. Duplexing with this setting then starts with initializing all digits of the state to zero [7]. After this, one can send the following to the initialized state: $D.\text{duplexing}(\sigma, l)$ where σ denotes an input string and where l represents the number of digits that will be output [7]. When D receives this command, it pads σ according to the provided padding rule, adds the result to the outer part of the state, applies $\mathcal{T}_q$ and, in the end, returns the outer part of the state truncated to l digits [7].

Just like keyed sponges existed, keyed duplexes also exist. Again, the key can be used in the initialization step [40]. This set-up can be used for authenticated encryption [40], of which SPONGEWRAP [50] an example is.

7.3 Encryption Scheme Ciminion

Ciminion is an "encryption scheme minimizing the number of field multiplications in large binary or prime fields, while using a very lightweight linear layer" [25]. Its encryption scheme is a nonce-based stream-encryption scheme [25] and is shown in Figure 12.

In Figure 12, one can see two permutations: the permutation p_C that takes a nonce and two keys as input and the permutation p_E whose output is added to the plaintext to obtain the ciphertext. As the inverse of these permutations is never used – neither in encryption nor in decryption [25] – one could replace such permutation with a transformation. This means that $\mathcal{T}_q$ could be used as one of these permutations p_C or p_E.

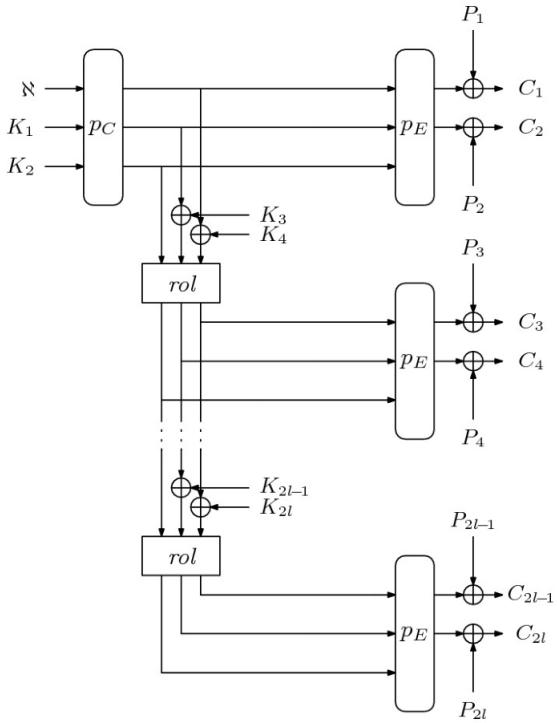


Figure 12: Encryption over $\mathbb{F}_{2^n}$ using Ciminion [25]. Note that encryption over $\mathbb{F}_p$ is similar: one should replace $\oplus$ by $+$ (the addition modulo p) [25].

Encryption using Ciminion then works as follows. First, the permutation p_C takes a nonce $\aleph$ and two subkey elements K_1 and K_2 as input and outputs an intermediate state [25]. The permutation p_E is then applied to the state and the output is truncated to two elements such that plaintexts P_1 and P_2 can be encrypted [25]. Note that it can happen that one needs more plaintext to be encrypted. In this case, the intermediate state should 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*" [25]. After this rolling function, two more plaintexts P_{2i} and P_{2i+1} can be encrypted after applying the permutation P_E to the resulting state [25].

7.4 Authenticated Encryption Scheme Elephant

Elephant is a "family of lightweight authenticated encryption schemes" [8]. The mode of Elephant is a "nonce-based encrypt-then-MAC construction, where encryption is performed using counter mode and message authentication using a variant of the protected counter sum MAC function" [9]. A visualization of Elephant is shown in Figure 13.

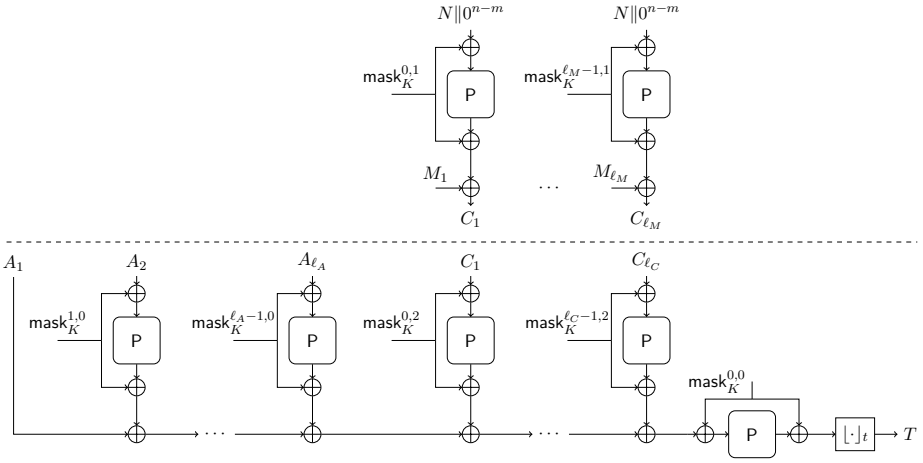


Figure 13: The authenticated encryption scheme Elephant [9]. Encryption is shown in the upper half of the figure. In the encryption, the message is padded as $M_1, \ldots, M_{l_M} \xleftarrow{n} M$ and the ciphertext equals $C = \lfloor C_1, \ldots, C_{l_M} \rfloor_{|M|}$ [9]. The authentication is shown in the lower half of the figure. In the authentication, the nonce and associated data are padded as $A_1, \ldots, A_{l_A} \xleftarrow{n} N||A||1$ and the ciphertext is padded as $C_1, \ldots, C_{l_C} \xleftarrow{n} C||1$ [9].

As can be seen in Figure 13, the method of authenticated encryption takes a key K, a nonce N, associated data A and message M as input and outputs a ciphertext C and a tag T [9]. Note that this ciphertext and tag are obtained using a permutation P.

If one wants to decrypt the obtained ciphertext C, one should provide the decryption algorithm with a key K, a nonce N, associated data A, the ciphertext C and the belonging tag T [9]. It then either returns the message M (if the tag is correct) or returns $\perp$ (if the tag is incorrect) [9]. Note that decryption uses the same permutation P as in the encryption process.

As the same permutation P is used in encryption and decryption, and because the permutation is only evaluated in the forward direction [8, 9], one could replace this permutation with a transformation like we did for Ciminion. Therefore, one could use $\mathcal{T}_q$ for authenticated encryption using it as the permutation P in Elephant.

Chapter 8

Case Study: Ternary Transformation τ in $\mathcal{T}_3$

In this chapter, we will treat an example instance of $\mathcal{T}_3$ called τ. First, we encode the trits to a set of bits and after this, we define arithmetic of the trits using this encoding. Then, we will clarify the state τ will operate on. Following on this, the round function of τ is established using the avalanche behavior. Then, we will look into the cryptanalysis as described in the previous chapters. To finish off, we describe how τ could be implemented in code.

8.1 Encoding of Trits

To obtain an efficient (hardware) implementation of τ, we need an efficient way of doing arithmetic in $\mathbb{F}_3$. This is possible by implementing the round function with only shift and bitwise Boolean instructions. However, this requires us to encode trits with a set of bits. Note that this does not change the specification of τ as this choice is only important for the implementation.

As one bit can only encode two values, we will look into an encoding using two bits. As there are multiple possibilities to assign two bits to each trit value, it yields that this encoding can also be established in multiple ways. Other researches [13, 31] have already invested time in finding an efficient representation. Their findings were that the representations in Table 7 proved to be good candidates.

Note that the first bit of an encoded value x is denoted by x_0 and the second bit by x_1. Hence, if e.g., trit value 1 is encoded with $x = 10$, we have $x_0 = 1$ and $x_1 = 0$.

All these encodings are good candidates as they can perform addition in $\mathbb{F}_3$ in six logical instructions [13, 31], which is proven to be the minimum number of logical instructions for $\mathbb{F}_3$-addition [31]. Note that, with encoding 1, addition can be performed by using only six logical instructions if `ANDN` is

Table 7: Promising encodings of trits with two bits [13, 31].

#	**Trit value 0**	**Trit value 1**	**Trit value 2**
1.	00	01	10
2.	00	10	11
3.	11	01	10
4.	00	01	11

counted as one instead of two instructions [31].

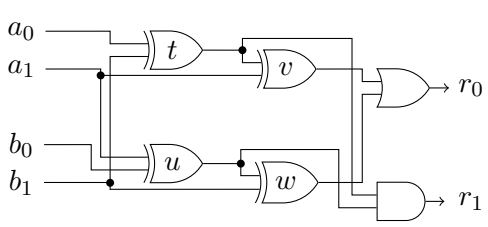
In our research, we will be using encoding 2. Besides an efficient method for addition, this encoding namely also provides efficient implementations for subtraction [13], negation [13], squaring and addition of a square. The specifications of all these operations are given in the following sections.

8.2 Arithmetic for τ

After having defined the encoding of trits, we can describe how to do addition, subtraction, negation and squaring in $\mathbb{F}_3$.

8.2.1 Addition

Addition of two terms in $\mathbb{F}_3$ using encoding 2 can be done using six bitwise operations [13]. In order to perform the operation $r = a+b$ where $a, b, r \in \mathbb{F}_3$, four intermediate values $t, u, v, w \in \mathbb{F}_2$ are needed. Addition in $\mathbb{F}_3$ is achieved by following the steps in Figure 14.



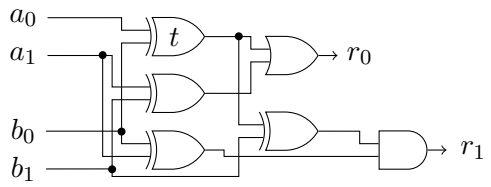
$$t \leftarrow a_0 \oplus b_1$$
$$u \leftarrow a_1 \oplus b_0$$
$$v \leftarrow t \oplus a_1$$
$$w \leftarrow u \oplus b_1$$
$$r_0 \leftarrow v \vee w$$
$$r_1 \leftarrow t \wedge u$$

Figure 14: Addition in $\mathbb{F}_3$.

8.2.2 Subtraction

Subtraction of two terms in $\mathbb{F}_3$ using encoding 2 can be done by first negating the second operand and then using the addition formula. However, there is a more efficient way as described in [13]. To calculate $r = a - b = a + 2b$ where $a, b, r \in \mathbb{F}_3$, again only six bitwise operations are needed. We will only

use one intermediate value $t \in \mathbb{F}_2$. Subtraction in $\mathbb{F}_3$ can then be defined as shown in Figure 15.

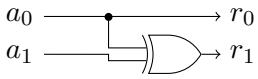


$$t \leftarrow a_0 \oplus b_0$$
$$r_0 \leftarrow t \vee (a_1 \oplus b_1)$$
$$r_1 \leftarrow (t \oplus b_1) \wedge (b_0 \oplus a_1)$$

Figure 15: Subtraction in $\mathbb{F}_3$.

8.2.3 Negation

Negation in $\mathbb{F}_3$ is equal to multiplication with two and, using encoding 2, can be done in just one bitwise operation [13]. Note that the result of the negation of $a \in \mathbb{F}_3$ is again stored in $r \in \mathbb{F}_3$: see Figure 16.

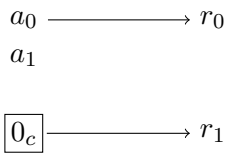


$$r_0 \leftarrow a_0$$
$$r_1 \leftarrow a_0 \oplus a_1$$

Figure 16: Negation in $\mathbb{F}_3$.

8.2.4 Squaring

Squaring a trit using encoding 2 is for free, i.e., it does not cost any bitwise operations. This is the case as the second bit will always be 0 and the first bit does not need to be changed in order to obtain the square value $r = a^2$ where $a, r \in \mathbb{F}_3$. This can be seen in Figure 17:

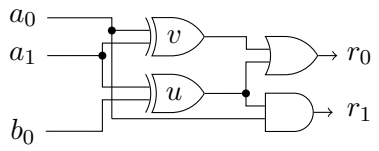


$$r_0 \leftarrow a_0$$
$$r_1 \leftarrow 0$$

Figure 17: Squaring in $\mathbb{F}_3$. Here, 0_c indicates the constant value 0.

8.2.5 Addition of a Square

Computing the addition of a square using encoding 2, i.e., if we want to compute $r = a + b^2$ where $a, b, r \in \mathbb{F}_3$, one could first square and follow this computation by an addition. This would lead to a cost of $6 + 0 = 6$ bitwise operations. However, as the result of squaring always yields 0 in the second bit, we automatically get $b_1 = 0$ in the addition formula. As it holds that $x \oplus 0 = x$, the addition of a square can be done in 4 operations instead of 6 by using two intermediate values $u, v \in \mathbb{F}_2$. Addition of a square in $\mathbb{F}_3$ is achieved by following the steps in Figure 18.



$$u \leftarrow a_1 \oplus b_0$$
$$v \leftarrow a_0 \oplus a_1$$
$$r_0 \leftarrow v \vee u$$
$$r_1 \leftarrow a_0 \wedge u$$

Figure 18: Addition of a square in $\mathbb{F}_3$.

8.2.6 Addition of Three Terms

Recall that the mixing layer θ is constructed as a sum of multiple state digits. As θ of our instance τ uses addition of three terms (as elaborated upon in Section 8.4.1), the following question arises: "Is there a more efficient way of adding three trits than twice adding two trits?". In other words: can we perform addition of three terms in $\mathbb{F}_3$ in less than $6 \cdot 2 = 12$ bitwise operations? We investigated this but unfortunately could not find a more efficient method.

We looked at all four encodings listed in Table 7 and tried to find a method to add three trits by using less than 12 bitwise operations. We tried so by using brute-force, but also studied methods as described in [41, 43]. Several algorithms rely on minimizing a sub-circuit and then automatically minimizing the full circuit [41, 43]. However, as proved by [31], the minimum number of logical instructions for $\mathbb{F}_3$-addition is equal to six. This thus means that the sub-circuit is already optimized and thus, though this way, we cannot optimize the full circuit used to add three trits.

Even though we could not find a more optimal method for addition of three terms in $\mathbb{F}_3$, it is possible that it still exists. This is left as future work.

8.3 State of τ

The state that τ will operate on is equal to the one shown in Figure 2. This means that we will be using $d = 3$ rows and $2^m = 2^6 = 64$ columns, which yields a state size of $n = d \cdot 2^m = 3 \cdot 64 = 192$.

8.4 Possible Values of the Parameters in τ

In Section 3.3, we have discussed what the possible values of the parameters of all step functions are for $\mathcal{T}_q$. Here, we will discuss the possible parameter values explicitly for τ and pick the parameter values we start off with. In the next section, Section 8.5, these choices will be tweaked if deemed necessary.

8.4.1 Parameter t in θ

In Section 3.3.1 we learned that the associated polynomial of the mixing layer, $S \leftarrow S \cdot \sum_{j=0}^{n-1} \left(c_j \cdot X^j\right) \bmod X^n - 1$, should be coprime to the polynomial $X^n - 1$. The first step we take is choosing the parameters c_j in such a way that this happens. For τ, we will choose $c_0 = 1$, $c_1 = 2$, $c_t = 1$ and all other $c_j = 0$. We do this because our goal is to minimize the number of operations for efficiency, while keeping good propagation properties. Using the sum of three terms is a result of the experiments conducted in Section 8.5. The associated polynomial of θ can then be represented as

$$S \leftarrow S \cdot (1 + 2X + X^t) \bmod X^n - 1\,.$$

Now, in order to make this polynomial invertible, we should pick a suitable value t. As $n = 192$ for τ, this invertibility constraint yields 71 possible values for t :

$$\begin{aligned} t \in \{&3, 5, 9, 11, 13, 17, 19, 21, 25, 27, 29, 33, 35, 37, 41, 43, 45, 49, 51, 53, 57, 59,\\ &61, 65, 67, 69, 73, 75, 77, 81, 83, 85, 89, 91, 93, 97, 99, 101, 105, 107, 109,\\ &113, 115, 117, 121, 123, 125, 129, 131, 133, 137, 139, 141, 145, 147, 149,\\ &153, 155, 157, 161, 163, 165, 169, 171, 173, 177, 179, 181, 185, 187, 189\}\,. \end{aligned}$$

Next to this constraint, we want the parameter t to be small, but not too small for trail search, i.e., we want $t < 15$ [30, 37]. We therefore initially choose $t = 5$. This way, when looking at the two-dimensional representation, θ uses a state trit from each $y \in \{0, 1, 2\}$ and is small, but not too small.

Note that the first θ we tested was originally containing four terms:

$$S \leftarrow S \cdot (1 + X + X^2 + X^t) \bmod X^n - 1\,.$$

Here, the parameter $t = 6$ was chosen with a similar reasoning. The 46 values for which this expression is invertible are namely the following:

$$t \in \{4, 6, 12, 14, 20, 22, 28, 30, 36, 38, 44, 46, 52, 54, 60, 62, 68, 70, 76, 78, 84, 86, \\ 92, 94, 100, 102, 108, 110, 116, 118, 124, 126, 132, 134, 140, 142, 148, 150, \\ 156, 158, 164, 166, 172, 174, 180, 182, 188, 190\}.$$

As can be read in Section 8.5, this θ was deemed less appropriate for usage in the round function R.

8.4.2 Parameters r_i in ρ

As described in Section 3.3.2, there are not a lot of restrictions on the values for the r_i parameters: we only need to make sure that $r_i \neq r_j$ for $i \neq j$ holds, that $r_0 = 0$ and that r_i for $i > 0$ are non-zero. Therefore, for τ, we start off with the parameters $r_0 = 0$, $r_1 = 11$ and $r_2 = 35$.

8.4.3 Parameter g in γ

Recall that the parameter g should obey one rule as stated in Section 3.3.3. It should be coprime to $n = 192$. The possible values for g for which this condition holds are:

$$g \in \{1, 5, 7, 11, 13, 17, 19, 23, 25, 29, 31, 35, 37, 41, 43, 47, 49, 53, 55, 59, 61, 65, \\ 67, 71, 73, 77, 79, 83, 85, 89, 91, 95, 97, 101, 103, 107, 109, 113, 115, 119, \\ 121, 125, 127, 131, 133, 137, 139, 143, 145, 149, 151, 155, 157, 161, 163 \\ 167, 169, 173, 175, 179, 181, 185, 187, 191\}.$$

The value for parameter g that we initially choose in τ is 7. It is thus not bigger than 15 such that the value is considered to be small, but not too small.

Note again that γ is not invertible, but could be made invertible by adding a constraint on a state digit as explained in Section 3.3.3.

8.5 Avalanche Behavior of τ

In this section, we will study the avalanche behavior of τ and report on the worst-case values. Note that these experiments are only conducted with input differences with Hamming weight 1 with 1 as the value for the difference digit. This is because of what was explained in Section 4.2: each pair $(x, x-1)$ with difference 1 can equivalently be seen as $(x-1, x)$ with difference -1. As the order does not matter, a difference digit with value 1 is thus equivalent to those with the value $-1 \equiv 2$.

8.5.1 Mixing Layer θ With Four Terms

We start off by using the following operations and parameters in the round function $\mathrm{R} = \gamma \circ \rho \circ \theta$:

$$\begin{aligned}
&\theta\colon s_i \leftarrow s_i + s_{i+1} + s_{i+2} + s_{i+t} \text{ with } t = 6\,;\\
&\rho\colon s_i \leftarrow s_{i+3r_i \bmod 3} \text{ with } r_0 = 0, r_1 = 11 \text{ and } r_2 = 35\,;\\
&\gamma\colon s_i \leftarrow s_i + (s_{i+g})^2 \text{ with } g = 7\,.
\end{aligned}$$

The results of the avalanche behavior of τ are shown in Table 8a.

Table 8: Results of the diffusion analysis.

(a) Avalanche behavior of the mixing layer with four terms.

Round(s)	0	1	2	3	4	5	6	7
D_{av}	1	8	50	142	191	192	192	192
$\overline{w}_{\mathrm{av}}$	1.0	6.7	34.1	87.8	122.9	127.8	127.8	127.8
H_{av}	0.0	4.0	38.8	130.1	188.4	192.0	192.0	192.0

(b) Avalanche behavior of the mixing layer with three terms.

Round(s)	0	1	2	3	4	5	6	7
D_{av}	1	6	33	116	185	192	192	192
$\overline{w}_{\mathrm{av}}$	1.0	5.0	22.9	69.9	115.6	127.5	127.9	127.9
H_{av}	0.0	3.0	24.3	97.7	177.4	191.9	192.0	192.0

(c) Avalanche behavior after changing the ρ parameters.

Round(s)	0	1	2	3	4	5	6	7
D_{av}	1	6	34	126	191	192	192	192
$\overline{w}_{\mathrm{av}}$	1.0	5.0	23.6	75.1	122.9	127.8	127.9	127.9
H_{av}	0.0	3.0	25.2	107.2	185.7	192.0	192.0	192.0

8.5.2 Mixing Layer θ With Three Terms

As can be seen in Table 8a, the avalanche metrics are roughly converged after 5 rounds. Even though the results are good using this mixing layer, it uses three additions, which cost $3 \cdot 6 = 18$ bitwise operations as has been explained in Section 8.2.1. We would like to use a θ with only two additions and/or subtractions (needing only 12 bitwise operations) and see if similar results can be obtained. Therefore, we will test the following set-up regarding the

round function $\mathrm{R} = \gamma \circ \rho \circ \theta$:

$$\begin{aligned}
&\theta\colon s_i \leftarrow s_i + 2s_{i+1} + s_{i+t} \text{ with } t = 5\,;\\
&\rho\colon s_i \leftarrow s_{i+3r_i \bmod 3} \text{ with } r_0 = 0, r_1 = 11 \text{ and } r_2 = 35\,;\\
&\gamma\colon s_i \leftarrow s_i + (s_{i+g})^2 \text{ with } g = 7\,.
\end{aligned}$$

Results of the avalanche behavior using this lighter θ is shown in Table 8b.

Now, the avalanche metrics converge after roughly 6 rounds instead of after 5 rounds. Even though τ needs one round more to converge, it only requires $72+24 = 96$ bitwise operations for 6 rounds instead of $90+20 = 110$ bitwise operations for 5 rounds as the heavier θ requires. Here, the first summation terms are the bitwise operations needed for the linear layer and the second summation terms are the bitwise operations needed for the non-linear layer as elaborated upon in Section 8.2. Because of this, we prefer a mixing layer with only two additions and/or subtractions instead of a mixing layer with three additions.

Instead of using the parameter $t = 5$, other options can be considered as well. Because t should be small but not too small, we will check options below 15. This thus means that the values $t \in \{3, 5, 9, 11, 13\}$ are checked. Our initial choice, $t = 5$, performs best along with $t = 11$. The other options, $t \in \{3, 9, 13\}$ perform worse. As we prefer a smaller parameter over a larger one, the initial choice $t = 5$ is kept.

Note that other flavors of θ are also available: instead of using the polynomial $1+2X+X^t$ as before, one can also use the polynomials $1+X+2X^t$, $1 + 2X^2 + X^t$ and $1 + X^2 + 2X^t$. The polynomial $1 + X + 2X^t$ is invertible for inter alia the values $t \in \{3, 4, 5, 6, 8, 9, 11, 12, 13, 14\}$, the polynomial $1 + 2X^2 + X^t$ for e.g., $t \in \{3, 6, 7, 10, 11\}$ and $1 + X^2 + 2X^t$ is invertible for, for instance, $t \in \{6, 8, 10, 12\}$. All these values were tested. The polynomial $1 + X + 2X^t$ with parameter $t = 5$ yielded the exact same results as was obtained in Table 8b. Besides this, we could obtain similar results with four set-ups: $1 + X + 2X^t$ with parameter $t = 11$, $1 + X + 2X^t$ with parameter $t = 14$, $1+2X^2+X^t$ with parameter $t = 10$ and $1+X^2+2X^t$ with parameter $t = 10$. Since the parameters that provide a good result are either larger than $t = 5$ or exactly equal to $t = 5$, there is no reason to switch to another polynomial in the mixing layer θ.

8.5.3 Testing Different Parameters for ρ

After fixing the parameter for θ, we would like to try and improve the avalanche behavior by looking at the parameters r_1 and r_2 in ρ. Note that r_0 is still fixed to zero. A lot of combinations of different values were tested for these parameters. Notable results were, for instance, $r_1 = 11$ and $r_2 = 54$ and $r_1 = 34$ and $r_2 = 47$, but the best results were obtained with the parameter set $r_1 = 1$ and $r_2 = 24$. This is a bit counter-intuitive: intuition would

propose parameters not too close together, but apparently the combination of 0, 1 and 24 performs better. This thus means that the current best set-up for round function $\mathrm{R} = \gamma \circ \rho \circ \theta$ is:

$$\begin{aligned}
&\theta\colon s_i \leftarrow s_i + 2s_{i+1} + s_{i+t} \text{ with } t = 5\,;\\
&\rho\colon s_i \leftarrow s_{i+3r_i \bmod 3} \text{ with } r_0 = 0, r_1 = 1 \text{ and } r_2 = 24\,;\\
&\gamma\colon s_i \leftarrow s_i + (s_{i+g})^2 \text{ with } g = 7\,.
\end{aligned}$$

The avalanche behavior of this new set-up can be seen in Table 8c.

8.5.4 Testing Different Parameters for γ

Keeping the parameters t, r_0, r_1 and r_2 fixed as discussed in the previous sections, one can still tweak the parameter g of the non-linear layer γ. We tested the values $g \in \{1, 5, 7, 11, 13\}$. This resulted in finding out that $g \in \{1, 5, 13\}$ performed worse than $g = 7$ and that $g = 11$ performed similar to $g = 7$. To not increase the parameter g for minimal to no improvement, the initial parameter for γ is kept. Note that if other parameter combinations are taken for θ and ρ, g might need to change to obtain equally good results.

8.6 Round Function of τ

After investigating the avalanche behavior of τ with different parameters in the previous section, we can conclude that, from now on, we will use the parameter set $c_0 = 1, c_1 = 2, c_5 = 1, c_j = 0$ for $j \in \{0, \ldots, 191\} \setminus \{0, 1, 5\}, r_0 = 0, r_1 = 1, r_2 = 24$ and $g = 7$. The round function $\mathrm{R} = \gamma \circ \rho \circ \theta$ with its final parameters can thus be noted down as:

$$\begin{aligned}
&\theta\colon s_i \leftarrow s_i + 2s_{i+1} + s_{i+5}\,;\\
&\rho\colon s_i \leftarrow s_{i+3r_i \bmod 3} \text{ with } r_0 = 0, r_1 = 1 \text{ and } r_2 = 24\,;\\
&\gamma\colon s_i \leftarrow s_i + (s_{i+7})^2\,.
\end{aligned}$$

8.7 Differential and Linear Propagation Properties of τ

The tools as provided in Section 5.6 and Section 6.7 can be used to find the differential and linear propagation properties of τ. Recall that the results are linked to the parameter $g = 1$ in the non-linear layer. One should thus not forget that τ uses the parameter $g = 7$ in γ and that because of this multiplicative shuffles are needed.

The tools as provided in the Algorithms 2, 3 and 4 are available in `Python` for τ at `https://github.com/DVerbakel/MasterThesis`.

8.8 Implementation of τ

In this section, we will explain how we implemented τ in code (in `C`). The software implementation can be found at `https://github.com/DVerbakel/MasterThesis`.

Having encoding 2, i.e., $\{0 \mapsto 00, 1 \mapsto 10, 2 \mapsto 11\}$, our implementation will represent the 192-trit state as six 64-bit words. The 64-bit words are used as the number of columns equals 64, which results in no left-over space in a word. Besides this, instead of having 3 rows with trits, 6 rows should be used for the bit encoding, because each trit is encoded using two bits.

Just like the original state, columns are numbered from left to right (from 0 to 63). However, the rows of the implementation state are numbered in the reverse direction: they are numbered from top to bottom (from 0 to 5).

To convert a 3×64 state to the 6×64 state in the implementation, one should perform the following steps for each element i of row j where $i \in \{0, \ldots, 63\}$ and $j \in \{0, 1, 2\}$:

1. If a 0 is encountered on position i in row j, then:
   a. 0 should be written on position i in row $4 - 2 \cdot j$;
   b. 0 should be written on position i in row $5 - 2 \cdot j$.
2. If a 1 is encountered on position i in row j, then:
   a. 1 should be written on position i in row $4 - 2 \cdot j$;
   b. 0 should be written on position i in row $5 - 2 \cdot j$.
3. If a 2 is encountered on position i in row j, then:
   a. 1 should be written on position i in row $4 - 2 \cdot j$;
   b. 1 should be written on position i in row $5 - 2 \cdot j$.

The conversion to the new state representation is shown in Figure 19.

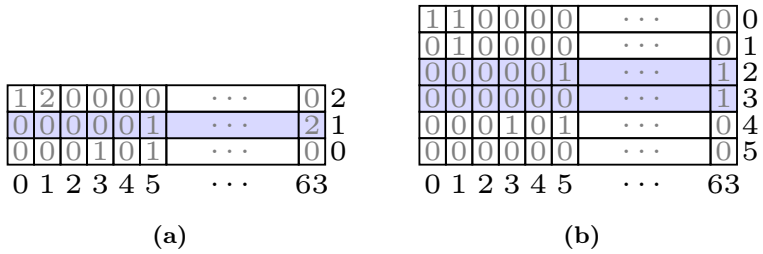


Figure 19: (a) Example state s for τ. (b) Implementation of the example state s for τ in code using encoding 2.

If we want to apply our round function to the implementation state, we should keep in mind that trits are encoded using two bits.

For the mixing layer θ this means that we should first do subtraction as defined in Section 8.2.2 and after this perform an addition as defined in Section 8.2.1. A visualization of this step is shown in Figure 20.

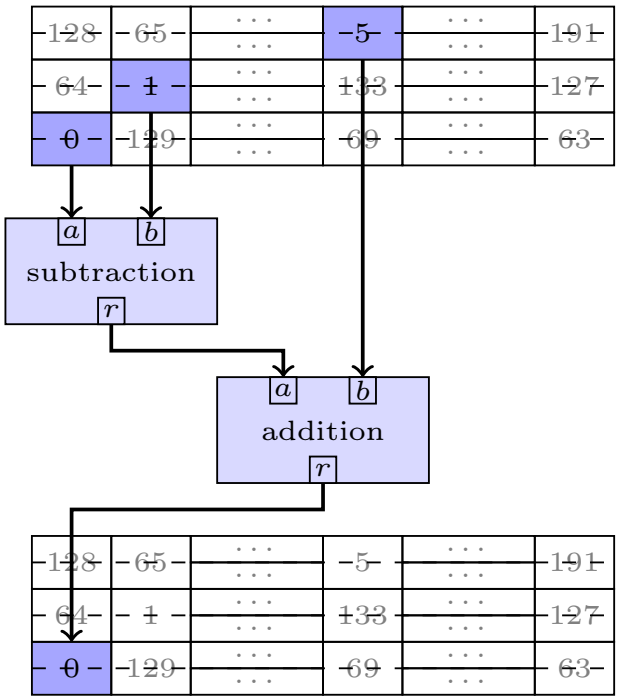


Figure 20: A schematic overview of the θ operation on the implementation state. Here, it is shown that the state digit s_0 is adapted by θ resulting in the state digit $\theta(s_0)$. In the figure the state digits s_i are denoted as i. Note that the parameter $t = 5$ is used and that a, b and r are the inputs and outputs of subtraction and addition as defined in respectively Section 8.2.2 and Section 8.2.1.

The shuffle layer ρ does not change a lot because of this representation: instead of shifting row 1 with r_1 and row 2 with r_2, we shift rows 0 and 1 with r_2 and rows 2 and 3 with r_1: see Figure 21. Again, note that the rows are numbered in the reverse direction in our implementation.

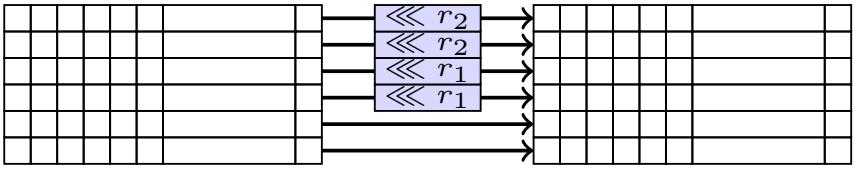


Figure 21: A schematic overview of the ρ operation on the implementation state. Here, it is shown that row 4 and 5 are not shifted, that row 2 and 3 are shifted with r_1 and that row 0 and 1 are shifted with r_2.

For the last step function, the non-linear layer γ, we simply perform addition of a square as defined in Section 8.2.5. This operation is shown in Figure 22.

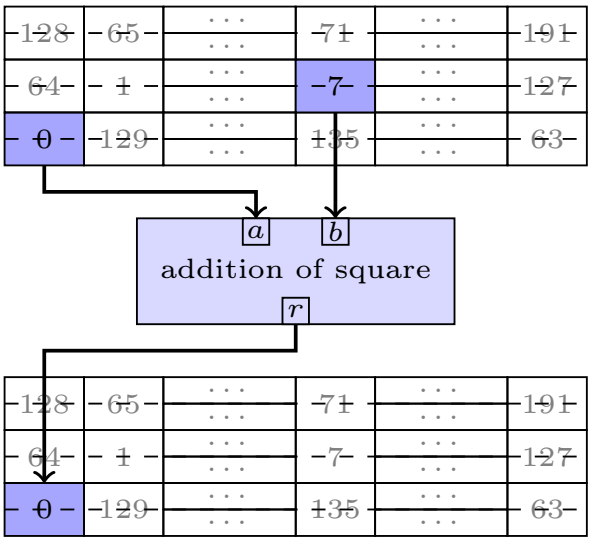


Figure 22: A schematic overview of the γ operation on the implementation state. Here, it is shown that the state digit s_0 is adapted by γ resulting in the state digit $\gamma(s_0)$. In the figure the state digits s_i are denoted as i. Note that the parameter $g = 7$ is used and that a, b and r are the inputs and output of addition of a square as defined in Section 8.2.5.

Chapter 9

Conclusions and Future Work

In this thesis, we have investigated how to design an efficient q-ary transformation with good propagation properties. We did this by first defining the family of q-ary transformations $\mathcal{T}_q$. After this, we generalized the concepts of avalanche behavior and differential cryptanalysis from the binary case to be applicable to q-ary transformations. For the linear cryptanalysis, we made a generalization of binary linear cryptanalysis to p-ary transformations. Having defined these, we looked into the differential and linear propagation properties of the non-linear layer $\gamma\colon s_i \leftarrow s_i + (s_{i+g})^2$ of $\mathcal{T}_q$. We found that there exists a dual relation of the DP of γ and the LP of γ, namely $\mathrm{DP}_\gamma(b,a) = q^{-\mathrm{HW}(b)}$ and $\mathrm{LP}_\gamma(u,v) = p^{-\mathrm{HW}(v)}$. We also found a dual relation of the DP and LP related to squaring: $\mathrm{DP}_s(b,a) = \frac{1}{q}$ and $\mathrm{LP}_s(u,v) = \frac{1}{p}$. The consequence of this similarity is that masks propagate as differences over γ. Concretely, with $f_{-1}\colon \mathbb{F}_p^n \to \mathbb{F}_p^n$ defined as $f_{-1}(x) = y$ where $\forall i\colon y_{-i} = x_i$, we could find that for $v = f_{-1}(b)$ and $u = f_{-1}(a)$ it holds that $\mathrm{DP}_\gamma(b,a) = \mathrm{LP}_\gamma(u,v)$. Such dual relations are for a non-linear mapping an intriguing property that is believed to only occur in our γ and the mapping χ_3 as used in Xoodoo. Additionally, we also provided tools in order to perform differential and linear trail search on $\mathcal{T}_q$. These tools are for finding the minimum reverse weight, showing that the Hamming weight represents the minimum direct weight and how to find compatible input and output differences and/or masks. Besides this, we looked into possible practical applications of $\mathcal{T}_q$. Last, but not least, we provided a case study of transformation τ in $\mathcal{T}_3$. Notable was the efficient software implementation with dedicated arithmetic using the encoding $\{0 \mapsto 00, 1 \mapsto 10, 2 \mapsto 11\}$ and the results of the investigation into the avalanche behavior. These avalanche tests were used to choose concrete parameter values of the steps mappings. From these experiments we found that the rotation offsets $r_0 = 0, r_1 = 1$ and $r_2 = 24$ of the shuffle layer ρ performed better than more spread out values, i.e., values that are not too close together, as intuition would propose.

In this research, we left some parts as future work: these are other interesting aspects to analyze. First of all, one could generalize the linear cryptanalysis to q-ary transformations. One could also perform the actual (differential and linear) trail search on $\mathcal{T}_q$ for which we provided the tools. Another possibility would be to do more research regarding an efficient hardware and software implementation of an instance of $\mathcal{T}_q$. This could be done by following a similar reasoning as in Chapter 8, finding an efficient way on how to do addition of three terms or perhaps by writing optimized assembly code for, for example, an Arm Cortex-M4 processor. Alternatively, research could be done to investigate the effects of adding a fourth step function, namely a round constant addition. One could also define concrete instances for the practical applications like the sponge construction, duplex construction, encryption scheme Ciminion or authenticated encryption scheme Elephant. Lastly, one could investigate algebraic attacks over $\mathbb{F}_q$ and look into the algebraic degree of $\mathcal{T}_q$.

Bibliography

[1] Martin Albrecht, Lorenzo Grassi, Christian Rechberger, Arnab Roy, and Tyge Tiessen. MiMC: Efficient Encryption and Cryptographic Hashing with Minimal Multiplicative Complexity. Cryptology ePrint Archive, Paper 2016/492, 2016. `https://eprint.iacr.org/2016/492`.

[2] Thanos Antoulas, Richard Baraniuk, Steven Cox, Benjamin Fite, Roy Ha, Michael Haag, Matthew Hutchinson, Don Johnson, Ricardo Radaelli-Sanchez, Justin Romberg, Phil Schniter, Melissa Selik, and JP Slavinsky. *Signals and Systems*. Rice University, Houston, Texas, USA, 2008.

[3] Steven Arno and Ferrell S. Wheeler. Signed digit representations of minimal Hamming weight. *IEEE Transactions on Computers*, 42(8):1007–1010, 1993.

[4] Thomas Baignères, Jacques Stern, and Serge Vaudenay. Linear Cryptanalysis of Non Binary Ciphers. In Carlisle Adams, Ali Miri, and Michael Wiener, editors, *Selected Areas in Cryptography*, pages 184–211. Springer Berlin Heidelberg, Heidelberg, Germany, 2007.

[5] Paul Bamberg and Shlomo Sternberg. *A course in mathematics for students of physics*. Press Syndicate of the University of Cambridge, The Pitt Building, Trumpington Street, Cambridge CB2 1RP, 1988.

[6] Guido Bertoni, Joan Daemen, Seth Hoffert, Michaël Peeters, Gilles van Assche, and Ronny van Keer. The sponge and duplex constructions. `https://keccak.team/sponge_duplex.html`, 2022. Last accessed: April 18, 2023.

[7] Guido Bertoni, Joan Daemen, Michaël Peeters, and Gilles van Assche. Cryptographic sponge functions. `https://keccak.team/files/CSF-0.1.pdf`, January 2021. Last accessed: June 27, 2023.

[8] Tim Beyne, Yu Long Chen, Christoph Dobraunig, and Bart Mennink. Elephant. `https://www.esat.kuleuven.be/cosic/elephant/`, May 2021. Last accessed: May 30, 2023.

[9] Tim Beyne, Yu Long Chen, Christoph Dobraunig, and Bart Mennink. Elephant v2. https://csrc.nist.gov/CSRC/media/Projects/lightweight-cryptography/documents/finalist-round/updated-spec-doc/elephant-spec-final.pdf, May 2021. Last accessed: May 30, 2023.

[10] Eli Biham and Adi Shamir. Differential Cryptanalysis of DES-like Cryptosystems. In Menezes A.J. and Vanstone S.A., editors, *Advances in Cryptology-CRYPT0' 90*, pages 2–21. Springer Berlin Heidelberg, Heidelberg, Germany, 1991.

[11] Alex Biryukov and Christophe Cannière. Linear Cryptanalysis for Block Ciphers. In Henk C. A. van Tilborg, editor, *Encyclopedia of Cryptography and Security*, pages 351–354. Springer US, Boston, MA, USA, 2005.

[12] Céline Blondeau and Benoît Gérard. Multiple Differential Cryptanalysis: Theory and Practice. In Joux A., editor, *Fast Software Encryption*, pages 35–54. Springer Berlin Heidelberg, Heidelberg, Germany, 2011.

[13] Tomas Boothby and Robert W. Bradshaw. Bitslicing and the Method of Four Russians Over Larger Finite Fields. *ArXiv*, abs/0901.1413, 2009.

[14] Nicolas Bordes, Joan Daemen, Daniël Kuijsters, and Gilles Van Assche. Thinking Outside the Superbox. Cryptology ePrint Archive, Paper 2021/293, 2021. https://eprint.iacr.org/2021/293.

[15] Christina Boura, Margot Funk, and Yann Rotella. Differential analysis of the ternary hash function Troika. Cryptology ePrint Archive, Paper 2023/036, 2023. https://eprint.iacr.org/2023/036.

[16] David M. Burton. *The History of Mathematics: An Introduction.* McGraw-Hill Publishing Company, New York, NY, USA, seventh edition, 2011.

[17] Jeff Connelly, C Patel, A Chavez, and P Nico. *Ternary Computing Testbed: 3-Trit Computer Architecture.* PhD thesis, California Polytechnic State University, 1 Grand Ave, San Luis Obispo, CA 93407, USA, 2008. Computer Engineering Department.

[18] Joan Daemen. *Cipher and Hash Function Design Strategies based on linear and differential cryptanalysis.* PhD thesis, KU Leuven, Leuven, Belgium, 1995. https://cs.ru.nl/~joan/papers/JDA_Thesis_1995.pdf.

[19] Joan Daemen. Xoodoo, Xoofff and differential propagation, September 2022. Lecture slides Cryptology, Autumn 2022, Institute for Computing and Information Sciences Radboud University.

[20] Joan Daemen and Gilles Van Assche. Differential Propagation Analysis of Keccak. In Anne Canteaut, editor, *Fast Software Encryption - 19th International Workshop. Revised Selected Papers*, volume 7549 of *Lecture Notes in Computer Science*, pages 422–441. Springer, March 2012.

[21] Joan Daemen, Seth Hoffert, Gilles Van Assche, and Ronny Van Keer. The design of Xoodoo and Xoofff. *IACR Transactions on Symmetric Cryptology*, 2018(4):1–38, December 2018.

[22] Joan Daemen, Bart Mennink, and Jan Schoone. Introduction to Cryptography Lecture Notes 2022, August 2022. Additional Lecture material Introduction to Cryptography, Applied Cryptography and Cryptology, Summer 2022, Institute for Computing and Information Sciences Radboud University.

[23] Joan Daemen and Vincent Rijmen. Correlation Analysis in GF(2^n). In *The Design of Rijndael: The Advanced Encryption Standard (AES)*, pages 181–194. Springer Berlin Heidelberg, Heidelberg, Germany, 2020.

[24] Cunsheng Ding, Dingyi Pei, and Arto Salomaa. *CHINESE REMAINDER THEOREM: Applications in Computing, Coding, Cryptography*. World Scientific, October 1996.

[25] Christoph Dobraunig, Lorenzo Grassi, Anna Guinet, and Daniël Kuijsters. Ciminion: Symmetric Encryption Based on Toffoli-Gates over Large Finite Fields. Cryptology ePrint Archive, Paper 2021/267, 2021. `https://eprint.iacr.org/2021/267`.

[26] Stephen H. Friedberg, Arnold J. Insel, and Lawrence E. Spence. *Linear Algebra*. Pearson Education, London, UK, 2013.

[27] Lorenzo Grassi. Bounded surjective quadratic functions over $\mathfrak{V}_p^n$ for mpc-/zk-/fhe-friendly symmetric primitives. Cryptology ePrint Archive, Paper 2022/1313, 2022. `https://eprint.iacr.org/2022/1313`.

[28] Lorenzo Grassi, Dmitry Khovratovich, Christian Rechberger, Arnab Roy, and Markus Schofnegger. Poseidon: A New Hash Function for Zero-Knowledge Proof Systems. In *USENIX Security Symposium*, 2021.

[29] Jim Hefferon. *Linear Algebra*. Orthogonal Publishing L3C, 2020. Fourth edition.

[30] Solane El Hirch, Silvia Mella, Alireza Mehrdad, and Joan Daemen. Improved Differential and Linear Trail Bounds for ASCON. Cryptology ePrint Archive, Paper 2022/1377, 2022. `https://eprint.iacr.org/2022/1377`.

[31] Yuto Kawahara, Kazumaro Aoki, and Tsuyoshi Takagi. Faster Implementation of η_T Pairing over GF(3^m) Using Minimum Number of Logical Instructions for GF(3)-Addition. In Steven D. Galbraith and Kenneth G. Paterson, editors, *Pairing-Based Cryptography – Pairing 2008*, pages 282–296. Springer Berlin Heidelberg, Berlin, Heidelberg, 2008.

[32] Stefan Kölbl, Elmar Tischhauser, Patrick Derbez, and Andrey Bogdanov. Troika: a ternary cryptographic hash function. *Designs, Codes and Cryptography*, 88(1):91–117, August 2019.

[33] Dexter Kozen and Marc Timme. Indefinite Summation and the Kronecker Delta. *Computing and Information Science Technical Reports*, October 2007.

[34] Rudolf Lidl and Harald Niederreiter. *Introduction to finite fields and their applications*. Cambridge University Press, 1986. (Reprinted 1988).

[35] Rudolf Lidl and Harald Niederreiter. *Finite Fields*, volume 20. Cambridge University Press, 1997. Second edition. (Reprinted 2000).

[36] Mitsuru Matsui. Linear Cryptanalysis Method for DES Cipher. In Helleseth T., editor, *Advances in Cryptology - EUROCRYPT '93*, pages 386–397. Springer Berlin Heidelberg, Heidelberg, Germany, 1994.

[37] Alireza Mehrdad, Silvia Mella, Lorenzo Grassi, and Joan Daemen. Differential Trail Search in Cryptographic Primitives with Big-Circle Chi: Application to Subterranean. *IACR Transactions on Symmetric Cryptology*, 2022(2):253–288, June 2022.

[38] Silvia Mella. Trail search in permutations, October 2022. Lecture slides Cryptology, Autumn 2022, Institute for Computing and Information Sciences Radboud University.

[39] Silvia Mella, Joan Daemen, and Gilles Van Assche. New techniques for trail bounds and application to differential trails in Keccak. *IACR Transactions on Symmetric Cryptology*, 2017(1):329–357, March 2017.

[40] Bart Mennink. Keyed Sponges, March 2022. Lecture slides Applied Cryptography, Spring 2022, Institute for Computing and Information Sciences Radboud University.

[41] Theodosis Mourouzis. *Optimizations in Algebraic and Differential Cryptanalysis*. PhD thesis, University College London, Gower Street, London, WC1E 6BT, 2015. Department of Computer Science.

[42] Tristan Needham. *Visual Complex Analysis*. Oxford University Press, New York, USA, 1997.

[43] NIST. Contacts: René Peralta, Meltem Sönmez Turan, Luís T.A.N. Brandão. Circuit Complexity. `https://csrc.nist.gov/Projects/circuit-complexity`, February 2023. Last accessed: April 18, 2023.

[44] Kenneth H. Rosen. *Discrete Mathematics and Its Applications*. McGraw-Hill Publishing Company, New York, NY, USA, seventh edition, 2012.

[45] Thierry Simon, Lejla Batina, Joan Daemen, Vincent Grosso, Pedro Maat Costa Massolino, Kostas Papagiannopoulos, Francesco Regazzoni, and Niels Samwel. Friet: An Authenticated Encryption Scheme with Built-in Fault Detection. In Anne Canteaut and Yuval Ishai, editors, *Advances in Cryptology – EUROCRYPT 2020*, pages 581–611, New York City, USA, 2020. Springer International Publishing.

[46] Murray R. Spiegel, John J. Schiller, and R. Alu Srinivasan. *Schaum's Outlines of Probability and Statistics*. McGraw-Hill Publishing Company, New York, NY, USA, fourth edition, 2013.

[47] Michael Spivak. *Calculus*. Publish or Perish, Inc., third edition, 1994.

[48] Pang-Ning Tan, Michael Steinbach, and Vipin Kumar. *Introduction to Data Mining*. Pearson Education, London, UK, 2006.

[49] A. F. Webster and S. E. Tavares. On the Design of S-Boxes. In H.C. Williams, editor, *Lecture Notes in Computer Science*, Advances in Cryptology - CRYPTO '85, LNCS 218, pages 523–534, Berlin, DE, 1986. Springer-Verlag Berlin Heidelberg.

[50] Tolga Yalçın and Elif Bilge Kavun. On the Implementation Aspects of Sponge-Based Authenticated Encryption for Pervasive Devices. In *Smart Card Research and Advanced Applications*, pages 141–157. Springer Berlin Heidelberg, 2013.