FACULTY
OF MATHEMATICS
AND PHYSICS
Charles University

MASTER THESIS

Marie Brožová

Projective polynomials and the S-Boxes of Streebog and Kuznyechik

Department of Algebra

Supervisor of the master thesis:	Dr. rer. nat. Faruk Göloglu
Study programme:	Matematics for Information Technologies
Study branch:	Matematics for Information Technologies

Prague 2024

I would like to thank Dr. rer. nat. Faruk Göloglu, my supervisor, who identified the topic of Projective polynomials and the S-Boxes of Streebog and Kuznyechik and led me through it. I am grateful for every advise, explanation and comment during consultations. I appreciate also the flexibility around the arrangement of consultations. I would also like to thank my family for supporting me, especially my husband and my son who gave me space for self-realisation.

Title: Projective polynomials and the S-Boxes of Streebog and Kuznyechik

Author: Marie Brožová

Department: Department of Algebra

Supervisor: Dr. rer. nat. Faruk Göloglu, Department of Algebra

Abstract: This thesis deals with possible connection between result of Léo Perrin's paper published at 2019 and Gologlu's paper published at 2022. Perrin deals with S-box π, which is a part of Kuznyechik cipher and hash function Streebog, especially he introduces the notion of TKlog, which is a structure found in the S-box. The main result of Gologlu's paper is classification of fractional q-projective functions, which seems similar to the structure of TKlog. In the thesis, there is a description of cipher Kuznyechik as well as the hash function Streebog. Then, results of Perrin's paper are described including cryptographical properties of the S-box. As a main contribution we give a design of an experiment with a goal to find a fractional q-projective function with same or similar invariants as the S-box π. Such function can be an initial point in a design of an attack.

Keywords: TKlog fractional polynomial projective polynomial Kuznyechik Streebog

Contents

Introduction

Cryptography, as we know it today, follows principles invented more than a hundred years ago. Today, there exist multiple security institutes around the world that publish cryptographical primitives. Published primitives are examined and tested by a team of specialists. One of them is NSA (National Security Agency) besides United States Department of Defense that published now most commonly used primitives.

Rosstandard, formerly Gosstandard, is Russian federal government agency subordinated to the Ministry of Industry and Trade. In 2015 the agency published a standard that introduced a block cipher Kuznyechik and a hash function Streebog. The cipher as well as the hash function uses an S-box called π as a non-linear part of the procedure of encryption. Original standard is not distributed freely to the public, but [DD13] and [Dol16] describe the block cipher and the hash function as well.

Since that time, several papers were published [PU16], [BPU16] and [Per19], that analyse the published standard and point out on found properties of its S-box. Published papers and especially the most recent paper [Per19] by Perrin found that there is a strong structure in the S-box and name it TKlog. It is a special property of the S-box that preserves structural properties of its input to its output. S-box π is a permutation of the finite field $\mathbb{F}_{2^8}$. Roughly speaking, the special property of S-box π is quite related to its subfield of index 2 $\mathbb{F}_{2^4}$. There is an apparent connection between the structure found in the S-box π and functions introduced in [Gö22]. This connection seems to stem from the relationship of trace and norm functions to both the S-Box π and projective polynomials.

We design an experiment where we inspect a set of fractional q-projective function. We make use of the result of [Gö22], the classification of fractional q-projective function. It gives us a tool how to generate all fractional q-projective function by only composition of one or two fixed forms with all projective linear transformations. A direct way how to inspect all of them takes more than reasonable time, therefore we have to reduce the number of function we will check. We use invariants with respect to affine-equivalency to overcome the complexity of the experiment, by inspection of one single representative of each affine-equivalency class.

We organise the thesis by chapters. In Chapter 1 we give all necessary backgroud for following chapters. Specifically we give definitions and notations of structures and mappings we will deal with such as finite fields, Boolean functions, affine and projective space and fractional polynomials that operate on the projective space. Moreover we give basic introduction into cryptosystems such as ciphers and hash function, that will be needed in Chapter 2. First, we describe there the cipher Kuznyechik and the hash function Streebog themselves and then we deal with results of previous papers [PU16] and [BPU16] that give us decompositions of the S-box. In another section we describe properties found in [Per19]. Last chapter (Chapter 3) contains description of fractional q-projective function and the result of [Gö22], which will help us in our experiment described in Section 3.4.

1. Preliminaries

Mappings and algorithms for cryptographical systems used to secure and store an information are based on mathematical structures. Analysis of the principles are usually based on general theory and structures of abstract algebra. We therefore start with a chapter that covers all necessary terms.

First, we list notation around used structures and mappings between them. Then, we give foundations of Boolean functions, within the range we will use them later in the thesis. We continue with description of types of cryptosystems we will analyse and of their properties used in linear and differential cryptanalysis, that plays a role in every analysis of an S-box. At the end of the chapter as last section we give elementary definitions from algebraic geometry with focus on projective permutations over finite field.

1.1 Structures and mappings

Finite fields and vector spaces are important structures in domain of algebra as well as in cryptanalysis. Together with integers they cover all structures we will be interested in. There exists a connection between them and almost a one-to-one correspondence between their elements. We give a list of mathematical structures and the notation we will use from now on in Notation 1.

Notation 1. *Let $N, m, k \in \mathbb{N}$, $N = p^m$ for p a prime then:*

- *$\mathbb{F}_N$ is the field with N elements,*
- *F^k is k dimensional vector space over a field F with standard basis $\{e_i\}_{i=0}^{k-1}$ and notation*

$$\overline{a} \in F^k \iff \overline{a} = (a_0, \dots, a_{k-1}) \text{ where } a_i \in F \ \forall i : 0 \leq i < k,$$

- *$\underbrace{F \times \cdots \times F}_{k \text{ times}}$ is set of all possible ordered k-tuples of elements from F.*

We focus in the thesis on analysis of one particular part of a cryptographical system that operates on structures with $p = 2$ and therefore we will be mostly interested in following structures:

- finite fields $\mathbb{F}_{2^m}$ and $\mathbb{F}_{2^{2m}}$ as structures that plays the main role in Chapter 2 and especially then with $m = 4$, as our mapping of interest is defined to permute $\mathbb{F}_{2^8}$,
- Boolean vector spaces $\mathbb{F}_2^m$ and $\mathbb{F}_2^{2m}$ as structures the most close to real configuration of bits in a computer and especially with $m = 4$, because $\mathbb{F}_2^8$ and $\mathbb{F}_2^4$ are the vector spaces that correspond to the finite fields from previous point.

Remark. Finite field $\mathbb{F}_{2^m}$ has structure of a factor ring of a polynomial ring $\mathbb{F}_2[x]$

$$\mathbb{F}_{2^m} = \mathbb{F}_2[x]/(g(x)) = \Big\{ \sum_{i=0}^{m-1} z_i \alpha^i \Big\}$$

for $g \in \mathbb{F}_2[x]$ irreducible of degree m, α a root of g and $\overline{z} = (z_0, \dots, z_{m-1}) \in \mathbb{F}_2^m$.

In every field there must exists an inverse of every nonzero element and therefore we require an irreducible polynomial g in such construction. Irreducible polynomial generates a prime ideal, which will guarantee the field. Representation of elements from the finite field is covered by all polynomials of degree less than m.

Let α be a root of g, which is not in $\mathbb{F}_2$ obviously, because g is an irreducible polynomial over $\mathbb{Z}_2$. The element α that also belongs to the residue class of x then ensures that every elements from $\mathbb{F}_{2^m}$ can be represented as $\sum_{i=0}^{m-1} z_i\alpha^i \bmod g(\alpha)$. The finite field is therefore generated by the element α or in other words by the polynomial bases $\{1, \alpha, \ldots, \alpha^{m-1}\}$.

For a construction of a finite field with 2^m elements we can chose an arbitrary root of f because there exists an isomorphism between two finite fields generated by two different root of an irreducible polynomial of degree m. Moreover we can chose any irreducible polynomial of degree m and its roots will always generate a finite fields isomorphic to each other.

Example. The field $\mathbb{F}_{2^{2m}}$ with $m = 4$ used in π is defined as

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

Every field contains groups with respect to multiplication and addition operation. Notation 2 and Observation 1 describe structure of multiplicative and additive group of a field, which will be used in Section 2.3 to explain discoveries of Perrin [Per19].

Notation 2. *The largest group of the finite field $\mathbb{F}_{2^m}$ with respect to multiplication and addition operation will be called multiplicative and additive group respectively and they will be denoted as:*

- *Multiplicative group $(\mathbb{F}_{2^m}^\times, \cdot)$ and*
- *Additive group $(\mathbb{F}_{2^m}, +)$*

with notation $\mathbb{F}_{2^m}^\times = \mathbb{F}_{2^m} \setminus \{0\}$.

Observation 1. *Let α be an element of $\mathbb{F}_{2^m}$ as in Remark after Notation 1. Then using Notation 2 we have that:*

- $(\mathbb{F}_{2^m}^\times, \cdot) \simeq (\mathbb{Z}_{2^m-1}, +)$ *is a cyclic group generated by element α with underlying set*
$$\mathbb{F}_{2^m}^\times = \{\alpha^i \ mod\ p | i \in \mathbb{Z}_{2^m-1}\},$$
with correspondence $\alpha^i + \alpha^j = \alpha^{i+j}$ for any $i, j \in \mathbb{Z}_{2^m-1}$ and $\alpha^{-1} = \alpha^{2^m-2}$ since $\alpha \cdot \alpha^{2^m-2} = \alpha^{2^m-1} = 1$,

- $(\mathbb{F}_{2^m}, +) \simeq (\mathbb{F}_2, +)^m$ *is a vector space over $\mathbb{F}_2$ of dimension d with standard basis $\{e_i\}_{i=0}^m$, with underlying set*
$$\mathbb{F}_{2^m} = \{\sum_{i=0}^{d-1} a_i\alpha^i | (a_0, \ldots, a_{m-1}) \in \mathbb{F}_2^m\},$$
with $\{1, \alpha^1, \ldots, \alpha^{m-1}\}$ the polynomial basis.

Remark. Observation 1 describe structures of finite fields where we can see that its additive group is isomorphic to a vector space. If we chose a different root of g in the construction of the finite field e.g., β, we get also an isomorphism to a vector space but elements themselves will be represented by different sequences of $(a_0, \dots, a_{m-1})$ analogously as if we chose a different basis of the vector space.

As we already mentioned, cryptographical systems runs over certain mathematical structures. The closest representation of a message or information to be protected is by elements of a vector spaces over $\mathbb{F}_2$, because the data are usually bitstrings of known length. Before we move on to transitions between structures, we focus on operations in vector spaces.

During manipulations with bitstrings as encryption, key generation or message consummation, several bit operations are needed. We represent bitstrings as elements from $\mathbb{F}_2^k$ and therefore bit operations will be represented as operation over $\mathbb{F}_2^k$. We cover all used operations by Notation 3 as a list of simple and well known functions to introduce them and use later in the thesis.

Notation 3.

- $\oplus$ *is binary operator* $\mathbb{F}_2^k \times \mathbb{F}_2^k \to \mathbb{F}_2^k$ *that works as addition in* $\mathbb{F}_2$ *by components with infix notation*

$$\overline{a} \oplus \overline{b} = \overline{c}, \ \textit{where } c_i = a_i + b_i, \forall i : 0 \le i < k,$$

- $\|$ *is binary operator* $\mathbb{F}_2^k \times \mathbb{F}_2^k \to \mathbb{F}_2^{2k}$ *that works as concatenation of two elements with infix notation*

$$\overline{a} \| \overline{b} = \overline{c}, \ \textit{where } c_i = a_i \textit{ and } c_{k+i} = b_i, \forall i : 0 \le i < k,$$

- *generalized* $\|$ *operator is n-ary operator* $\overbrace{\mathbb{F}_2^k \times \cdots \times \mathbb{F}_2^k}^{n \textit{ times}} \to \mathbb{F}_2^{nk}$ *that works as concatenation of n elements with infix notation*

$$\overline{a}^{(0)} \| \dots \| \overline{a}^{(n-1)} = \overline{a}, \ \textit{where } a_{ik+j} = \overline{a}_j^{(i)}, \forall i : 0 \le i < n \textit{ and } \forall j : 0 \le j < k,$$

- $\mathrm{Split}_n : \mathbb{F}_2^{nk} \to \overbrace{\mathbb{F}_2^k \times \cdots \times \mathbb{F}_2^k}^{n \textit{ times}}$, *that works as*

$$\mathrm{Split}_n(\overline{a}) = \overline{a}_0, \dots, \overline{a}_{n-1}$$

  for $\overline{a}^{(0)} \| \dots \| \overline{a}^{(n-1)} = \overline{a}$ *and* $\overline{a}^{(0)}, \dots, \overline{a}^{(n-1)} \in \mathbb{F}_2^k$, $\overline{a} \in \mathbb{F}_2^{nk}$,

- $\mathrm{MSB}_n : \mathbb{F}_2^k \longrightarrow \mathbb{F}_2^n$ *with* $k > n$ *that works as orthogonal projection to* $\mathbb{F}_2^n$ *with basis* $\{e_{k-n}, \dots, e_{k-1}\}$

$$\mathrm{MSB}_n(\overline{x}) = (x_{k-n}, \dots, x_{k-1})$$

  i.e. takes n most significant bits,

- $\mathrm{LSB}_n : \mathbb{F}_2^k \longrightarrow \mathbb{F}_2^n$ *with* $k > n$ *that works as orthogonal projection to* $\mathbb{F}_2^n$ *with basis* $\{e_0, \dots, e_{n-1}\}$

$$\mathrm{LSB}_n(\overline{x}) = (x_0, \dots, x_{n-1})$$

  i.e. takes n less significant bits.

Bitstring of fixed lenght can be interpreted in more than one way. We can always talk about k bit information but the way we interpret it gives us the real meaning and possibilities how to manipulate it.

Cryptographical systems can for instance change the interpretation in the middle of computation as it is for example in well known cipher AES, where we switch between interpretation of a bitstring as an element from vector space $\mathbb{F}_2^k$ and a finite field $\mathbb{F}_{2^k}$.

There is also another way how to interpret a bitsting and that is interpretation as an integer. We will see in Observation 2, that there exists a mapping that connects integers and other representation of a bitstring but the mapping does not preserve the structure. It is the reason way interpretation as integers is usually used only to express the value table of a function that operates on vector spaces. We list mappings that connect mentioned structures in Notation 4.

Notation 4. *Let $k \in \mathbb{N}$.*

- *Transition from integer representation to vector representation will be denoted as $\mathrm{V}_k : \mathbb{Z}_{2^k} \longrightarrow \mathbb{F}_2^k$ with*

  $$\mathrm{V}_k(a) := (a_0, \dots, a_{k-1}) = \overline{a},$$

  such that $a = \sum_{i=0}^{k-1} a_i 2^i$ and $a_i \in \mathbb{F}_2$.

- *Transition from vector representation to integer representation will be denoted as $\mathrm{Int}_k : \mathbb{F}_2^k \longrightarrow \mathbb{Z}_{2^k}$ with*

  $$\mathrm{Int}_k(\overline{a}) := \sum_{i=0}^{k-1} a_i 2^i,$$

  such that $\overline{a} = (a_0, \dots, a_{k-1})$.

- *Transition from the finite field representation to vector representation will be denoted as $\mathrm{V}_F : \mathbb{F}_{2^m} \longrightarrow \mathbb{F}_2^m$ with*

  $$\mathrm{V}_F(z) := (z_0, \dots, z_{d-1}),$$

  such that $z = \sum_{i=0}^{d-1} z_i \alpha^i$.

- *Transition from the vector representation to finite field representation will be denoted as $\mathrm{F}_\mathrm{V} : \mathbb{F}_2^m \longrightarrow \mathbb{F}_{2^m}$ with*

  $$\mathrm{F}_\mathrm{V}(\overline{z}) = \sum_{i=0}^{m-1} z_i \alpha^i,$$

  such that $\overline{z} = (z_0, \dots, z_{d-1})$.

To summarize, we depict all mappings from Notation 4 in Figure 1.1 with $k = d$.

$$\begin{tikzcd}
\mathbb{Z}_{2^m} \arrow[r, shift left, "\mathrm{V}_m"] & \mathbb{F}_2^m \arrow[l, shift left, "\mathrm{Int}_m"] \arrow[r, shift left, "\mathrm{F}_\mathrm{V}"] & \mathbb{F}_{2^m} \arrow[l, shift left, "\mathrm{V}_F"]
\end{tikzcd}$$

Figure 1.1: Transitions between main structures

We describe connection between mappings from Notation 4 in Observation 2. All of the mappings has different domains and images, however they are all bijections and we can observe several similarities between them. First we can see that mappings Int_k and V_k are inverse to each other as well as F_V and V_F. Then we also observe that Int_k is somehow an analogy of F_V if we chose $\alpha = 2$ and V_k is analogy for F_V for $\alpha = 2$. We list up all observed properties to complete the concept of transitions between studied structures by those mappings.

Observation 2. *We can derive simple observations about mappings from Notation 4:*

1. Int_k *and* V_k *are both injective mapping since* $a_i < 2$ *and remainders are always unique.*
2. $\mathrm{Int}_k = \mathrm{V}_k^{-1}$ *because of point 1. and*
$$\mathrm{Int}_k(a) = (a_0, \ldots, a_{k-1}) \iff \mathrm{V}_k(a_0, \ldots, a_{k-1}) = a$$
for every $a \in \mathbb{Z}_{2^k}$ *(respectively for every* $(a_0, \ldots, a_{k-1}) \in \mathbb{F}_2^k$*).*
3. *Analogously we have that* F_V *and* V_F *are both injective mappings and therefore* $\mathrm{F}_\mathrm{V} = \mathrm{V}_F^{-1}$.
4. *Since* Int_k *nor* V_k *is a homomorphism, we cannot derive the isomorphism between* $\mathbb{Z}_{2^k}$ *and* $\mathbb{F}_2^k$ *from* $\mathrm{Int}_k = \mathrm{V}_k^{-1}$ *(and obviously* $\mathbb{Z}_{2^k} \not\simeq \mathbb{F}_2^k$*).*
5. *We have* $(\mathbb{F}_2, +)^k \simeq (\mathbb{F}_{2^k}, +)$ *as vector space and additive group as elementary isomorphism.*

1.2 Boolean functions

We already described simple bitstring operations used by cryptographical systems in previous section and in this section we focus generally on mappings that operate on Boolean vector space $\mathbb{F}_2^k$ as space of representation of bitstrings with length k.

A Boolean function is a building block to its extended variant a vectorial Boolean function. We define them together in Definition 1 and we will lately focus more on vectorial Boolean functions as an interpretation of our main object of interest - an S-box - a part of cipher Kuzniechik and hash function Streebog.

Lately in this section, we list properties of vectorial Boolean functions and their relevance that will be also touched on in analysis of the S-box.

Definition 1 ([WF16])**.** *A function*

- $f : \mathbb{F}_2^k \longrightarrow \mathbb{F}_2$ *is called the* ***Boolean function*** *BF,*
- $F : \mathbb{F}_2^k \longrightarrow \mathbb{F}_2^n$ *is called the* ***vectorial Boolean function*** *VBF.*

Remark. We can treat a VBF $F : \mathbb{F}_2^k \longrightarrow \mathbb{F}_2^n$ as an ordered n-tuple of BFs $f_0, \ldots, f_{n-1} : \mathbb{F}_2^k \longrightarrow \mathbb{F}_2$ with notation $F(\overline{x}) = (f_0(\overline{x}), \ldots, f_{n-1}(\overline{x}))$ for $\overline{x} \in \mathbb{F}_2^k$

We will be mostly interested in VBF $F : \mathbb{F}_2^k \longrightarrow \mathbb{F}_2^k$ with same dimension of domain and image space and so we consider only this type of VBFs from now on.

Concrete VBF can be defined by a value table where we specify that a preimage $(a_0, \dots, a_{k-1}) \in \mathbb{F}_2^k$ goes to image $(a'_0, \dots, a'_{k-1}) \in \mathbb{F}_2^k$. This representation is usually used for function that does not have any structure in the image e.g., non-linear layer for a cipher as we describe in Section 1.3.1. It is one of simplest definition of a function, since a value table exists for every deterministic function with finite domain.

Another simple representation is the algebraic normal form defined in Definition 2.

Definition 2 ([WF16])**.** *Let $f : \mathbb{F}_2^k \longrightarrow \mathbb{F}_2$ be a BF and let $F : \mathbb{F}_2^k \longrightarrow \mathbb{F}_2^k$ be a VBF. We say that f has the **algebraic normal form** (ANF) if there exist $a_{\overline{u}} \in \mathbb{F}_2$ for every $\overline{u} \in \mathbb{F}_2^k$ such that*

$$f(x) = \sum_{\overline{u} \in \mathbb{F}_2^k} a_{\overline{u}} x^{\overline{u}},$$

with notation $x^{\overline{u}} = \prod_{i=0}^{k-1} x_i^{u_i}$ for $\overline{u} = (u_0, \dots, u_{k-1})$ and specially for F we have

$$F(x) = \sum_{\overline{u} \in \mathbb{F}_2^k} \overline{a}_{\overline{u}} x^{\overline{u}},$$

with $\overline{a}_{\overline{u}} \in \mathbb{F}_2^k$.

We show how to find an algebraic normal form for a VBF in Section 3.2.1. It can be proven that every VBFs have such representation and that such representation is unique for every VBF, because for every two different ANF there is at least one preimage on which those two functions differs. Moreover it allows us to define the degree of VBFs in Definition 3. Degree of a VBF is an important property that influences other properties.

Definition 3 ([WF16])**.** *Let $f : \mathbb{F}_2^k \longrightarrow \mathbb{F}_2$ be a BF and let $F : \mathbb{F}_2^k \longrightarrow \mathbb{F}_2^k$ be a VBF with ANFs*

$$f(x) = \sum_{\overline{u} \in \mathbb{F}_2^k} a_{\overline{u}} x^{\overline{u}}, \ F(x) = \sum_{\overline{u} \in \mathbb{F}_2^k} \overline{a}_{\overline{u}} x^{\overline{u}}.$$

*Let $\mathrm{wf} : \mathbb{F}_2^k \longrightarrow \mathbb{N}_0$ be the **Hamming weight** of $\overline{u} \in \mathbb{F}_2^k$ defined as*

$$\mathrm{wf}(u) = \sum_{i=0}^{k-1} u_i.$$

*We define the **degree** of f as*

$$\deg(f) = \max_{\overline{u} \in \mathbb{F}_2^k} \Big(\mathrm{wf}(\overline{u}) | a_{\overline{u}} \neq 0 \Big),$$

*and the **degree** of F as*

$$\deg(F) = \max_{\overline{u} \in \mathbb{F}_2^k} \Big(\mathrm{wf}(\overline{u}) | \overline{a}_{\overline{u}} \neq (0, \dots, 0) \Big).$$

Remark. Equivalently we can define the degree of a VBF $F = (f_1, \dots, f_k)$ with $f_i : \mathbb{F}_2^k \to \mathbb{F}_2$ for $i = 1, \dots, k$ as

$$\max_{i=1,\dots,k}\Big(\deg(f_i)\Big),$$

where $\deg(f_i)$ is defined as above.

Let us continue with other properties of VBFs. Walsh transform and Walsh spectrum defined in Definition 4 are basic properties that can be computed straightforwardly from the lookup table. They are also tightly connected with linear cryptanalysis of an S-box viewed as a VBF as a part of a cipher. We use the Definition 4 in Section 1.3.3 to define elements in the linear approximation table.

Definition 4. *Let $F : \mathbb{F}_2^k \longrightarrow \mathbb{F}_2^k$ be a VBF and $\langle , \rangle$ the standard scalar product on $\mathbb{F}_2^k$. We define the* ***Walsh transform*** *of F as*

$$\hat{F}(\overline{u}, \overline{v}) = \sum_{\overline{x} \in \mathbb{F}_2^k} (-1)^{\langle \overline{u}, F(\overline{x}) \rangle + \langle \overline{v}, \overline{x} \rangle},$$

for $\overline{u}, \overline{v} \in \mathbb{F}_2^k$ and the ***Walsh spectrum*** *of $F : \mathbb{F}_2^k \longrightarrow \mathbb{F}_2^k$ a vectorial Boolean function as a multiset of Walsh transforms*

$$\mathcal{F} = \{\hat{F}(\overline{u}, \overline{v}) | \overline{u}, \overline{v} \in \mathbb{F}_2^k, \overline{u} \neq (0, \dots, 0)\}.$$

In the same way that the Walsh transforms of a function form the Walsh spectrum, we have $\delta_F(\overline{a}, \overline{b})$ that form the differential spectrum from Definition 5. Walsh transforms are used as elements of linear approximation table in linear cryptanalysis while elements of the differential spectrum are used as elements of difference distribution table in differential cryptanalysis.

Both terms from Definition 5 can be computed straightforwardly from the look up table as well and we use them in Section 1.3.3 to define elements in the differential distribution table.

Definition 5. *Let $F : \mathbb{F}_2^k \longrightarrow \mathbb{F}_2^k$ be a VBF. We define*

$$\delta_F(\overline{a}, \overline{b}) = \#\{\overline{x} \in \mathbb{F}_2^k | F(\overline{x} \oplus \overline{a}) + F(\overline{x}) = \overline{b}\},$$

for $\overline{a}, \overline{b} \in \mathbb{F}_2^k$ and the ***differential spectrum*** *of $F : \mathbb{F}_2^k \longrightarrow \mathbb{F}_2^k$ a vectorial Boolean function as a multiset*

$$\Delta_F = \{\delta_F(\overline{a}, \overline{b}) | a, b \in \mathbb{F}_2^k\}.$$

We just defined properties of vectorial Boolean function that can play a role in an analysis. There exist equivalencies between function that preserves those properties i.e., we have always function with same properties in one equivalency class.

In the thesis we are focused on one particular VBF - the S-box of Kuznyechik and Streebog - and we want to somehow locate is in the set of all VBF. If we divide the set into equivalency classes we can then more easily locate our VBF in one or more equivalency classes by finding value of several properties. Or more precisely, we can identify those in which equivalency classes our VBF is not located, which is in those that contain functions with different values of the properties.

We define a equivalency relation in Definition 6.

Definition 6. *Let $F_1, F_2 : \mathbb{F}_2^k \longrightarrow \mathbb{F}_2^k$ be two VBFs. We say that F_1 is **affine-equivalent** to F_2 and we write $F_1 \sim_{EA} F_2$ if and only if*

$$F_1(\overline{x}) = A_2 \circ F_2 \circ A_1(\overline{x}) + A_3(\overline{x}) \text{ for every } \overline{x} \in \mathbb{F}_2^k,$$

for some A_1, A_2 affine permutation of $\mathbb{F}_2^k$ and A_3 affine map on $\mathbb{F}_2^k$.

Properties in Definition 3, 4 and 5 are not chosen at random to be mentioned in the thesis. These are properties that are preserved through composition with affine mappings as we prove it in Theorem 3. We formally name such properties in Definition 7.

Definition 7. *Let $\sim$ be an equivalency and ρ a property. We say that ρ is an **invariant** with respect to $\sim$ if for every $F_1 \sim F_2$ it holds that ρ of F_1 is equal to ρ of F_2.*

We continue straightforward to the theorem which says that properties such as the degree of a function, absolute value of elements from the Walsh spectrum and the differential spectrum are preserved through affine equivalency. We give also its proof to demonstrate that it follows from basic theory and only needs some technical manipulation.

Theorem 3. *Let $F : \mathbb{F}_2^k \longrightarrow \mathbb{F}_2^n$ be a not constant vectorial Boolean function. The following properties are invariant with respect to affine-equivalency:*

1. $\deg(F)$ *the degree of F,*
2. $|\mathcal{F}| = \{|\hat{F}(\overline{u}, \overline{v})| \; |\hat{F}(\overline{u}, \overline{v}) \in \mathcal{F}\}$ *the multiset of absolute values of Walsh spectrum,*
3. Δ_F *the differential spectrum.*

Proof. We denote the degree of F_1, F_2 as d_1, d_2 respectively. We suppose that $d_1, d_2 > 0$ since constant and zero functions are not interesting for us.

Let us follow the notation from Definition 6. We show always that none of transformations

a. $F \mapsto F + A_3$

b. $F \mapsto F \circ A_1$

c. $F \mapsto A_2 \circ F$

change the invariant.

1. We will follow notation from Definition 3.

   a. Let us consider $A_3 : \mathbb{F}_2^k \to \mathbb{F}_2^k$ as

$$(x_1, \ldots, x_k) \mapsto ((\sum_{i_1=1}^{k} x_{i_k}) + b_{3,1}, \ldots, (\sum_{i_k=1}^{k} x_{i_k}) + b_{3,k})$$

then transformation $F \mapsto A_3$ will give us

$$F = (f_1, \dots, f_k) \mapsto (f_k + (\sum_{i_1=1}^{k} x_{i_1}) + b_{3,1}, \dots, f_k + (\sum_{i_k=1}^{k} x_{i_k}) + b_{3,k})$$

where $b_{3,i}$ are constants and $deg(f_i) >= 1$. Degree of the sum is always less or equal 1 and therefore it cannot change the degree of f_i. Finally, by Definition 3 the degree of the right side is not changed neither.

b. Let us consider $A_1 : \mathbb{F}_2^k \to \mathbb{F}_2^k$ as

$$(x_1, \dots, x_k) \mapsto ((\sum_{i_1=1}^{k} x_{i_1}) + b_{1,1}, \dots, (\sum_{i_k=1}^{k} x_{i_k}) + b_{1,k})$$

then transformation $F \mapsto F \circ A_1$ give us

$$F = (f_1, \dots, f_k) \mapsto (f_1(\sum_{i_1=1}^{k} x_{i_1}), \dots, f_k(\sum_{i_k=1}^{k} x_{k_k}))$$

Let $x^{\overline{u}}$ be the term for which $\deg(F) = \sum_{i=1}^{k} u_i$. The transformation will give us

$$x^{\overline{u}} = \prod_{i=1}^{k} u_i x_i \mapsto \prod_{i=1}^{k} u_i((\sum_{j=1}^{k} x_i) + c_i)$$

where c_i is a constant. Number of not zero u_i stays the same after the transformation and therefore also degree of every terms after we multiply out the sum stays the same. Analogically every term of the algebraic normal form keeps its degree and therefore also degree of the right side of transformations stays the same.

c. Let us consider $A_2 : \mathbb{F}_2^k \to \mathbb{F}_2^k$ as

$$(x_1, \dots, x_k) \mapsto ((\sum_{i_1=1}^{k} x_{i_1}) + b_{2,1}, \dots, (\sum_{i_k=1}^{k} x_{i_k}) + b_{2,k})$$

then transformation $F \mapsto F \circ A_1$ give us

$$F = (f_1, \dots, f_k) \mapsto ((\sum_{i_1=1}^{k} f_{i_1}) + b_{2,1}, \dots, (\sum_{i_k=1}^{k} f_{k_k}) + b_{2,k})$$

where $b_{2,i}$ are constant. Sum of function of certain degree cannot produce any new term of higher degree than the maximal one that existed before and at the same time cannot reduce the term of maximal degree because A_2 is a permutation and the only part of A_2 that can reduce the degree because of sum of two or more f_i can be represented by a non singular matrix and therefore if one row reduce the degree there must exist another that will keep it.

2. We follow the notation from Definition 4. We have

$$\hat{F}(\overline{u}, \overline{v}) = \sum_{\overline{x} \in \mathbb{F}_2^k} (-1)^{\langle \overline{u}, F(\overline{x}) \rangle + \langle \overline{v}, \overline{x} \rangle}$$

a. after the transformation $F \mapsto F + A_3$ we get

$$\begin{aligned} F \hat{+} A_3(\overline{u}, \overline{v}) &= \sum_{\overline{x} \in \mathbb{F}_2^k} (-1)^{\langle \overline{u}, F(\overline{x}) + A_3(\overline{x}) \rangle + \langle \overline{v}, \overline{x} \rangle} \\ &= \sum_{\overline{x} \in \mathbb{F}_2^k} (-1)^{\langle \overline{u}, F(\overline{x}) \rangle + \langle \overline{u}, A_3(\overline{x}) \rangle + \langle \overline{v}, \overline{x} \rangle} \\ &= \sum_{\overline{x} \in \mathbb{F}_2^k} \left((-1)^{\langle \overline{u}, F(\overline{x}) \rangle + \langle \overline{v}, \overline{x} \rangle} \cdot (-1)^{\langle \overline{u}, A_3(\overline{x}) \rangle} \right) \end{aligned}$$

b. after the transformation $F \mapsto F \circ A_1$ we get

$$\begin{aligned} F \hat{\circ} A_1(\overline{u}, \overline{v}) &= \sum_{\overline{x} \in \mathbb{F}_2^k} (-1)^{\langle \overline{u}, F \circ A_1(\overline{x}) \rangle + \langle \overline{v}, \overline{x} \rangle} \\ &= \sum_{\overline{x} \in \mathbb{F}_2^k} (-1)^{\langle A_1^{-1}(\overline{u}), F(\overline{x}) \rangle + \langle \overline{v}, \overline{x} \rangle} \\ &= \hat{F}(A_1^{-1}(\overline{u}), \overline{v}) \end{aligned}$$

A_1 is a permutation and therefore $|\mathcal{F}|$ by definition stays the same.

c. after the transformation $F \mapsto F \circ A_2$ we get analogously

$$A_2 \hat{\circ} F(\overline{u}, \overline{v}) = \hat{F}(A_2^{-1}(\overline{u}), \overline{v})$$

A_2 is a permutation and therefore $|\mathcal{F}|$ by definition stays the same.

3. We follow the notation from Definition 5. We have

$$\delta_F(\overline{a}, \overline{b}) = \#\{\overline{x} \in \mathbb{F}_2^k | F(\overline{x} \oplus \overline{a}) + F(\overline{a}) = \overline{b}\}$$

a. after the transformation $F \mapsto F + A_3$ we get

$$\begin{aligned} \delta_{F+A_3}(\overline{a}, \overline{b}) &= \#\{\overline{x} \in \mathbb{F}_2^k | F(\overline{x} \oplus \overline{a}) + A_3(\overline{x} \oplus \overline{a}) + F(\overline{x}) + A_3(\overline{a}) = \overline{b}\} \\ &= \#\{\overline{x} \in \mathbb{F}_2^k | F(\overline{x} \oplus \overline{a}) + A_3(\overline{x}) + A_3(\overline{a}) + A_3(\overline{0}) + F(\overline{x}) + A_3(\overline{x}) = \overline{b}\} \\ &= \#\{\overline{x} \in \mathbb{F}_2^k | F(\overline{x} \oplus \overline{a}) + A_3(\overline{a}) + A_3(\overline{0}) + F(\overline{a}) = \overline{b}\} \\ &= \#\{\overline{x} \in \mathbb{F}_2^k | F(\overline{x} \oplus \overline{a}) + F(\overline{a}) = \overline{b} + A_3(\overline{a}) + A_3(\overline{0})\} \\ &= \delta_F(\overline{a}, \overline{b} + A_3(\overline{a}) + A_3(\overline{0}), \end{aligned}$$

b. after the transformation $F \mapsto F \circ A_1$ we get

$$\begin{aligned} \delta_{F \circ A_1}(\overline{a}, \overline{b}) &= \#\{\overline{x} \in \mathbb{F}_2^k | F \circ A_1(\overline{x} \oplus \overline{a}) + F \circ A_1(\overline{x}) = \overline{b}\} \\ &= \#\{\overline{x} \in \mathbb{F}_2^k | A_1^{-1}(F \circ A_1(\overline{x} \oplus \overline{a}) + F \circ A_1(\overline{x})) = A_1^{-1}(\overline{b})\} \\ &= \#\{\overline{x} \in \mathbb{F}_2^k | A_1^{-1}(\overline{0}) + F(\overline{x} \oplus \overline{a}) + F(\overline{x})) = A_1^{-1}(\overline{b})\} \\ &= \#\{\overline{x} \in \mathbb{F}_2^k | F(\overline{x} \oplus \overline{a}) + F(\overline{x})) = A_1^{-1}(\overline{0}) + A_1^{-1}(\overline{b})\} \\ &= \delta_F(\overline{a}, A_1^{-1}(\overline{0}) + A_1^{-1}(\overline{b})) \end{aligned}$$

A_1 is a permutation and therefore Δ_F stays the same,

c. after the transformation $F \mapsto A_2 \circ F$ we get

$$\delta_{F\circ A_2}(\overline{a}, \overline{b}) = \delta_F(\overline{a}, A_2^{-1}(\overline{0}) + A_2^{-1}(\overline{b}))$$

A_2 is a permutation and therefore Δ_F stays the same.

□

Theorem 3 gives us a way how to distinguish whether two vectorial boolean functions can be affine equivalent or more precisely thanks to the theorem we can identify the case where two function are not affine equivalent by comparison of their properties. We actually use the inverted implication, which says that if we came up on two functions with different properties, we know that they are not affine equivalent.

In Chapter 3, we will use the comparison of certain property in our experiment where we will try to locate our VBF of interest in specific subset of functions or more precisely to show that it can not be located there.

1.3 Cryptosystems and basics of cryptanalysis

Ciphers and hash functions are two of main tools that cryptography uses to provide security and integrity of data. We give a general scheme and a definition of a cipher, specifically the block cipher with SPN scheme and a hash function as general foundations so that we can describe specifics of the cipher Kuznyechik and the hash function Streebog that use our S-box of interest in Chapter 2.

1.3.1 Ciphers

Ciphers as we understand them here are designed to transfer secret message from person A to person B so that only persons A and B are able to get the original message from the encrypted one thanks to exclusive knowledge of a key. Such scheme follows from Kirchhoff's principles. We define a scheme of a cipher in Definition 8.

Definition 8 (Cipher)**.** *We define a **cipher** as a 5-tuple $(\mathcal{P}, \mathcal{C}, \mathcal{K}, \mathcal{E}, \mathcal{D})$, where:*

- *$\mathcal{P}$ is a set of plaintexts (messages to be encrypted),*
- *$\mathcal{C}$ is a set of ciphertexts (encrypted messages),*
- *$\mathcal{K}$ is a set of keys (secret parameter in process of encryption),*
- *$\mathcal{E} : \mathcal{P} \times \mathcal{K} \longrightarrow \mathcal{C}$ is the encryption function,*
- *$\mathcal{D} : \mathcal{C} \times \mathcal{K} \longrightarrow \mathcal{P}$ is the decryption function,*

*To be considered as a **well defined cipher** we require*

$$\mathcal{D}(\mathcal{E}(m, k), k) = m, \ \forall m \in \mathcal{P}, \ \forall k \in \mathcal{K}.$$

Remark. Definition 8 can be used for both symmetric and asymmetric cipher, where the main difference lie in the set $\mathcal{K}$ - set of keys - and its usage in $\mathcal{E}$ and $\mathcal{D}$.

- Symmetric cryptosystem uses the same key for encryption and decryption,
- asymmetric cryptosystem uses a couple of keys, so-called public key k_p for encryption and so-called secret key k_s for decryption. They are related in a way that

$$\mathcal{D}(\mathcal{E}(m, k_p), k_s) = m, \ \forall m \in \mathcal{P}, \ \forall (k_p, k_s) \in \mathcal{K}.$$

Kuznyechik as the cipher of interest in this thesis is as we call it a block cipher and moreover Kuznyechik is a block cipher that uses a structure similar to a substitution-permutation network in encryption and decryption function and therefore we focus here on symmetric ciphers and especially block ciphers that has specific structure of $\mathcal{P}$ and $\mathcal{C}$. We define notion of block ciphers in Definition 9.

Definition 9. [MvOV01, Definition 7.1] *We define a m-bit block cipher as cipher* $(\mathcal{P}, \mathcal{C}, \mathcal{K}, \mathcal{E}, \mathcal{D})$ *with:*

1. $\mathcal{P} = \mathcal{C} = \mathbb{F}_2^m$,
2. $\mathcal{E}(p, K)$ *invertible mapping for every* $K \in \mathcal{K}$ *with inverse mapping* $\mathcal{D}(c, K)$ *such that we have* $m = \mathcal{D}(\mathcal{E}(m, K), K)$ *for every* $m \in \mathcal{P}$.

Remark. Second point in Definition 9 says only that we use the same key for encryption and decryption and that it is a well defined cipher by Definition 8

Remark. In practice, to encrypt longer messages than k bits, we split the message into k-bits block and encrypt each block on its own. To connect those block one can use several modes of operation as CFB, FCB or OFB and many more. Majority of them are described in [KR11, Chapter 4], but we are not interested in how the modes of operation plays the role and focus therefore only on case when size of the message matches exactly the size of the block.

As we already mentioned, block ciphers can be of special kind that uses a substitution-permutation network (SPN). It has a specific structure of encryption and decryption. The advantage lies in easy implementation, possible scalability. It is a proven model that provides confusion and diffusion, two basic properties introduced by Shannon in [eS49, Chapter 23]. These are properties of a cipher that should provide resistance for instance against statistical attack or attempt of key recovery from the ciphertext.

Definition 10 (SPN)**.** *The cryptosystem* $(\mathcal{P}, \mathcal{C}, \mathcal{K}, \mathcal{E}, \mathcal{D})$ *is called* ***substitution-permutation network*** *with* $n \in \mathbb{N}$ *rounds if:*

- $\mathcal{P} = \mathcal{C} = \mathbb{F}_2^m$,
- $\mathcal{K} = \mathbb{F}_2^k$ *with expansion function* $\mathrm{KeyGen} : \mathcal{K} \longrightarrow (\mathbb{F}_2^m)^r$ *expanding the main key* $k \in \mathbb{F}_2^k$ *into* r *round keys* $k_1, \ldots, k_r \in \mathbb{F}_2^m$,
- $\mathcal{E}$ *consist of* r *rounds, the first* $r - 1$ *rounds performing sequence of layers:*
  - $*$ *round key addition,*
  - $*$ *linear permutation of bits,*

  * *non-linear substitution,*

  the last round applying only the key addition,

- *$\mathcal{D}$ is the inverse of $\mathcal{E}$ in a way that we apply inverse of each layer in reverse order.*

Remark. There exist variations of SPN that replace the permutation layer by a linear layer. Then, we can call it substitution-linear network (SLN). It is the structure used in Kuznyechik block cipher, which is described in Section 2.1.2. However we have to always keep the round key addition, the non-linear substitution and one layer in between that provides diffusion.

If we let apart the size of a block, ciphers using the SPN differs by choice of KeyGen, the permutation layer and the non-linear substitution layer. We give some comment on each of mentioned part of a cipher.

KeyGen should be a function expanding the main key into round keys in a way that every bit of information in the main key is used and no information is left out. Individual round keys should be independent with each other in a way that knowledge of one round key or part of the main key does not provide much information about the rest.

Permutation layer is commonly chosen similarly so that the change of one bit on the input is distributed into approximately half of bits on the output, which can provide us diffusion in a cipher as defined by Shannon [eS49, Chapter 23].

The main strength of the cipher lies in the non-linear substitution layer. Linear and differential cryptanalysis can deal with other layers quite easily, but this is the one that can make the cipher hard or easy to attack thanks to confusion as defined by Shannon [eS49, Chapter 23].

The most common practice is the so–called S–box function as the non-linear layer. There exist various types of S-boxes, there are fixed S-boxes that are independent of the key or those generated dynamically with respect to to key. An S-box can have domain and image vector spaces of the same or different dimension. In Definition 11 we define notion of S-box as we will use in the thesis.

Definition 11. *Let m be the block size of a block cipher. We define the **S-box** as a function $\varsigma : \mathbb{F}_2^{\frac{m}{n}} \longrightarrow \mathbb{F}_2^{\frac{m}{n}}$ with $n \in \mathbb{N}$, such that ς permutes $\mathbb{F}_2^{\frac{m}{n}}$.*

Let us explain how the S-box as the non-linear layer in a SPN can work. We first split the input with m bits into n smaller bitstrings with $\frac{m}{n}$ bits, apply the S-box on each smaller bitstring and we concatenate outputs of the S-box back again in one bitstring of m bits. All of these bitstring manipulations are covered by Notation 3.

As an S–box is usually given by a value table, it could be hard to find a structure in this layer since first one have to convert this value table into another representation. We already mentioned the ANF and we will also discuss our S-box of interest as the Lagrange interpolation polynomial in Section 3.2.2. Representation by a Lagrange interpolation polynomial exists for every function if it is considered as function on a finite field.

A good S-box is considered if the function ς provides significant confusion if it is used in a SPN as the non-linear substitution layer. Properties that can be mesured and that play a role in linear or cryptanalysis can be e.g. the degree of the S-box function. For instance if the degree is equal to 1, we have a linear function and we do not get any confusion.

Also other properties can be considered such as the Walsh spectrum and differential spectrum, which are propagated to linear approximation table and diference distribution table as we define them in Section 1.3.3. They are also propagated to linear and differential anomaly that we define in Section 2.3.2 with a brief comment about how to interpret those terms as indicators for easy or hard to attack functions.

1.3.2 Hash functions

Hash functions are intended to preserve integrity of data. A hash function can be used in a situation, when we want to sign a data to guarantee the non-repudiation of an authorization. Therefore we introduce hash function because then we can take only a hash print of the data of fixed size and sign it instead of signing probably large data of different size at each time. We define a hash function in Definition 12 taken directly from [MvOV01].

Definition 12 ([MvOV01], Definition 9.1)**.** *We define a **hash function** as mapping h having two properties:*

1. *(compression) $h : (\mathbb{F}_2)^* \longrightarrow \mathbb{F}_2^k$ maps any element of finite bit length from $(\mathbb{F}_2)^*$ to an element of fixed bit length from $\mathbb{F}_2^k$,*

2. *(ease of computation) computation of $h(x)$ from an input x is "easy".*

It is relevant to explain here what can be meant by "easy" computation. We do not want to sink in the theory of complexity or computational and provable security and we give only an rough comment. In practice a hash function should be used as that anyone can generate a hash of a data and any another can validate the hash by computing it again from the data. Therefore we require the computation to be "easy" to cover this use case.

At the same time if we want to use a hash function as described at the beginning of the section we implicitly require that inverse of the hash function is "hard" to compute so that no one can systematically compute preimages from the hash value. The requirement of "easy" computation of the hash value also exclude those function that has inverse that is "hard" to compute but it is also "hard" to compute the direct way to get the hash value, because such functions can be secure but not practical.

A good hash function has certain properties. The aim is to be able to see the hash value $h(m)$ of a message m as somehow unique representative for the message m. It suits the purpose to use it in signature schemes. But since a hash function accepts more inputs than it can produce different outputs, we have guarantied existence of collisions. A signature of a hash value can be considered as unique with respect to signed data without knowledge of the signature key only if it is "hard" to find collisions for the hash function.

There exists several problems around finding preimages of the hash functions and until one prove that he can solve them efficiently, the hash function is considered secure and collision resistant. We list them with respect to its complexity in descending order:

1. For $y \in \mathbb{F}_2^k$ a hash print find arbitrary number of preimages $x_i \in (\mathbb{F}_2)^*$, pairwise different such that $h(x_i) = y$, $\forall i$,
2. For $x \in (\mathbb{F}_2)^*$ find a $x' \in (\mathbb{F}_2)^*$ such that $x \neq x'$ and $h(x) = h(x')$,
3. Find any $x, x' \in (\mathbb{F}_2)^*$ such that $x \neq x'$ and $h(x) = h(x')$.

In this thesis we are interested in an S-box used in Streebog hash function described in Section 2.1.2. Streebog has commonly used sponge structure which consumes the message by blocks of fixed length and with each block it proceed operations similar to SPN with the S-box that is common to the Kuznyechich cipehr described in Section 2.1.1.

1.3.3 Basics of linear and differential cryptanalysis

When we study an S–box and its properties, we always come across its linear and differential properties. S–boxes are usually permutations that are designed not to have an algebraic structure, but we have several tools to analyse some properties.

Process of linear or differential attack themselves can be found in multiple publications, for example in [KR11], whereas we focus only on several terms used in Section 2.3.2. We will describe there Perrin's discoveries in [Per19] about properties of the S–box π and we will need basic terms to be defined here.

First of all we give a definition of the linear approximation table and the differential distribution table in Definition 13, where both designations indicate the purpose of such terms. We use here also already defined terms from Definition 4 and 5.

Definition 13 (LAT and DDT, [KR11]). *Let $F : \mathbb{F}_2^n \longrightarrow \mathbb{F}_2^n$ be a vectorial Boolean function, $i, j \in \mathbb{F}_2^n$. We define:*

$$\begin{aligned} \mathrm{LAT}_F(i,j) &= \hat{F}(i,j), \\ \mathrm{LAT}_F &= \{\mathrm{LAT}_F(i,j)\}_{i,j\in\mathbb{F}_2^n} \text{ the } \textbf{\textit{linear approximation table}}, \\ \mathrm{DDT}_F(i,j) &= \delta_F(i,j), \\ \mathrm{DDT}_F &= \{\mathrm{DDT}_F(i,j)\}_{i,j\in\mathbb{F}_2^n} \text{ the } \textbf{\textit{difference distribution table}}, \end{aligned}$$

The linear approximation table shows how close the S–box is to a linear function. On the position (i, j) of LAT_F we can find a whole number representing the bias of $\langle a, i\rangle \oplus \langle F(a), j\rangle$, i.e how often we can see that $\langle a, i\rangle = \langle F(a), j\rangle$ relatively to $\langle a, i\rangle \neq \langle F(a), j\rangle$. If one of them happens more often, $\mathrm{LAT}_F(i, j)$ has bigger absolute value than if both cases happen similarly likely. If we have F linear the LAT_F would look like a scaled permutation matrix, i.e. there will be one and only non–zero term in each line and column and those will have a large absolute value.

The difference distribution table describes the situation, when we take two elements with a fixed difference and we observe the difference of their images, after we run them through the S–box. If we consider the output differences with input difference equal to i as a 'random' variable, we can say that the differential distribution table has distribution of this 'random' variable as the i–th row.

The other terms used in Section 2.3.2 are the differential uniformity and the linearity of S–box. These terms should denote how easy it would be to attack the

S–box with differential and linear attack. High values are sign of possibly easy attacks. Formally we define those terms in Definition 14.

Definition 14 (differential uniformity and linearity,[Per19])**.** *Let $F : \mathbb{F}_2^n \longrightarrow \mathbb{F}_2^n$ be a vectorial Boolean function and $i, j \in \mathbb{F}_2^n$, then we define:*

$$\lambda_F = \max_{i \neq 0, j}(|\operatorname{LAT}_F(i,j)|) \text{ to be the } \textbf{\textit{linearity}} \text{ of } F,$$
$$\delta_F = \max_{i \neq 0, j}(\operatorname{DDT}_F(i,j)) \text{ to be the } \textbf{\textit{differential uniformity}} \text{ of } F.$$

Remark. In Definition 14 we take the maximum over i, j with $i \neq 0$ because the first column of the LAT_F and DDT_F is always special. On the position (0,0) always the highest value 2^n and the rest is equal to 0 and we are not interested in that case.

For LAT_F position (0,0) corresponds to situation when we take input and output mask equal to 0 and we have $\Sigma_{x \in \mathbb{F}_2^k}(-1)^0 = 2^k$, and other positions became 0 since $\Sigma_{x \in \mathbb{F}_2^k}(-1)^{\langle F(x), v\rangle} = \Sigma_{x \in \mathbb{F}_2^k}(-1)^{\langle x, v\rangle} = 0$ for any $v \in \mathbb{F}_2^k$, For DDT_F first column corresponds to the situation when we take two same inputs and then we observe the difference between outputs that has to be 0 as well.

1.4 Algebraic geometry

In order to start talking about projective permutations over finite field later in the thesis, especially in Chapter 3, we summarise basic terms in this section with focus on fractional polynomial functions and projective linear mappings. Projective permutations as the designation indicates operates on projective space, which has to be defined first and we do so in Definition 16. Definition of a projective space is based on definition of an affine space, which is simple algebraic structure as we can see it in Definition 15.

We define following basic terms in general context, but we will need the definitions mostly for dimension 1 or 2. Affine and projective space are structures related one to another and projective permutations are operating on them.

Definition 15 ($\mathbb{A}^n(K)$,[Wil08])**.** *We define an **affine n–space over K** as a set of n–tuples of elements from K, where K is a field. (notation: $\mathbb{A}^n(K) = \{(a_1, \ldots, a_n) | a_i \in K\}$)*

Definition 16 ($\mathbb{P}^n(K)$,[Wil08])**.** *We define a **projective n–space over K** as a set of all lines in $\mathbb{A}^{n+1}(K)$ passing by the point (0,...,0). (notation: $\mathbb{P}^n(K)$)*

Remark. Lines in $\mathbb{A}^{n+1}$ are sets

$$\{(\lambda x_1, ..., \lambda x_{n+1}) | \lambda \in K\}.$$

Therefore every point $x \in \mathbb{A}^{n+1}$

$$x = (x_1, \ldots, x_{n+1}) \neq (0, \ldots, 0)$$

belongs to one and only line that passes through point $(0, \ldots, 0)$.

Two points $x, y \in \mathbb{A}^{n+1}$

$$x = (x_1, \dots, x_{n+1}),\ y = (y_1, \dots, y_{n+1}),\ x, y \neq (0, \dots, 0),$$

belong to the same line that passes through $(0, \dots, 0)$ if and only if

$$\exists \lambda \neq 0, \lambda \in K \text{ such that } y_i = \lambda x_i\ \forall i = 1, ..., n+1.$$

We can then consider x and y equivalent and represent elements of $\mathbb{P}^n(K)$ as the set of equivalence classes of points in $\mathbb{A}^{n+1} \setminus \{(0, \dots, 0)\}$ with following notation.

Notation 5 ([Wil08])**.** *We will represent elements of $\mathbb{P}^n(K)$ as the set of equivalence classes of points in $\mathbb{A}^{n+1} \setminus \{(0, \dots, 0)\}$ with notation*

$$[x_1 : \dots : x_{n+1}] = \{(\lambda x_1, ..., \lambda x_{n+1}) | \lambda \in K\}.$$

as homogeneous coordinates.

Let us choose a fixed coordinates. We can separate those elements that has 0 or a non zero element on that coordinate. Since $\lambda \in K$ there is always one point from the line that has the coordinate equal to 1. Elements with 1 on that coordinate corresponds one to one to elements from $\mathbb{A}^n(K)$ because contain all combinations on n coordinates. Those lines that has 0 on the chosen positions correspond on the other hand to elements from $\mathbb{P}^{n-1}$. Inductively we get following observation.

Observation 4. *We have $\mathbb{P}^n(K) = \mathbb{A}^n(K) \cup \mathbb{P}^{n-1}(K)$ and inductively*

$$\mathbb{P}^n(K) = (\bigcup_{i=1}^{n} \mathbb{A}^i(K)) \cup \mathbb{P}^0(K).$$

As we already mentioned, we will be operating on case $n = 1$ and therefore we can use decomposition $\mathbb{P}^1(K) = \mathbb{A}^1(K) \cup \mathbb{P}^0(K)$. It is not difficult to deduce that $\mathbb{P}^0(K)$ is a single point set and so we name it in Notation 6.

Notation 6. *The single point in $\mathbb{P}^0(K)$ will be called **point at infinity** with notation ∞.*

With Notation 6 we can finally consider the projective 1-space with structure:

$$\mathbb{P}^1(K) = \mathbb{A}^1(K) \cup \{\infty\}.$$

and we can move on to the mappings that operate on $\mathbb{P}^1(K)$, especially, we will focus on fractional polynomials. We define such mappings and their evaluation in Definition 18.

One more thing to define remains and that is an algebraic variety representing a selection of points from an affine space, that are roots to a set of polynomials. We define it in Definition 17 and we use it right after in Definition 18 to define a fractional polynomial and its evaluation.

Definition 17. *Let K be a field and $g \in K[x_1, \dots, x_n]$. We define a **variety** as*

$$V(g) := \{a \in \mathbb{A}^n(K) | g(a) = 0\}.$$

Definition 18. *Let K be a field and $f, g \in K[x]$ such that $V(f) \cap V(g) = \{\}$. We define a* ***fractional polynomial*** *that operates on $\mathbb{P}^1(K)$ as*

$$\frac{f}{g}(x) = \frac{f(x)}{g(x)}$$

with ***evaluation*** *that covers every element of $\mathbb{P}^1(K)$:*

1. *for $x \in \mathbb{A}^1(K) \setminus V(g)$ we define $\frac{f}{g}(x) := \frac{f(x)}{g(x)}$ naturally as it is well defined in K,*
2. *for $x \in V(g)$ we define $\frac{f}{g}(x) := \infty$ as $V(f) \cap V(g) = \{\}$,*
3. *for $x \in \mathbb{P}^1(K) \setminus \mathbb{A}^1(K)$ (i.e. $x = \infty$) we define*

$$\frac{f}{g}(x) = \frac{f(x)}{g(x)} = \begin{cases} \frac{a_{q+1}}{b_{q+1}} & \textit{deg } f = \textit{def } g \\ \infty & \textit{deg } f > \textit{def } g \\ 0 & \textit{deg } f < \textit{def } g \end{cases} \tag{1.1}$$

with common practice where $\frac{c}{0} = \infty$ and $\frac{c}{\infty} = 0$ for every $c \in K \setminus \{0\}$.

We define a special kind of fractional polynomial in Chapter 3 as a key term taken from [Gö22]. In our analysis in Chapter 3 we will need notion of projective equivalence using Möbius transformations as we define it in Definition 19 and 20. Both definition are taken from [Gö22].

Definition 19. *Let K be a field. We define the* ***Möbius transformations*** *as*

$$\mathfrak{M}(K) = \Big\{x \mapsto \frac{ax+b}{cx+d} : a, b, c, d \in K | ad - bc \neq 0\Big\}.$$

Observation 5. *$\mathfrak{M}(K)$ forms a group with composition operation and*

$$\mathfrak{M}(K) \cong \mathrm{PGL}_2(K),$$

where $\mathrm{PGL}_2(K)$ is a set of fractional linear transformations.

Definition 20 ($\sim_{\mathrm{PGL}_2(\mathbb{L})}$)**.** *For Π_{f_1,g_1}, Π_{f_2,g_2} fractional q-projective functions we say that they are* ***projectively equivalent*** *(notation: $\Pi_{f_1,g_1} \sim_{\mathrm{PGL}_2(\mathbb{L})} \Pi_{f_2,g_2}$) if for $\mu_1, \mu_2 \in \mathrm{PGL}_2(\mathbb{L})$*

$$\Pi_{f_1,g_1} = \mu_1 \circ \Pi_{f_2,g_2} \circ \mu_2.$$

Möbius transformations are interesting set of mappings because up to scalar multiplication it corresponds to set of all regular matrices of dimension 2×2 over a field. Another property can be observed, every Möbius transform can be obtained by composition of selected simple transformation. We name them in Notation 7.

Notation 7. *Let K be a field. We will call some of simple Möbius transformations as followed:*

- *$f_{T,b}(x) = x + b$, i.e. $f_1 = \frac{ax+b}{cx+d}$ for $a = c = 0, d = 1$ will be called translation by element b,*

- *$f_{D,a}(x) = ax$, i.e. $f_2 = \frac{ax+b}{cx+d}$ for $b = c = 0, d = 1$ will be called dilation,*
- *$f_I(x) = \frac{1}{x}$, i.e. $f_3 = \frac{ax+b}{cx+d}$ for $a = d = 0, c = b = 1$ will be called inversion.*

It is known that composition of translations, homotheties and inversions cover whole set of Möbius transformation and in Observation 6 we give explanation of how the composition looks like for arbitrary Möbius transform.

Observation 6. *Let K be a field and $f = \frac{ax+b}{cx+d}$ a Möbius transformation. Using simple transformations from Notation 7 we can get f as composition*

$$f_{T,\frac{a}{c}} \circ f_{D,\frac{bc-ad}{c^2}} \circ f_I \circ f_{T,\frac{d}{c}} = f.$$

Explicitly, we have:

$$\begin{aligned}
f_{T,\frac{a}{c}} \circ f_{D,\frac{bc-ad}{c^2}} \circ f_I \circ f_{T,\frac{d}{c}}(x) &= (x + \frac{a}{c}) \circ (\frac{bc-ad}{c^2}x) \circ (\frac{1}{x}) \circ (x + \frac{d}{c}) \\
&= (x + \frac{a}{c}) \circ (\frac{bc-ad}{c^2}x) \circ (\frac{1}{x + \frac{d}{c}}) \\
&= (x + \frac{a}{c}) \circ (\frac{\frac{bc-ad}{c^2}}{x + \frac{d}{c}}) \\
&= \frac{\frac{bc-ad}{c^2}}{x + \frac{d}{c}} + \frac{a}{c} \\
&= \frac{\frac{1}{c}(bc-ad) + a(x + \frac{d}{c})}{cx+d} \\
&= \frac{b - \frac{ad}{c} + ax + \frac{ad}{c})}{cx+d} \\
&= \frac{ax+b}{cx+d} = f(x)
\end{aligned}$$

Observation 6 will be our start point in an analysis of classes of equivalency concerning the projective equivalency from Definition 20 on special kind of fractional polynomials defined later in the thesis. One of aim of the analysis will be selection of those Möbius transforms that preserves invariants mentioned in Observation 3. We dedicate Chapter 3 to it.

We show in the following example how does the Möbius transformations act on a special kind of functions. We take as example functions we will define later in Definition 18 and how does the composition with Möbius transformation looks like.

Example. Let $\mu_1, \mu_2 \in \mathfrak{M}(K)$

$$\mu_1 = \frac{c_1x + c_2}{c_3x + c_4}, \mu_2 = \frac{d_1x + d_2}{d_3x + d_4},$$

$q \in \mathbb{N}, q < |K| - 1$ and Π a fractional polynomial with coefficients from K

$$\Pi(x) = \frac{a_{q+1}x^{q+1} + a_qx^q + a_1x + a_0}{b_{q+1}x^{q+1} + b_qx^q + b_1x + b_0}.$$

then

$$\mu_2 \circ \Pi \circ \mu_1 = \frac{d_1\left(\frac{a_{q+1}(\frac{c_1x+c_2}{c_3x+c_4})^{q+1}+a_q(\frac{c_1x+c_2}{c_3x+c_4})^{q}+a_1(\frac{c_1x+c_2}{c_3x+c_4})+a_0}{b_{q+1}(\frac{c_1x+c_2}{c_3x+c_4})^{q+1}+b_q(\frac{c_1x+c_2}{c_3x+c_4})^{q}+b_1(\frac{c_1x+c_2}{c_3x+c_4})+b_0}\right)+d_2}{d_3\left(\frac{a_{q+1}(\frac{c_1x+c_2}{c_3x+c_4})^{q+1}+a_q(\frac{c_1x+c_2}{c_3x+c_4})^{q}+a_1(\frac{c_1x+c_2}{c_3x+c_4})+a_0}{b_{q+1}(\frac{c_1x+c_2}{c_3x+c_4})^{q+1}+b_q(\frac{c_1x+c_2}{c_3x+c_4})^{q}+b_1(\frac{c_1x+c_2}{c_3x+c_4})+b_0}\right)+d_4},$$

which after processing to elementary form of fraction gives huge fractional polynomial. The interesting thing is that numerator as well as the denominator have then the same degree equal to the degree of the one that has bigger degree at the beginning.

Another interesting thing could be coefficients of numerator and denominator for x^{q+1} because as we define in Definition 18, their quotient is image of the point at infinity.

By basic development of composed fractional we get that $\mu_2 \circ \Pi \circ \mu_1(\infty) =$

$$\frac{(d_1a_{q+1}+d_2b_{q+1})c_1^{q+1}+(d_1a_q+d_2b_q)c_1^qc_3+(d_1a_1+d_2b_1)c_1c_3^q+(d_1a_0+d_2b_0)c_3^{q+1}}{(d_3a_{q+1}+d_4b_{q+1})c_1^{q+1}+(d_3a_q+d_4b_q)c_1^qc_3+(d_3a_1+d_4b_1)c_1c_3^q+(d_3a_0+d_4b_0)c_3^{q+1}}.$$

2. S–box of Kuznyechik and Streebog

We are about to study S-box used in Kuznyechik block cipher and Streebog hash function and its properties. In Section 2.1 we describe the block cipher and the hash function in details. Two other sections are based on paper by Perrin [Per19] and they are focused on various decompositions of the S-box in 2.2 and its properties in 2.3.

Paper [Per19] describes the problem of the S-box of Kuznyechik and Streebog, Perrin gives there two previously found decomposition of the S-box and introduce the notion of TKlog mapping. He also shows connection of TKlog to previous decompositions and how TKlog can be seen as a master decomposition to them. He deals with properties uncovered and explained by the TKlog structure in S-box. Last but not least he analyzes design of the S-box and possible motivation based on strong cryptographical properties.

2.1 Kuznyechik cipher and Streebog hash function

We will present the cipher and the hash function themselves for which π was designed. Both algorithms are a part of Russian cryptographic standard algorithms (called GOST algorithms). GOST standards are not available to the public freely, so we derive information from RFC7801 [DD13] and RFC6986 [Dol16].

Some functions and algorithms used in the cipher and the hash function demand several constants, which are not interesting for our purposes and therefore the only constant presented in attachments is the S-box itself in A.1. We may also skip some technical details about used functions and we make them available in Attachment A.3.

Kuznyechik and Streebog as ordinary block cipher and hash function operate on Boolean vector spaces. Basic bit operations used in description are listed in Notation 3. In the process of computation they uses also structure of finite field $\mathbb{F}_N$ or iteger ring $\mathbb{Z}_N$ and transition between the Boolean vector space, the integer ring and the finite field. The correspondence is explained in more details in Section 1.1 and explicit transition function are mentioned in Notation 4.

Both Kuznyechik cipher and Streebog hash function use S–box π. Kuznyechik is a block cipher that uses substitution–linear network as a modification of SPN defined in Definition 10, where the S–box stands for the non–linear layer. Streebog, as a hash function, has not scheme of a block cipher, but it consumes the message with a round function, where the round function has a substitution-permutation network scheme defined in Definition 10. The S–box is again used as the non–linear layer.

2.1.1 Kuznyechik

The cipher was presented in June 2015 by the Federal Agency on Technical Regulating and Metrology in its Decree #749 as GOST R 34.12–2015. The real freely available source is RFC7801 [Dol16], published in march 2016, from which we draw majority of information.

Kuznyechik is a block cipher with blocks of size 128 bits and 10 rounds that process as typical SLN. The main key has 256 bits. Round keys of 128 bits, one for every round, are derived from the main key K using 32 fixed known constants in generating process described below in subsection 2.1.1.3.

Notation 8 might help us in orientation between notion of master key and round keys as we already saw it in Definition 10.

Notation 8. *For the key $K \in \mathbb{F}_2^{256}$, called master key, we will denote $K_1, \dots, K_{10} \in \mathbb{F}_2^{128}$ round keys derived from K by KeyGen process.*

2.1.1.1 Encryption

Process of encryption can be applied on blocks of 128 bits, it uses 10 round keys, one for each round. There are all layers (round key addition, substitution, linear transformation) in the first 9 rounds and the last round only applies the round key addition, as it is common for substitution network. Formally we define the encryption in Definition 21.

Definition 21. *Encryption of Kuznyechik block cipher is defined as mapping $E(a, K) : \mathbb{F}_2^{128} \longrightarrow \mathbb{F}_2^{128}$:*

$$\mathcal{E}_{Kuzn}(\overline{a}, K) := X_{K_{10}} \circ L \circ S \circ X_{K_9} \circ \dots \circ L \circ S \circ X_{K_1}(\overline{a}),$$

with K and $K_1, \dots, K_{10}$ following Notation 8 and particular layers defined as:

- *round key addition layer is defined as mapping $X_{K_i} : \mathbb{F}_2^{128} \to \mathbb{F}_2^{128}$ such that*

$$X_{K_i}(\overline{a}) = K_i \oplus \overline{a},$$

  for $K_i, \overline{a} \in \mathbb{F}_2^{128}$,

- *substitution layer is defined as mapping $S : \mathbb{F}_2^{128} \to \mathbb{F}_2^{128}$ such that*

$$S(\overline{a}) = \pi(a_0 \| \dots \| a_7) \| \dots \| \pi(a_{120} \| \dots \| a_{127}),$$

  for $\overline{a} = (a_0, \dots, a_{127})$ and $\pi : \mathbb{F}_2^8 \to \mathbb{F}_2^8$ the S-box defined in Attachment A.1,

- *linear layer is defined as mapping $L : \mathbb{F}_2^{128} \to \mathbb{F}_2^{128}$ such that*

$$L(\overline{a}) = R^{16}(\overline{a}),$$

  for R linear mapping defined in more details in Attachment A.3 and R^{16} stands for 16 application of mapping R.

2.1.1.2 Decryption

The decryption is naturally defined as application of inverses of each layer in each round in reverse order. Formally we define the decryption in Definition 22.

Definition 22. *Decryption of Kuznyechik block cipher is defined as mapping* $D(a, K) : \mathbb{F}_2^{128} \longrightarrow \mathbb{F}_2^{128}$

$$\mathcal{D}_{Kuzn}(\bar{b}, K) = X_{K_1} \circ S^{-1} \circ L^{-1} \circ X_{K_2} \circ \cdots \circ S^{-1} \circ L^{-1} \circ X_{K_{10}}(\bar{b}),$$

with K *and* $K_1, \ldots, K_{10}$ *following Notation 8.*

Observation 7 shows the form of inverse layers.

Observation 7.

- *Inverse of round key addition mapping* X_{K_i} *is* X_{K_i} *itself:*

  $$X_{K_i}^{-1} = X_{K_i},$$

  since

  $$X_{K_i} \circ X_{K_i}(\overline{a}) = K_i \oplus (K_i \oplus \overline{a}) = \overline{a},$$

  for every $K_i, a \in \mathbb{F}_2^{128}$*.*

- *Inverse of substitution mapping* S *is* $S^{-1} : \mathbb{F}_2^{128} \to \mathbb{F}_2^{128}$ *such that*

  $$S^{-1}(\bar{b}) = \pi^{-1}(b_0 \| \ldots \| b_7) \| \ldots \| \pi^{-1}(b_{120} \| \ldots \| b_{127}),$$

  for $\bar{b} = (b_0, \ldots, b_{127})$ *and* $\pi^{-1} : \mathbb{F}_2^8 \to \mathbb{F}_2^8$ *inverse of the S-box defined in Attachment A.1.*

- *Inverse of linear mapping* L *is* $L^{-1} : \mathbb{F}_2^{128} \to \mathbb{F}_2^{128}$ *such that*

  $$L^{-1}(\bar{b}) = (R^{-1})^{16}(\bar{b}),$$

  for R^{-1} *inverse of linear mapping* R *defined in more details in A.3. It is easily verifiable that* R *can be inverted.*

Definition 8 gives us also notion of a well defined cipher. We therefore demand that the encryption together with the decryption as we just defined them satisfy

$$\mathcal{D}_{Kuzn}(\mathcal{E}_{Kuzn}(\overline{m}, K), K) = \overline{m},$$

for every $\overline{m} \in \mathcal{P}_{Kuzn}$ messages and every $K \in \mathcal{K}_{Kuzn}$ keys. We can easily verify that this is really the case by straightforward rewriting $\mathcal{D}_{Kuzn}(\mathcal{E}_{Kuzn}(m, K), K)$ in terms of X_{K_i}, L, S as

$$\mathcal{D}_{Kuzn}(\mathcal{E}_{Kuzn}(m, K), K) = X_{K_1} \circ \cdots \circ S^{-1} \circ L^{-1} \circ X_{K_{10}} \circ X_{K_{10}} \circ L \circ S \circ \cdots \circ X_{K_1}(m)$$

and we find that all transformation are simplified, we get the condition satisfied and Kuznyechik can be considered as a well define cipher.

Observation 8. *To fulfill the condition on* $\mathcal{D}_{Kuzn}, \mathcal{E}_{Kuzn}, K$ *above we need only* X_{K_i}, S^{-1} *and* L^{-1} *to be the left inverse to* X_{K_i}, S *and* L *respectively.*

2.1.1.3 Key generation

As we already mentioned above, the secret key has 256 bites and 10 round keys, each with 128 bites, are derived from the master key by KeyGen generating process.

Algorithm 1 shows how the round keys are derived. We use L, S, X from Definition 21, a bit operation Split_2 from Notation 3 and transition function from integer representation of a number to its vectorial representation V_{128} from Notation 4.

Algorithm 1: Kuznyechik KeyGen

```
   input  : $K$ master key
   output: $K_1, \dots, K_{10}$ round keys for master key $K$
1  for $i \leftarrow 1, \dots, 32$ do
       $C_i = L(\text{V}_{128}(i))$
   end
2  $K_1, K_2 \leftarrow Split_2(K)$
3  for $i \leftarrow 1, \dots, 4$ do
       $K' \leftarrow K_{2i-1}, K'' := K_{2i}$
       for $j \leftarrow 1, \dots, 8$ do
           temp $\leftarrow K'$
           $K' \leftarrow (LSX_{C_{8(i-1)+j}}(K')) \oplus K''$
           $K'' \leftarrow$ temp
       end
       $K_{2i+1} \leftarrow K', K_{2i+2} \leftarrow K''$
   end
4  return $K_1, \dots, K_{10}$
```

As we can see, in step 1 Algorithm 1 uses 32 vector constants $C_1, \dots, C_{32}$ that do not depend on K. Value of the master key K come in by step 2 and gives the value to first two round keys K_1, K_2.

Figure 2.1 shows how we can imagine one evaluation of inner loop with $k := 8(i-1)+j$, which computes value of following pair of round keys by repetition for $j = 1. \dots, 8$.

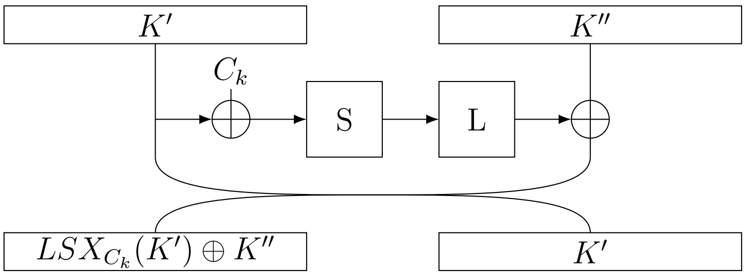

Figure 2.1: Main function used in key generation algorithm

Figure 2.2 on the other hand shows all 8 evaluation of inner loop as one evaluation of outer loop i.e. process of generation of K_3, K_4 from K_1, K_2. For $K_5, \dots, K_{10}$ the process works similarly. The initial state i.e. value of K', K'' is the last pair of generated K_{2i+1}, K_{2i+2} as well as a set of 8 used constants $C_{8(i-1)+1}, \dots, C_{8(i-1)+8}$ for new i.

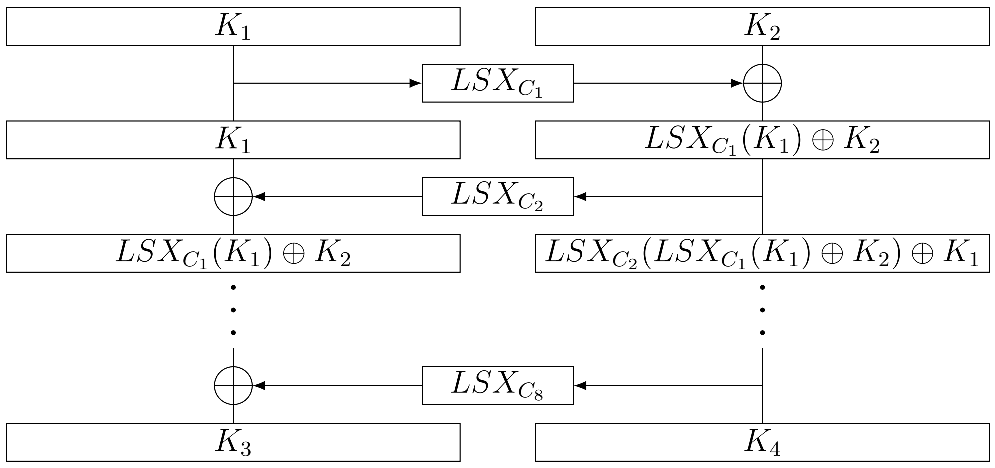
K_1
K_2
LSX_{C_1}
K_1
$LSX_{C_1}(K_1) \oplus K_2$
LSX_{C_2}
$LSX_{C_1}(K_1) \oplus K_2$
$LSX_{C_2}(LSX_{C_1}(K_1) \oplus K_2) \oplus K_1$
LSX_{C_8}
K_3
K_4

Figure 2.2: Setting up a pair of round keys

Remark. Over all we can see that the S-box π is used in the process of encryption as well as in the keys generation, always as a part of substitution-linear layer. Therefore any progress in analysis of π might be applicable on both schemes.

2.1.2 Streebog

The hash function was presented in the standard GOST R 34.11–2012 in January 2013 which replace the standard GOST R 34.11–94. The new standard was introduced by Decree #216 of the Federal Agency on Technical Regulating and Metrology in August 2012. The source, we used to get majority of information, is [DD13].

Definitions of functions and algorithms in the following subsection may need several constants, which are not interesting for our purposes and therefore the only constant presented in attachments is the S-box itself in A.1. Auxiliary function are defined in Notation 3 and 4.

2.1.2.1 Hashing process

Main hashing process can be divided into 3 phases:

1. Initialisation of variables,
2. Consumation of message by 512-bit blocks.
3. Consumation of the last block.

Initialisation phase sets up default value of variables, that are modified later during the process. Initialised value do not depend on consummated message.

In the middle phase we use as we call it the round function. We always take a block of 512 bits from the message and apply several operations. The round function that uses a substitution-permutation network is described in details in next section 2.1.2.2. It is the phase where the S-box π is used and therefore the step most interesting for us.

We complete the process by consummation of the last block of the message which can be shorter than 512 bits and therefore it is first bit padded. Last steps are similar to those in the middle phase again with the round function.

Algorithm 2: Hash function Streebog

```
   input  : $M$ the message
   output: $h$ the hash of $M$
   (initialisation of variables)
1  $h \leftarrow \mathrm{V}_{512}(0)$, $N \leftarrow \mathrm{V}_{512}(0)$, $\epsilon \leftarrow \mathrm{V}_{512}(0)$
   (consummation of message M by 512bit blocks)
2  while $|M| > 512$ do
       $m \leftarrow \mathrm{LSB}_{\mathbf{512}}(M)$
       $M' \leftarrow \mathrm{MSB}_{|\mathbf{M}|-\mathbf{512}}(M)$
       $h \leftarrow \mathbf{g_N}(h, m)$
       $N \leftarrow \mathrm{V}_{\mathbf{512}}(\mathrm{Int}_{\mathbf{512}}(N)[+]_{2^{512}}512)$
       $\epsilon \leftarrow \mathrm{V}_{\mathbf{512}}(\mathrm{Int}_{\mathbf{512}}(\epsilon)[+]_{2^{512}}\mathrm{Int}_{\mathbf{512}}(m))$
       $M \leftarrow M'$
   end
   (consummation of the last, probably shorter, block)
3  $m \leftarrow \overbrace{0\|\ldots\|0}^{511-|M|}\|1\|M_1\|\ldots\|M_{|M|}$
4  $h \leftarrow \mathbf{g_N}(h, m)$
5  $N \leftarrow \mathrm{V}_{\mathbf{512}}(\mathrm{Int}_{\mathbf{512}}(\epsilon)[+]_{2^{512}}\mathrm{Int}_{\mathbf{512}}(m))$
6  return $\mathbf{g_0}(\mathbf{g_0}(h, N), \epsilon)$
```

Algorithm 2 shows the process of calculation of the hash h from the message $M \in \mathbb{F}_2^*$. The algorithm uses auxiliary functions $\mathrm{LSB}_n, \mathrm{MSB}_n, \mathrm{V}_n, \mathrm{Int}_n$ defined and explained in Notation 4 and 3, the "+" operation is operating as natural addition in $\mathbb{Z}_{2^{512}}$.

For simplicity we mention only the 512-bit variant. The other 256-bit variant works pretty much the same, there is only chosen a different initialisation vector for h in the initialisation phase and 256 most significant bits are returned in the last step.

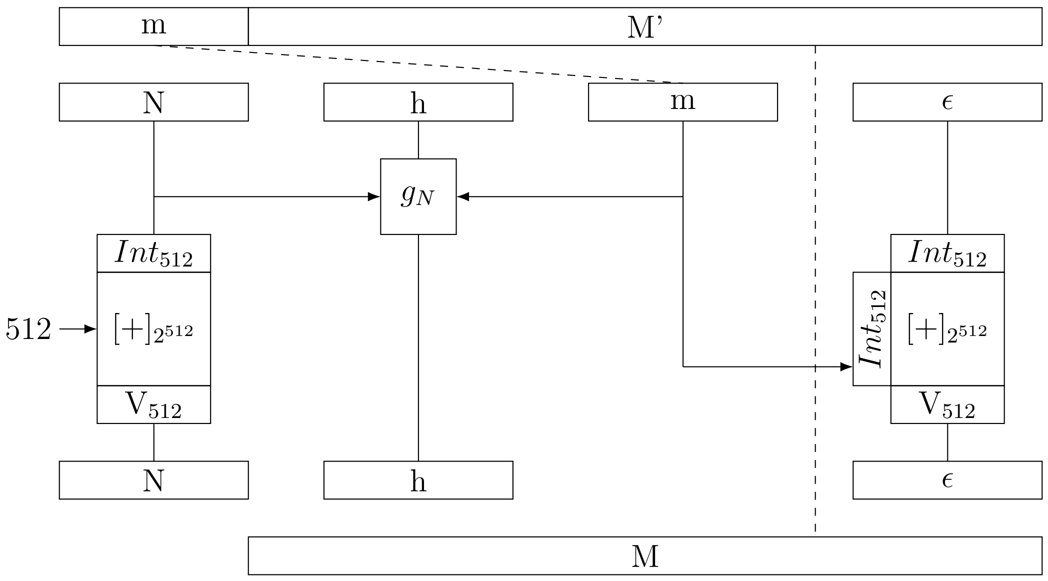

Figure 2.3: One round of consummation of message M by blocks of 512 bits

To describe more what goes really on in the middle phase and what is the role of N and ϵ in relation with value h, we show how we can imagine one evaluation of the while cycle in step 2 in figure 2.3. As we can see in the figure, we work primary with 4 variables and each of them is somehow modified and some of them are used to modify the others.

Section 2.1.2.2 is dedicated to so called round function g_N, which is the main former of the hash value h.

2.1.2.2 Round function $\mathbf{g_N}$

Function called round function that modifies value of h in each evaluation of the while loop is used in hashing process in step 2 in Algorithm 2. It takes N, h, m as parameters and outputs new value of h according to Definition 23.

Definition 23. *We define* $\mathbf{g_N} : \mathbb{F}_2^{512} \times \mathbb{F}_2^{512} \longrightarrow \mathbb{F}_2^{512}$ *with* $N \in \mathbb{F}_2^{512}$ *to be called the* ***round function of Streeboog*** *that works as:*

$$g_N(h, m) = E(LPS(h \oplus N), m) \oplus h \oplus m,$$

where $E(m, K) : \mathbb{F}_2^{512} \times \mathbb{F}_2^{512} \longrightarrow \mathbb{F}_2^{512}$ *such that:*

$$E(m, K) = X_{K_{13}} LPSX_{K_{12}} ... LPSX_{K_1}(m),$$

with $K_1, \ldots, K_{13}$ *are derived from* K *by Algorithm 3.*

As we already mentioned, round function g_N and especially the inner function E is a substitution–permutation network with a linear layer extra and 13 rounds and 13 round keys. Each but last round consist of every mentioned layer and the last round contains only the key addition.

The substitution layer S that uses the S-box π is defined just as in Definition 21 as well as the key addition X_{K_i}. Other mappings from Definition 23 (i.e. layers of the network) are defined in Definition 24.

Definition 24. *We define layers of* g_N*:*

- *The permutation layer* $P : \mathbb{F}_2^{512} \longrightarrow \mathbb{F}_2^{512}$ *defined as*

  $$P(\overline{a}) := a_{\tau(63)} \| \ldots \| a_{\tau(0)},$$

  for $a_{63}, \ldots, a_0 = \mathrm{Split}_{64}(\overline{a})$ *and* $\tau : \mathbb{Z}_{64} \longrightarrow \mathbb{Z}_{64}$ *a permutation of indexes defined as*

  $$\tau(x) := 8 \cdot (x \ mod\ 8) + (x \ div\ 8),$$

  i.e. $\tau(x) = 8v + u$ *where* $x = 8u + v$ *for* $u, v \in \mathbb{N}, \ v < 8$.

- *The linear layer* $L : \mathbb{F}_2^{512} \longrightarrow \mathbb{F}_2^{512}$ *defined as:*

  $$L(\overline{a}) := L(a_7 \| \ldots \| a_0) = l(a_7) \| \ldots \| l(a_0),$$

  for $a_7, \ldots, a_0 = \mathrm{Split}_8(\overline{a})$ *and* $l : \mathbb{F}_2^{64} \longrightarrow \mathbb{F}_2^{64}$ *defined as:*

  $$l(\overline{a}) := A\overline{a},$$

  with matrix $A \in \mathbb{F}_2^{64 \times 64}$ *defined in [DD13].*

Remark. We add a comment to both mappings in Definition 24:

- P could be redefined to be an application of a block matrix $P \in \mathbb{F}_2^{512\times512}$ partitioned into 64×64 blocks of size 8×8, where $P = \{P_{i,j}\}_{i,j=0}^{63}$, $P_{i,j} \in \mathbb{F}_2^{8\times8}$ defined as

$$P_{i,j} = \begin{cases} I_8 & \text{if } i = 8\cdot(j \bmod 8) + (j \operatorname{div} 8) \\ 0_8 & \text{otherwise} \end{cases}$$

  with I_8 identity matrix of size 8 and 0_n null matrix of size 8.

- L could be redefined to be an application of a block diagonal matrix $\mathbf{L} \in \mathbb{F}_2^{512\times512}$ with 8 blocks on diagonal of size 64×64, where $\mathbf{L} = \operatorname{diag}\{L_i\}_{i=0}^{7}$, $L_i \in \mathbb{F}_2^{64\times64}$ defined as

$$L_i = A, \forall i : 0 \leq i \leq 7.$$

Split and $\|$ from Definition 24 can be replaced by large matrices but such large matrices are not usable in practice. Especially if there is a way how to implement P and L in a form more suitable for today's computers.

In Figure 2.3 we can see how $g_N(h, m)$ can be written in a diagram.

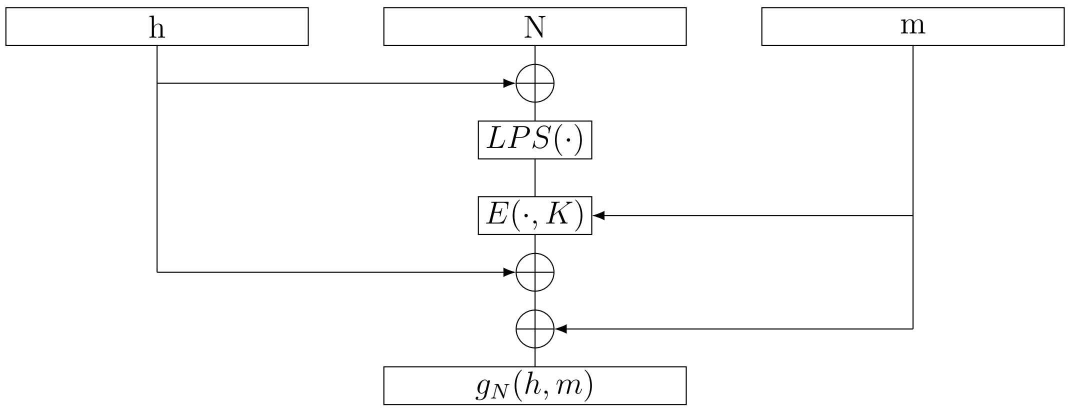

Figure 2.4: Function g_N

The parameter $K \in \mathbb{F}_2^{512}$ in function E has role of key and values $K_i \in \mathbb{F}_2^{512}$ for $i = 1, \ldots, 13$ can be regarded as round keys. K_i are generated from K by Algorithm 3. Analogously to Notation 8 we can call K and $K_1, \ldots, K_{13}$ master key and round keys respectively. Mappings S, P, L are used the same as for g_N.

Algorithm 3: Generation of round keys in g_N

```
  input  : $K \in \mathbb{F}_2^{512}$ master key
  output: $K_1, \ldots, K_{13} \in \mathbb{F}_2^{512}$ round keys
1 define constants $C_i$ as in [DD13]
2 $K_1 \leftarrow K$
3 for $i \leftarrow 2, \ldots, 13$ do
      $K_i \leftarrow L \circ P \circ S(K_{i-1} \oplus C_{i-1})$
  end
4 return $K_1, \ldots, K_{13}$
```

Remark. This key generation process can be seen as another substitution permutation network (without last round key addition), where key addition is actually known constant addition and after the *i–th* round we store current value as K_{i-1}.

Remark. Looking at the round function with E mapping and its key generation we can see that again the S-box π is present at multiple places and therefore, as well as in Kuznyechik, any progress in analysis of π might be applicable on all places.

2.2 Decompositions of the S–box π

Motivation of defining TKlog structure was to find a 'master' decomposition of S–box π. Two such decompositions was already found. The first one by Biryukov et al. presented in [BPU16] and second one by Perrin and Uduvenko presented in [PU16], but they have noting in common at first glance.

Firstly, we will present the definition of TKlog itself and in the second part of this section we will describe two previously found decompositions and how TKlog structure is related to them.

2.2.1 TKlog

The notion of TKlog structure is one of the main results of Perrin and the definition can be found in [Per19][Section 3.1]. Analyzing every branches of definition we might be confused by the notation.

For example mapping κ is defined to have image in $\mathbb{F}_{2^{2m}}$ in Perrin's Definition 1, while we give an example of κ for π with image in $\mathbb{F}_2^{2m}$. Additionally Perrin's Definition 1 puts elements of $\mathbb{N}$ as argument to κ while κ have domain $\mathbb{F}_2^m$.

To clarify the situation we reformulate the process of computation in Algorithm 4 by explicit expression of every manipulations. Perrin comments the situation around transitions between different structures in [Per19][Section 2.1] but lately in main definition he does not mention those transformation.

Every mapping from the algorithm is explained in Notation 3, 4 or Definition 25. We take a TKlog instance $\tau_{\kappa,s}$ as a mapping defined by a lookup table over $\mathbb{Z}_{2^{2m}}$ as π is defined in Attachment A.1.

Remark. In Algorithm 4 we use arithmetical operations mod, div, +, - and $\cdot$ taken over $\mathbb{N}$. Operation $+_F$ is taken as standard addition over $\mathbb{F}_{2^{2m}}$.

Remark. We need to briefly comment the situation that starts at step 3:

1. after step 3 k is by Definition 25 in $\mathbb{N}$, but since $\alpha^{2^{2m}-1} = 1 = \alpha^0$ we have $k \in \{1, \dots, 2^{2m} - 1\}$,
2. after step 4 we have $j \in \{0, \dots, 2^m - 1\}$ and $i \in \{0, \dots, 2^m\}$,
3. step 6 happens in case $i = 0 \Rightarrow k = j \cdot (2^m + 1)$ and therefore since point 1 we have $j \in \{1, \dots, 2^m - 1\}$ and $2^m - j \in \{1, \dots, 2^m - 1\} \subset \mathrm{dom}(\mathrm{V}_m)$ as in Notation 4, additionally as $j \neq 0$ we return another value than in step 2,
4. steps 7 - 11 happen in case $i \neq 0$ and therefore $2^m - i \in \{0, \dots, 2^m - 1\} = \mathrm{dom}(\mathrm{V}_m)$, we have also $i > 0 \Rightarrow j < 2^m - 1$ and therefore $j \in \{0, \dots, 2^m - 2\} = \mathrm{dom}(s)$.

Algorithm 4: TKlog mapping

```
   input  : $a \in \mathbb{Z}_{2^{2m}}$
   output: $\tau_{\kappa,s}(a) \in \mathbb{Z}_{2^{2m}}$
   given  : $\kappa : \mathbb{F}_2^m \longrightarrow \mathbb{F}_2^{2m}$ an affine mapping
            $s : \mathbb{Z}_{2^m-1} \longrightarrow \mathbb{Z}_{2^m-1}$ a permutation
1  if $a = 0$ then
2      return $\mathrm{Int}_{2m} \circ \kappa \circ \mathrm{V}_m(0)$
   end
3  $k \leftarrow \log_F \circ \mathrm{F}_\mathrm{V} \circ \mathrm{V}_{2m}(a)$
4  $i \leftarrow k \bmod (2^m + 1)$, $j \leftarrow k \operatorname{div} (2^m + 1)$
5  if $i = 0$ then
6      return $\mathrm{Int}_{2m} \circ \kappa \circ \mathrm{V}_m(2^m - j)$
   else
7      $\beta \leftarrow \mathrm{F}_\mathrm{V} \circ \kappa \circ \mathrm{V}_m(2^m - i)$
8      $\gamma \leftarrow \exp_F((2^m + 1) \cdot s(j))$
9      $\delta \leftarrow \beta +_F \gamma$
10     $b \leftarrow \mathrm{Int}_{2m} \circ \mathrm{V}_F(\delta)$
11     return $b$
   end
```

We already touched uniqueness of outputs from different branches of the algorithm in point 3 of previous remark. The question whether Algorithm 4 returns different outputs for different inputs is answered by TKlog being always a permutation as Perrin gives an inverse mapping. We handle uniqueness of outputs in Observation 9 in a more constructive way.

Observation 9. *Let κ be $\kappa : \mathbb{F}_2^m \longrightarrow \mathbb{F}_2^{2m}$, $\kappa(\overline{a}) = A\overline{a} \oplus \overline{b}$. We show that each branch of Algorithm 4 returns different outputs.*

- *Branches that lead to step 2 and 6 are discussed in step 3 of previous remark*
- *Step 11 returns the same value as step 2 if only, for $i, j, \beta, \gamma, \delta$ as in Algorithm 4 and $i' = 2^m - i$, $j' = s(j)$, the following holds:*

$$\begin{aligned}
\mathrm{Int}_{2m} \circ \kappa \circ \mathrm{V}_m(0) &= \mathrm{Int}_{2m} \circ \mathrm{V}_F(\delta) \\
\kappa \circ \mathrm{V}_m(0) &= \mathrm{V}_F(\delta) \\
\mathrm{F}_\mathrm{V} \circ \kappa \circ \mathrm{V}_m(0) &= \delta \\
\mathrm{F}_\mathrm{V} \circ \kappa \circ \mathrm{V}_m(0) &= \gamma +_F \beta \\
\mathrm{F}_\mathrm{V} \circ \kappa \circ \mathrm{V}_m(0) &= \exp_F((2^m + 1) \cdot s(j)) +_F \mathrm{F}_\mathrm{V} \circ \kappa \circ \mathrm{V}_m(2^m - i) \\
\mathrm{F}_\mathrm{V} \circ \kappa \circ \mathrm{V}_m(0) &= \exp_F((2^m + 1) \cdot j') +_F \mathrm{F}_\mathrm{V} \circ \kappa \circ \mathrm{V}_m(i') \\
\kappa \circ \mathrm{V}_m(0) &= \mathrm{V}_F \circ \exp_F((2^m + 1) \cdot j') \oplus \kappa \circ \mathrm{V}_m(i') \\
\overline{b} \oplus \mathrm{V}_m(0) &= \mathrm{V}_F \circ \exp_F((2^m + 1) \cdot j') \oplus \overline{b} \oplus A \circ \mathrm{V}_m(i') \\
\mathrm{V}_m(0) &= \mathrm{V}_F \circ \exp_F((2^m + 1) \cdot j') \oplus A \circ \mathrm{V}_m(i') \\
\mathrm{F}_\mathrm{V} \circ \mathrm{V}_m(0) &= \exp_F((2^m + 1) \cdot j') +_F \mathrm{F}_\mathrm{V} \circ A \circ \mathrm{V}_m(i')
\end{aligned}$$

Let us denote $Q = \langle\alpha^{2^m+1}\rangle \cup \{0\} \simeq \mathbb{F}_{2^m}$. *Mapping* κ *and matrix* A *is defined to satisfy*

$$\mathbb{F}_{2^{2m}} = \{x_F +_F \mathrm{F_V}(\kappa(\overline{x}_\kappa) +_F \kappa(0)) | x_F \in Q, x_\kappa \in \mathbb{F}_2^m\} \tag{2.1}$$

$$i.e.\ \mathbb{F}_{2^{2m}} = \{x_F +_F \mathrm{F_V} \circ A(\overline{x}_\kappa) | x_F \in Q, x_\kappa \in \mathbb{F}_2^m\} \tag{2.2}$$

Since A *has full column rank and*

$$|\mathbb{F}_{2^{2m}}| = 2^{2m} = 2^m \cdot 2^m = |Q| \cdot |\mathbb{F}_2^m|$$

and therefore every element from $\mathbb{F}_{2^{2m}}$ *is unequivocally represented by a pair* $(x_F, \overline{x}_\kappa)$. *Specially we have only one way how to write*

$$0 = x_F +_F \mathrm{F_V} \circ A(\overline{x}_\kappa) \text{ for } x_F = 0 \in Q \text{ and } \overline{x}_\kappa = (0, \dots, 0) \in \mathbb{F}_2^m.$$

This situation does not correspond to our situation above

$$\mathrm{F_V} \circ \mathrm{V}_m(0) = \exp_F((2^m + 1) \cdot j') +_F \mathrm{F_V} \circ A \circ \mathrm{V}_m(i')$$

since $i' = 2^m - i \in \{1, \dots, 2^m - 1\}$ *and* $\exp_F((2^m + 1) \cdot j') \in Q \setminus \{0\}$.

- *Analogously we can prove that steps 6 and 11 does not return any same output by expression*

$$\begin{aligned}
\mathrm{Int}_{2m} \circ \kappa \circ \mathrm{V}_m(2^m - j) &= \mathrm{Int}_{2m} \circ \mathrm{V}_F(\delta) \\
&\vdots \\
\overline{b} \oplus A \circ \mathrm{V}_m(2^m - j) &= \mathrm{V}_F \circ \exp_F((2^m + 1) \cdot j') \oplus \overline{b} \oplus A \circ \mathrm{V}_m(i') \\
A \circ \mathrm{V}_m(2^m - j) &= \mathrm{V}_F \circ \exp_F((2^m + 1) \cdot j') \oplus A \circ \mathrm{V}_m(i') \\
V_m(0) &= \mathrm{V}_F \circ \exp_F((2^m + 1) \cdot j') \oplus \\
&\quad \oplus A(\mathrm{V}_m(2^m - j) \oplus \mathrm{V}_m(i')) \\
\mathrm{F_V} \circ V_m(0) &= \exp_F((2^m + 1) \cdot j') +_F \\
&\quad +_F \mathrm{F_V} \circ A(\mathrm{V}_m(2^m - j) \oplus \mathrm{V}_m(i'))
\end{aligned}$$

and we come across the same thing: $\exp_F((2^m + 1) \cdot j') \in Q \setminus \{0\}$.

The original definition by Perrin uses elements from 4 different types of structures:

- $\mathbb{F}_{2^{2m}}$ the finite field as domain and image of $\tau_{\kappa,s}$,
- $\mathbb{Z}_{2^m-1}$ and $\mathbb{Z}_{2^{2m}-1}$ additive group e.g. as domain and image of the permutation s,
- $\mathbb{F}_2^m$ and $\mathbb{F}_2^{2m}$ the vector spaces, κ operates on,
- $\mathbb{N}$ natural numbers, where operations as mod, div etc. are proceeded.

In Notation 4 we already introduced transition $\mathbb{F}_2^d \rightleftarrows \mathbb{Z}_{2^d}$ and $\mathbb{F}_2^d \rightleftarrows \mathbb{F}_{2^d}$. We introduce remaining transitions in Definition 25.

Definition 25. *Let α be the generator and p the minimal polynomial of $\mathbb{F}_{2^d}$ as in Remark after Notation 1, We define following mappings:*

- $\log_F : \mathbb{F}_{2^d} \longrightarrow \mathbb{N}$ *defined as*

$$\log_F(z) := k,$$

  where k is the lowest such that α^k mod $p = z$.

- $\exp_F : \mathbb{N} \longrightarrow \mathbb{F}_{2^d}$ *defined as*

$$\exp_F(k) := \alpha^k \text{ mod } p.$$

With Definition 25 we can complete Figure 1.1 with one structure and two transformations more in Figure 2.5.

$$\begin{tikzcd}
\mathbb{Z}_{2^d} \arrow[r, shift left, "\mathrm{V}_d"] & \mathbb{F}_2^d \arrow[l, shift left, "\mathrm{Int}_d"] \arrow[r, shift left, "\mathrm{F}_\mathrm{V}"] & \mathbb{F}_{2^d} \arrow[l, shift left, "\mathrm{V}_F"] \arrow[r, shift left, "\log_F"] & \mathbb{N} \arrow[l, shift left, "\exp_F"] \arrow[r, shift left, "\mathrm{mod}"] & \mathbb{Z}_{2^m-1} \arrow[l, shift left]
\end{tikzcd}$$

Figure 2.5: Transitions between structures

Perrin also gives an example of κ and the permutation s for $\tau_{\kappa,s} = \pi$. We reformulate the example as well. Permutation s is probably expressed in its best form by a look up table, but κ can be expressed as an affine mapping by a matrix multiplication and vector addition instead of expression of images on basal vectors in hexadecimal form.

Example. The function $\kappa : \mathbb{F}_2^4 \longrightarrow \mathbb{F}_2^8$ for our S–box π can be defined as:

$$\kappa(x) = Ax + b \text{ with } A = \begin{pmatrix} 0&0&0&0\\ 0&0&0&0\\ 0&1&1&1\\ 1&0&0&1\\ 0&0&0&0\\ 0&1&1&0\\ 1&1&0&0\\ 0&0&0&0 \end{pmatrix} \text{ and } b = \begin{pmatrix} 1\\1\\1\\1\\1\\1\\0\\0 \end{pmatrix}$$

Matrix A represents the linear part of κ mapping 4-dimensional vectors to 8-dimesional vectors. However, A have exactly 4 linearly independent rows and vector addition of b does not change dimension of the image and therefore image of κ is a 4-dimensional vector subspace of $\mathbb{F}_2^8$.

2.2.2 Previous decompositions

Perrin briefly describes two decompositions published previously in [BPU16] and [PU16] and he also shows how the TKlog structure can be the 'master' decomposition for the others. We will present both decompositions as well as the link between them and relation to TKlog structure.

2.2.2.1 TU–decomposition

The first decomposition, that he mention, is the TU–decomposition presented by Biryukov at al. in [BPU16]. The main thing they show is that S-box π is affine–equivalent to a permutation of $(\mathbb{F}_2^4)^2$:

$$(x, y) \mapsto (T_y(x), U_{T_y(x)}(y)).$$

The notion of affine-equivalency is defined in Definition 6.

Permutations T_y and U_x are described as decomposition to simpler functions, where each of then is defined in [BPU16] and Perrin is also giving expressive graphical scheme in [Per19].

Algorithm 5 shows process of the application of S-box π decomposed into several steps as specified by TU decomposition. We describe functions used in the TU decomposition in Notation 9 while we take $\mathrm{Int}_k, \mathrm{V}_k, \mathrm{Split}_2$ and $\|$ from Notation 3 and 4.

Notation 9.

- *$\alpha, \omega : \mathbb{F}_2^8 \longrightarrow \mathbb{F}_2^8$ are linear permutations such that*

$$\{\alpha^{-1}(x\|0); x \in \mathbb{F}_2^4\} = \{\omega(0\|x); x \in \mathbb{F}_2^4\}.$$

- *$\nu_0, \nu_1, \sigma : \mathbb{F}_2^4 \longrightarrow \mathbb{F}_2^4$ are permutations.*
- *$\phi : \mathbb{F}_2^4 \longrightarrow \mathbb{F}_2^4$ is a function such that $\phi(x) \neq 0 \forall x \in \mathbb{F}_2^4$.*
- *operations $\cdot,^{-1}$ are standard operations on $(\mathbb{Z}_{2^4-1}, \cdot)$.*

Algorithm 5: TU decomposition

```
  input  : $a \in \mathbb{F}_2^8$
  output: $\pi(a) \in \mathbb{F}_2^8$
1 $b \leftarrow \alpha(a)$
2 $x, y \leftarrow \mathrm{Split}_2(b)$
3 if $y = (0, \dots, 0)$ then
  |  $t \leftarrow \nu_0(x)$
  else
  |  $t \leftarrow \nu_1(\mathrm{V}_4(\mathrm{Int}_4(x) \cdot \mathrm{Int}_4(y)^{-1})$
  end
4 $u \leftarrow \sigma \circ \mathrm{V}_4(\mathrm{Int}_4(\phi(t)) \cdot \mathrm{Int}_4(y))$
5 $v \leftarrow t\|u$
6 $w \leftarrow \omega(v)$
7 return $w$
```

Observation 10. *Connecting the notation from [Per19] and Algorithm 5, we can see that*

- *step 3 corresponds to the permutation $T_y(x)$,*
- *step 4 corresponds to the permutation $U_{T_y(x)}(y)$,*
- *steps 1 and 6 provide the affine equivalency choosing*

  - *$A_1 := \alpha$,*
  - *$A_2 := \omega$ and*
  - *$A_3 := 0$ the null mapping on $\mathbb{F}_2^8$,*

- *steps 2 and 5 provide bit operations as separation of the input into halves and concatenation of them back again.*

2.2.2.2 Decomposition with discrete logarithm

Second decomposition, the one presented in [PU16] by Perrin and Udovenko, is based on the discrete logarithm. The discrete logarithm is actually a 'pseudo' discrete logarithm, where we choose an integer j, we define α^j to have 0 as image and we shift images of following exponents by 1. We define such mapping in Definition 26.

Definition 26 (Pseudo discrete logarithm)**.** *Let K be a finite field $K = \mathbb{F}_{2^n}$ as in Remark after Notation 1. We define* ***pseudo discrete logarithm*** *in K with parameter $j \in \{0, \dots, 2^n - 2\}$ as mapping $\mathrm{PDL}_{j,K} : K \longrightarrow \mathbb{Z}_{2^n-1}$*

$$\mathrm{PDL}_{j,K}(x) := \begin{cases} j, & \textit{for } x = 0 \\ i, & \textit{for } x = \alpha^i \textit{ and } i < j \\ i+1, & \textit{for } x = \alpha^i \textit{ and } i \geq j. \end{cases}$$

Example. $\mathrm{PDL}_{j,\mathbb{F}_{2^8}} : \mathbb{F}_{2^8} \longrightarrow \mathbb{Z}_{2^8-1}$ maps

$$(1, \alpha^1, \dots, \alpha^{j-1}, 0, \alpha^j, \dots, \alpha^{2^8-2}) \mapsto (0, 1, \dots, 2^8 - 1).$$

Algorithm 6 shows process of the application of S-box π decomposed into several steps as specified by the log-based decomposition. Particular layers presented in Notation 10 have all different domains and images and therefore conversions between them such as V_n and Int_n defined in Notation 4 have to be applied, so that one layer can be followed by the other. All layers are specified in [PU16] as well as graphical scheme presented in [Per19].

Notation 10.

- *$A : \mathbb{Z}_{2^4} \times \mathbb{Z}_{2^4} \longrightarrow \mathbb{Z}_{2^4} \times \mathbb{Z}_{2^4}$ are modular arithmetical operations.*
- *$q' : \mathbb{F}_2^4 \longrightarrow \mathbb{F}_2^4$ is a permutation.*
- *$\omega' : \mathbb{F}_2^8 \longrightarrow \mathbb{F}_2^8$ is a permutation.*

Algorithm 6: Decomposition using pseudo descrete logarithm

```
  input  : $a \in \mathbb{F}_2^8$
  output: $\pi(a) \in \mathbb{F}_2^8$
1 $b \leftarrow \mathrm{V}_8 \circ \mathrm{PDL}_{j,\mathbb{F}_{2^8}} \circ \mathrm{Int}_8(a)$
2 $x, y \leftarrow \mathrm{Split}_2(b)$
3 $u, v \leftarrow A(\mathrm{Int}_4(x), \mathrm{Int}_4(y))$
4 $w \leftarrow u \| v$
5 $z \leftarrow \omega' \circ q'^{-1}(z)$
6 return $z$
```

2.2.3 Link between decompositions

Perrin says in [Per19] that the missing link is exactly the TKlog structure and he shows how TKlog is related to both of them.

Relation to TU–decomposition is described by Theorem 2 in [Per19][Section 4.1]. The Theorem 2 followed by a proof says that a permutation with TKlog structure has always the TU–decomposition. The main idea of the proof is to perform case analysis for each step of both TU–decomposition and TKlog function algorithms.

Relation between TKlog and the decomposition with discrete logarithm is commented in [Per19] as natural since both use a discrete logarithm in some form. TKlog uses discrete logarithm in form of mapping α^n to $\kappa(A'(n)) + \textit{something}$ where A' are some arithmetical operations. The other decomposition uses the pseudo discrete logarithm as we define it in Definition 26 followed by certain arithmetical operations and another permutations. Pseudo discrete logarithm includes as well a kind of multiplexer dependent to the parameter j.

2.3 Properties of π as TKlog

TKlog structure has several properties and one can hope to use them to break the S–box. However, no attack was design yet. We will present two kind of interesting properties in the section. First one is partition–preserving property, which allows us to map one kind of cosets to another and it is let as an open problem, whether this property can be used against Streebog. The other kind of properties that we will mention are cryptographical properties of π namely, concerning linear and differential anomalies as we define them below.

2.3.1 Mapping cosets to cosets

We are about to explain the partition preserving property of TKlog mapping, where cosets for particular subgroups play a role. We will denote those subgroups as in Notation 11.

Notation 11. *Important subgroups will be denoted as:*

- $H_{mult} := \langle \alpha^{2^m+1} \rangle$ *multiplicative subgroup of* $(\mathbb{F}_{2^{2m}}, \cdot)$ *generated by* $\alpha^{2^m+1} \in (\mathbb{F}_{2^{2m}}, \cdot)$.
- $H_{add} := \langle \beta, \dots, \beta^m \rangle$ *additive subgroup of* $(\mathbb{F}_{2^{2m}}, +)$ *generated by* m *first powers of* $\alpha^{(2^m+1)}$ *mod* $p(\alpha) = \beta$.
- $H_{vect} := \{(a_0, \dots, a_{2m-1}) \in (\mathbb{F}_2, +)^{2m} | \sum_{i=0}^{2m-1} a_i \alpha^i \in H_{add}\}$ *a vector subspace of* $(\mathbb{F}_2, +)^{2m}$ *generated by vectors of coefficients of* $\beta, \dots, \beta^m$.

Observation 11. *Using Notation 11:*

- $H_{mult} = \{\alpha^{(2^m+1)j} | j \in \mathbb{Z}_{2^m-1}\} \simeq (\mathbb{Z}_{2^m-1}, +) \simeq (\mathbb{F}_{2^m}, \cdot)$ *since*

$$\alpha^{(2^m+1)(2^m-1)} = \alpha^{2^{2m}-1} = 1.$$

- $H_{add} = \{\sum_{i=0}^{m-1} a_i\beta^i | (a_0, \dots, a_{m-1}) \in (\mathbb{F}_2, +)^m\} \simeq (\mathbb{F}_2, +)^m \simeq (\mathbb{F}_{2^m}, +)$ *since the set of generators is a polynomial base in β of dimension m.*
- $H_{add} = H_{mult} \cup \{0\}$ *by definition.*
- $H_{vect} \simeq H_{add}$ *by definition.*

The main thing on partition–preserving property is presented by [Per19, Theorem1, Section 3.2]. The theorem is mentioned here as Theorem 13, but first we need an explanation of used terms and notation. We start with notion of cosets from basic algebra in Definition 27.

Definition 27. *Let* $\mathbf{G} = (G, *, ', e)$ *be a group,* $H \subseteq G$ *its subgroup and* $g \in G$. *We define* ***left coset of* H *in* G** *as*

$$g * H := \{g * h | h \in H\}.$$

Remark. As we work over commutative fields we can, without lost of generality, use strictly notion of left cosets for our purposes, because the commutativity makes the right and the left variant of the definition equivalent.

Remark. Usual notation of cosets do not use symbol of the operation between the element g and the subgroup H, because usually it is clear which operation is used. However, in our case, as we talk about both multiplicative and additive group of a field, it is more reliable to explicitly write the symbols to keep expressions evident.

Using Observation 11 and related Notation 11 we express decompositions of $\mathbb{F}_{2^{2m}}$ into cosets from [Per19][section 3.2] in Claim 12.

Claim 12. *Let* $m \in \mathbb{N}$ *and* $\mathbb{F}_{2^{2m}}$ *be a finite field, then we have two ways of decomposition of* $\mathbb{F}_{2^{2m}}$*:*

1.

$$\begin{aligned} \{0\} \cup (\mathbb{F}_{2^{2m}}, \cdot) =& (\bigcup_{k=0}^{2^m} \alpha^k \cdot H_{mult}) \cup \{0\} \\ =& (\bigcup_{k=1}^{2^m} \alpha^k \cdot H_{mult}) \cup H_{mult} \cup \{0\}, \end{aligned} \tag{2.3}$$

2.

$$\begin{aligned} (\mathbb{F}_{2^{2m}}, +) =& \bigcup_{w \in W_{add}} w + H_{add} \\ =& (\bigcup_{w \in W_{add}} w + (H_{add} \setminus \{0\})) \cup W_{add}. \end{aligned} \tag{2.4}$$

Proof.

1. As α is generator of $(\mathbb{F}_{2^{2m}}, \cdot)$, every element from $(\mathbb{F}_{2^{2m}}, \cdot)$ can be written as

$$alpha^i = \alpha^{k+(2^m+1)l} = \alpha^k \cdot (\alpha^{(2^m+1)})^l$$

for unique $i, k, l \in \mathbb{N}_0$ such that

$$i < 2^{2m} - 1,\ k < 2^m + 1,\ l < 2^m - 1,\ i = k + (2^m + 1)l$$

and therefore $\alpha^k \cdot (\alpha^{(2^m+1)})^l \in \alpha^k \cdot H_{\text{mult}}$ for some $k \in \{0, \ldots, 2^m\}$.

Second equality of 2.3 hold because we only take the coset $\alpha^0 \cdot H_{\text{mult}} = H_{\text{mult}}$ and we exclude it from the big union.

2. By Observation 1 and 11 $(\mathbb{F}_{2^{2m}}, +)$ and H_{add} are isomorphic to vector spaces of dimensions $2m$ and m respectively. There must exist a vector space, let us call it W_{vect}, such that

$$(\mathbb{F}_2, +)^{2m} = W_{\text{vect}} + H_{\text{vect}},\ \text{where } W \subsetneq (\mathbb{Z}_2, +)^{2m} \text{ and } \dim(W) = m$$

and therefore there must exist an additive subgroup of $(\mathbb{F}_{2^{2m}}, +)$, let us call it W_{add}, such that

$$(\mathbb{F}_{2^{2m}}, +) = W_{\text{add}} + H_{\text{add}}$$

as a direct sum. W_{add} must be generated by m linearly independent elements so that generators of W_{add} with generators of H_{add} give a generator set for $(\mathbb{F}_{2^{2m}}, +)$. This decomposition allows us to write $(\mathbb{F}_{2^{2m}}, +)$ as union of cosets $w + H_{\text{add}}$ because each element $v \in (\mathbb{F}_{2^{2m}}, +)$ can be written as

$$v = w \oplus a \text{ where } w \in W_{\text{add}} \text{ and } a \in H_{\text{add}}$$

Second equality of 2.4 holds because we only take one element $w + 0$ from each coset $w + H_{\text{add}}$ and store them in separate set behind the big union.

□

Notation 12 completes everything we need to reformulate Theorem 13 into Theorem 14 and to explain it right after.

Notation 12.

Let ψ be a mapping, A be a subset of domain of ψ, then

$$\psi(A) := \{\psi(a) |\ a \in A\}.$$

Analogously we will use notation $A * B = \{a * b | a \in A, b \in B\}$.

Theorem 13 uses exactly the formulation from [Per19] with Perrin's notation, while following theorem uses notation introduced above and notation from Section 2.2.1.

Theorem 13 ([Per19], Theorem 1)**.** *Let $\tau_{\kappa,s} : \mathbb{F}_{2^{2m}} \longrightarrow \mathbb{F}_{2^{2m}}$ be a valid TKlog instance, then the following always holds:*

$$\tau_{\kappa,s}(\mathbb{F}_{2^m}) = \kappa(\mathbb{F}_2^m),$$
$$\tau_{\kappa,s}(\alpha^i \odot \mathbb{F}_{2^m}^*) = \kappa(2^m - i) \oplus \mathbb{F}_{2^m}^* \text{ for } i \neq 0.$$

Theorem 14. *Let $\tau_{\kappa,s} : \mathbb{F}_{2^{2m}} \longrightarrow \mathbb{F}_{2^{2m}}$ be a valid TKlog instance, with $\kappa : \mathbb{F}_2^m \longrightarrow \mathbb{F}_2^{2m}$. Let $\log_{F'}(x) = \log_{\alpha^{2^m+1}}(x) = \log_F(x)/(2^m + 1)$, then the following always holds:*

$$\tau_{\kappa,s}(H_{mult} \cup \{0\}) = \mathrm{F_V} \circ \kappa \circ \mathrm{V}_m \circ \log_{F'}(H_{add}) \tag{2.5}$$

$$\tau_{\kappa,s}(\alpha^i \cdot H_{mult}) = \mathrm{F_V} \circ \kappa \circ \mathrm{V}_m(2^m - i) +_F (H_{add} \setminus \{0\}) \text{ for } i \in \{1, \ldots, 2^m\}. \tag{2.6}$$

Remark. Reformulation in Theorem 14 deserves brief comments:

- In 2.5 we enlarged the set in argument H_{mult} by the element 0, since $\tau_{\kappa,s}(0) \in \mathrm{F_V}(\mathrm{Im}(\kappa))$ as well as $\tau_{\kappa,s}(H_{\text{mult}}) \subsetneq \mathrm{F_V}(\mathrm{Im}(\kappa))$. Moreover it really corresponds to the first equality in Theorem 13, since $H_{\text{mult}} \cup \{0\} \simeq \mathbb{F}_{2^m}$.
- Equality 2.6 says that $\tau_{\kappa,s}$ maps multiplicative cosets of H_{mult} in $(\mathbb{F}_{2^{2m}}, \cdot)$ to something like additive cosets of H_{add} in $(\mathbb{F}_{2^{2m}}, +)$. The permutation s only permutes $\mathbb{Z}_{2^m-1} \simeq H_{\text{add}} \setminus \{0\}$.
- Although we do not have true additive cosets on the right side, we see evident structure, where the element 0 is always somehow specific case.
- $\mathbb{F}_{2^{2m}}$ is covered by Theorem 14, since in Claim 12 we have in 2.3 a partition of $\mathbb{F}_{2^{2m}}$ into a big union that is covered by left side of equality 2.6 and a set $H_{\text{mult}} \cup \{0\}$ behind the big union that is covered by left side of equality 2.5. Right sides of 2.6 and 2.5 cover the partition 2.4 analogously.

Observation 15. *With notation from proof of Claim 12 and $\kappa(\overline{x}) = A\overline{x} \oplus \overline{b}$ it holds that W_{vect} is an affine vector space*

$$W_{vect} = \kappa(0, \ldots, 0) + \mathrm{Im}(A).$$

Equalities 2.5, 2.6 and Observation 15 corresponds to how Perrin defines mapping κ to satisfy the condition 2.1 and 2.2. Without those condition we could not observe uniqueness of outputs in Observation 9.

2.3.1.1 Impact on Streebog

Consequences of the partition–preserving property are discussed in [Per19] especially how the S–box π interacts with the linear layer of Streebog. Special case for input $(0, \ldots, 0, x, 0, \ldots, 0)$ is discussed and he concludes that if $x \in \mathbb{F}^*_{2^m}$ then π is mapping a subfield to its multiplicative coset.

In the discussion at the end of the paper Perrin describe usual setting around the layers and interaction between them and conclude that this is quite unprecedented case. The unprecedency is caused by combination of the two things.

Firstly, by the fact that TKlog operates on one structure, while for example the discrete logarithm preserve partitions into cosets, but it maps elements from a multiplicative group into natural numbers.

Secondly, we have the fact that TKlog maps multiplicative cosets into additives cosets, so it is not the case similar to S–box in AES, where the linear layer breaks this property.

Overall Perrin did not find a way how to use this property to attack Kuznyechik or Streebog. He let it as an open problem, whether the property can be used as an advantage for an attack.

2.3.2 Cryptographical properties

First of all Perrin asks himself a question whether the S–box with TKlog properties could be chosen at random. He analyzed the cardinality of set of TKlog instances in comparision to cardinality of set of all permutations (or at least the affine ones) of the field $\mathbb{F}_{2^8}$. The probability of choosing a permutation with TKlog properties between affine permutations is about $\frac{1}{4000}$. Thus we can be suspicious about the randomness of choice declared by designers of π.

There is also another perspective presented by Perrin in [Per19]. If we admit the choice of π to be purposeful to get some strong cryptographic properties and we analyze linear and differential properties, we do not get any significant results. Perrin in [Per19] introduce the concept of anomaly of S–box, as we mention it in the following definition.

Definition 28. *For $F : \mathbb{F}_2^n \to \mathbb{F}_2^n$ a permutation, $u(F)$ differential uniformity of F, $l(F)$ linearity of F, $N_k(F)$ number of occurrences of k in* DDT_F *and $N'_k(F)$ sum of number of occurrences of $\pm k$ in the* LAT *of F. We define the **differential anomaly of F** as*

$$A_F^d = -\log_2(Pr[u(G) \leq u(F) \wedge N_{u(F)}(G) \leq N_{u(F)}(F)])$$

*and the **linear anomaly of F** as*

$$A_F^l = -\log_2(Pr[l(G) \leq l(F) \wedge N'_{l(F)}(G) \leq N'_{l(F)}(F)]),$$

with probability taken over all permutations G.

The Definition 28 uses several terms as DDT, LAT, differential uniformity and linearity, which we covered in the first chapter in Definitions 13 and 14. We can stop at the definition to explain the meaning. If we take the differential anomaly of F we can see its close relation to the DDT_F and DDT_G for every other permutation G. Since the number of possible G for fixed n is final, the probability in the definition is equal to the number of positive cases divided by the total number of permutations G.

Firstly, a positive case can come if G has the differential uniformity strictly lower than the one of F. It is because then the number of occurrences of $u(F)$ in DDT_G is equal to 0 by Definition 14 and therefore lower than number of occurrences of $u(F)$ in DDT_F.

$$u(G) < u(F) \Longrightarrow 0 = N_{u(F)}(G) < N_{u(F)}(F)$$

Also, a positive case come when the differential uniformity of G is the same as the differential uniformity of F and the number of occurrences of the value in DDT_G is the same or lower than in DDT_F.

$$u(G) = u(F) \text{ and } N_{u(F)}(G) \leq N_{u(F)}(F)$$

In the context of the meaning of the differential uniformity the positive case come when it is 'harder to attack' permutation G than permutation F. Higher the number of 'harder to attack' permutations G is, higher the probability in the definition is and lower the value of anomaly is.

In other words, the anomaly denote how 'hard' it is to attack a permutation F in the context of a set of permutations, where the 'hardest to attack' permutation would have the anomaly equal to ∞.

Analogously we can think of the linear anomaly.

If we take a look on anomaly of our S–box π, we can come to interesting discovery. Perrin compares in [Per19] the differential and linear anomaly of π

with those of some other TKlog instances (randomly chosen) and with those of discrete logarithms denoted as $\log_\alpha^{\mathrm{FLY}}$ and $\log_\alpha^{\mathrm{HN}}$ and defined in Definition 29

Definition 29 ($\log_\alpha^{\mathrm{FLY}}$ and $\log_\alpha^{\mathrm{HN}}$.)**.** *We define* $\log_\alpha^{\mathrm{FLY}}$ *and* $\log_\alpha^{\mathrm{HN}}$ *as mappings* $\mathbb{F}_{2^n} \longrightarrow \mathbb{Z}/_{2^n\mathbb{Z}}$*, defined as:*

$$\log_\alpha^{\mathrm{HN}}(\mathbf{x}) = \begin{cases} 2^n & \textit{for } x = 0, \\ 0 & \textit{for } x = 1, \\ \log_\alpha(x) & \textit{for } x \notin \{0,1\} \end{cases}, \log_\alpha^{\mathrm{FLY}}(\mathbf{x}) = \begin{cases} 0 & \textit{for } x = 0, \\ 2^n & \textit{for } x = 1, \\ \log_\alpha(x) & \textit{for } x \notin \{0,1\}. \end{cases}$$

Perrin found out that in comparison with random TKlog instances, π has its anomalies better than an average TKlog instance but not the best ones and even worse than both anomalies of $\log_\alpha^{\mathrm{FLY}}$ and $\log_\alpha^{\mathrm{HN}}$. He shows his findings in Figure [Per19, Figure 3] where one can get good intuition about the situation.

In the light of those experimental findings we can ask ourselves a question: if the designers chooses π to have TKlog properties, why they did not chose some other TKlog instance with better anomalies? This doubt can lead us to the effort to look for other reasons of the choice.

3. Permutations over finite field

The main aim of this thesis is to inspect whether the S-Box π is affine-equivalent to a fractional q-projective permutation. To achieve this we use a classification result from [Gö22] that proves that a fractional q-projective permutation is equivalent to two different forms. A direct way to inspect whether there exists such equivalence is computationally infeasible. We use mathematical tools to render such an inspection feasible. We use invariants of Boolean functions with respect to affine-equivalency to overcome the complexity of the experiment.

We dedicate the first section of this chapter to mention selected terms and results of [Gö22] and fit the arrangement to the situation of the S-box π. Next, we go through elementary representations of the S-box to show that they does not help us in finding any other valuable structure in Section 3.2. In section 3.3 we complete the preparation of the experiment described in last section, Section 3.4. The experiment is accompanied by all used code in Attachment A.7, A.11 and A.10 and its result can be seen in attached `invariants.txt` and `invariants special.txt` files.

3.1 Classification of fractional projective permutations

Paper [Gö22] focuses on a special kind of projective permutation and provides a classification of all such permutations. We already prepared a background of definitions in Section 1.4 in the introductory chapter and now we will present the special kind of permutations and explain the way we want to use them.

Let us start with the notation and the definitions that are taken directly from [Gö22] to introduce the situation. Firstly, we give the notation which we will follow from now on.

Notation 13. *Let $p \in \mathbb{P}$ be a prime, $k, l \in \mathbb{N}$, $k < l$, then*

- $r = p^l$,
- $q = p^k$,
- $\mathbb{L}$ *is the field with r elements,*
- $\mathbb{K}$ *is the field with q elements.*

Now we give the definition of the main kind of terms. These terms are defined for arbitrary choice of used parameters. For our purposes we will choose

$$p = 2,\ l = 8 \text{ and } k = 4,$$

as we describe in Example after Theorem 16, because that setting seems to fit the situation around the S-box, which we will describe later in the section.

Definition 30. *Let $p, q, \mathbb{L}$ be as in Notation 13. We define the* **q–*projective polynomial*** *as*

$$f(x) = a_{q+1}x^{q+1} + a_q x^q + a_1 x + a_0 \in \mathbb{L}[x].$$

Let f, g be q–projective polynomials. The ***fractional* q–*projective function*** *is defined as*

$$\Pi_{f,g}(x) = \frac{f(x)}{g(x)},\ \Pi_{f,g} : \mathbb{P}^1(\mathbb{L}) \to \mathbb{P}^1(\mathbb{L}).$$

The main result of [Gö22] is the classification of fractional q-projective functions by projective equivalency that we define in Definition 20. The theorem is mentioned here as Theorem 16 and it uses carefully chosen constants, which are listed here in Notation 14 with need of Definition 31.

Definition 31. *Let $p \in \mathbb{P}$ a prime, $l, d \in \mathbb{N}$, $d|l$. Let $\mathbb{L}$ and $\mathbb{D}$ be finite fields such that $\mathbb{L} \supseteq \mathbb{D}$, $\mathbb{L} = \mathbb{F}_{p^l}$, $\mathbb{D} = \mathbb{F}_{p^d}$. We define the* ***trace function*** *as*

$$\mathrm{tr}_{\mathbb{L}\setminus\mathbb{D}}(x) = \sum_{j=0}^{\frac{l}{d}-1} x^{p^{dj}}.$$

We keep the notation used in Definition 31 and complete the exposition with terms we will need in Theorem 16 and an analysis right after. We write down all of important terms in Notation 14.

Notation 14. *Let $q, r, k, l, \mathbb{K}, \mathbb{L}$ represent same structures as in Notation 13. Let $d = \gcd(k, l)$ then:*

- *$\mathbb{D}$ stands for $\mathbb{L} \cap \mathbb{K} = \mathbb{F}_{2^d}$,*
- *$\mathbb{L}' := \mathbb{F}_{r^{2^i}}$ where i is maximal such that $2^i | \frac{k}{d}$,*
- *ω is an element such that $\mathbb{L}'(\omega)$ is an extension of $\mathbb{L}'$ of degree 2,*
- *ϵ_1 an element from $\mathbb{L}$ forced to have $\mathrm{tr}_{\mathbb{D}\setminus\mathbb{F}_2}(\epsilon_1) = 1$,*
- *$\epsilon_2 = \omega^2 + \omega \in \mathbb{L}'$,*
- *$\epsilon_q = \omega^q + \omega \in \mathbb{L}$,*
- *$\delta \in \mathbb{K}$ with $\delta + \epsilon_2 \in \mathbb{L}$.*

We list terms in Notation 14 only to help us with representation of the S-box π as a fractional polynomial function. Knowledge of construction or properties of those terms is not much important for our purposes, but in Attachment A.7 we show how to find a consistent set of values for those terms.

It is showed in [Gö22][Proposition 3.1] that every fractional q-projective functions over a finite field is projectively equivalent to one of two separate forms. Those forms are fractional q-projective functions with carefully chosen coefficients. We write down the proposition as Theorem 16 with the notion of projective equivalency taken from Definition 20.

Theorem 16 ([Gö22], Proposition 3.1)**.** *Let $\Pi(x)$ be a fractional q-projective permutation of $\mathbb{P}^1(\mathbb{L})$. Then with Notation 14, it holds that char($\mathbb{L}$)=2 and $\Pi(x)$ is projectively equivalent to, either:*

1. *$\psi_1(x) = \frac{x^{q+1}+(\epsilon_q+1)x+\epsilon_2+\delta+\epsilon_1}{x^q+x+\epsilon_q}$, with $\mathrm{tr}_{\mathbb{D}/\mathbb{F}_2}(\epsilon_1) = 1$, or*

2. $\psi_2(x) = \frac{x^{q+1}+(\epsilon_q+1)x+\epsilon_2+\delta}{x^q+x+\epsilon_q}$.

Remark. We can observe that ψ_1 and ψ_2 can both be permutations of affine space $\mathbb{A}(\mathbb{L}) \simeq \mathbb{L} \simeq \mathbb{F}_2^l$ because degree of the numerator is strictly greater then degree of the denominator and so by Definition 18 we get

$$\Pi(\infty) = \infty$$

and therefore since Π is assumed to be a permutation we get also

$$\Pi(\mathbb{L}) = \Pi(\mathbb{A}(\mathbb{L})) = \mathbb{A}(\mathbb{L}) = \mathbb{L}.$$

Theorem 16 can be interpreted so that for every fractional q-projective permutation there exist $\mu_1, \mu_2 \in \mathrm{PGL}_2(\mathbb{L})$ such that $\Pi(x)$ can be written as

$$\Pi(x) = \mu_1 \circ \frac{a_{q+1}x^{q+1} + a_q x^q + a_1 x + a_0}{b_{q+1}x^{q+1} + b_q x^q + b_1 x + b_0} \circ \mu_2$$

where

$$\begin{array}{llll} a_{q+1} = 1, & a_q = 0, & a_1 = \epsilon_q + 1, & a_0 = \epsilon_2 + \delta + \epsilon_1 \text{ or } a_0 = \epsilon_2 + \delta, \\ b_{q+1} = 0, & b_q = 1, & b_1 = 1, & b_0 = \epsilon_q \end{array}$$

with $\epsilon_q, \epsilon_2, \epsilon_1, \delta$ satisfying requirements from Notation 14. This give us a tool how to inspect every single fractional q-projective function. We can choose arbitrary valid values for parameters e_q, e_2, e_1, δ, define two initial functions ψ_1, ψ_2 and then run through

$$\mu_2 \circ \psi_1 \circ \mu_1 \text{ and } \mu_2 \circ \psi_2 \circ \mu_1$$

for all possible $\mu_1, \mu_2 \in \mathfrak{M}(\mathbb{L})$ (defined in Definition 19). Theorem 16 ensures that like this we meet every fractional q-projective function.

3.1.1 Apparent connection between fractional q-projective functions and TKlog

Let us now talk about the initial motivation of searching for connection between Perrin's paper and fractional projective permutations. Perrin describe the structure of the S-box π as a TKlog structure which has a partition preserving property. We describe the property in Section 2.3.1. We recall that instances of TKlog map multiplicative cosets on the input to additive cosets of the finite field on the output. The cosets are taken with respect to the maximal subgroups of the subfield of half dimension.

There is two interesting function between a finite field and its subfield of index two related to the special property of TKlog. We already defined one of them, the trace function in Definition 31. We define now the norm function.

Definition 32. *Let $p \in \mathbb{P}$ a prime, $l, d \in \mathbb{N}$, $d|l$. Let $\mathbb{L}$ and $\mathbb{D}$ be finite fields such that $\mathbb{L} \supseteq \mathbb{D}$, $\mathbb{L} = \mathbb{F}_{p^l}$, $\mathbb{D} = \mathbb{F}_{p^d}$. We define the* ***norm function*** *as*

$$\mathrm{norm}_{\mathbb{L}\setminus\mathbb{D}}(x) = \prod_{j=0}^{\frac{l}{d}-1} x^{p^{dj}}.$$

If we take the case of a finite field and its subfield of index 2 i.e. $l = 2k, q = p^k$ we get

$$\mathrm{tr}_{\mathbb{L}\setminus\mathbb{K}}(x) = \sum_{j=0}^{\frac{l}{k}-1} x^{p^{kj}} = x^q + x$$

and

$$\mathrm{norm}_{\mathbb{L}\setminus\mathbb{K}}(x) = \prod_{j=0}^{\frac{l}{k}-1} x^{p^{kj}} = x^{q+1}.$$

The trace function classify all element of $\mathbb{L}$ into additive cosets by the value of its trace. It means that two elements belonging to the same additive coset have the same value of its trace. Analogously the norm function divides $\mathbb{L} \setminus \{0\}$ into multiplicative cosets by the value of their norm. Moreover the trace function and the norma functions are polynomial mappings with exponents that we can see in a q-projective polynomial and in both numerator and denominator of a fractional q-projective function.

Therefore it seems as a good idea to explore whether there is a connection between TKlog a structure that preserves partitions into cosets and fractional q-projective functions that are composed by mappings trace and norm functions that divide the finite field into cosets.

3.1.2 Setting of constants

As we already mentioned, we want to find a representations of the S-box π or an affine-equivalent function to π in the set of all fractional q-projective functions, if there exists one. In this sections we give concrete values to terms that we defined in general to suit our situation around the S-box.

The first choice should be

$$p = 2, l = 8,$$

since the S-box permutes $\mathbb{F}_{2^8}$. Another choice has to be made and that is

$$k = \frac{1}{2} \cdot l = 4$$

because of the partition preserving property of TKlog. Definition of every other terms are consequences of these to choices.

We will go through all terms from from Notation 13, Definition 31 and Notation 14 and we list their values, so that they suits the situation:

- $p = 2, l = 8, k = 4, d = \gcd(8, 4) = 4,$
- $r = 2^8, q = 2^4,$
- $\mathbb{L} = \mathbb{F}_{2^8}, \mathbb{K} = \mathbb{D} = \mathbb{F}_{2^4},$
- $\frac{k}{d} = 1 \Rightarrow i = 0$ and $\mathbb{L}' = \mathbb{L}.$

If we propagate those values into Definition 18 and we get that we deal with fractional projective functions

$$\Pi(x) = \frac{a_{17}x^{17} + a_{16}x^{16} + a_1x + a_0}{b_{17}x^{17} + b_{16}x^{16} + b_1x + b_0}$$

and in Theorem 16 we focus on functions

$$\psi_1(x) = \frac{x^{17} + (\epsilon_q + 1)x + \epsilon_2 + \delta + \epsilon_1}{x^{16} + x + \epsilon_q} \text{ and } \psi_2(x) = \frac{x^{17} + (\epsilon_q + 1)x + \epsilon_2 + \delta}{x^{16} + x + \epsilon_q}$$

with

- $\mathrm{tr}_{\mathbb{L}\setminus\mathbb{D}}(x) = \mathrm{tr}_{\mathbb{F}_{2^8}\setminus\mathbb{F}_{2^4}}(x) = \sum_{j=0}^{1} x^{2^{4j}} = x + x^{2^4} = x + x^{16}$
- $\mathrm{tr}_{\mathbb{D}\setminus\mathbb{F}_2}(x) = \mathrm{tr}_{\mathbb{F}_{2^4}\setminus\mathbb{F}_2} = \sum_{j=0}^{3} x^{2^j} = x + x^2 + x^4 + x^8$
- $\mathrm{tr}_{\mathbb{D}\setminus\mathbb{F}_2}(x) = \mathrm{tr}_{\mathbb{F}_{2^8}\setminus\mathbb{F}_2} = \sum_{j=0}^{7} x^{2^j} = x + x^2 + x^4 + x^8 + x^{16} + x^{32} + x^{64} + x^{128}$
- $\omega \in \mathbb{F}_{2^{16}} \setminus \mathbb{F}_{2^8}$ such that $\mathbb{F}_{2^8}(\omega) = \mathbb{F}_{2^{16}}$
- $\epsilon_q = \omega^q + \omega$, $\epsilon_q \in \mathbb{F}_{2^8}$
- $\epsilon_2 = \omega^2 + \omega$, $\epsilon_2 \in \mathbb{F}_{2^8}$
- $\epsilon_1 \in \{e \in \mathbb{F}_{2^8} : \mathrm{tr}_{\mathbb{D}\setminus\mathbb{F}_2}(e) = 1\}$
- $\delta \in \mathbb{F}_{2^4}$ and by [Gö22, Section 3.1 and Lemma 4.1] since

  $$\mathbb{K} = \mathbb{D} \Rightarrow [\mathbb{K} : \mathbb{D}] = 1$$

  is odd, we can choose $\delta = 0$.

With such setting we can choose any valid combination of constants $\epsilon_q, \epsilon_2, \epsilon_1, \delta$ and define our initial ψ_1 and ψ_2 that will be combined later with Möbius transformations of special type. We give example code in Attachment A.7 how we can get concrete values of those constants. Constants are computed in a different way then defined in [Gö22], because it is more suitable for computer. We compute $\epsilon_q, \epsilon_2, \omega$ as

- $\epsilon_2 \in \{e \in \mathbb{F}_{2^8} : \mathrm{tr}_{\mathbb{L}\setminus\mathbb{F}_2}(e) = 1\}$ arbitrary,
- $\omega \in \{e \in \mathbb{F}_{2^{16}} : e^2 + e = \epsilon_2\}$ arbitrary,
- $\epsilon_q = \omega^q + \omega$

and we define ψ_1 and ψ_2 as already described.

3.2 Elementary representations of S-box π

Before we dive into the main experiment that tries to connect results of the two paper, we make a step aside and take a look on simple representation of S-box π.

A function such as our S-box can be represented by a lookup table, which is actually the way how S-box π is defined in [DD13] and [Dol16]. Another representation that can come in mind among first options is algebraic normal form of the S-box if it is interpreted as a vectorial Boolean function or representation by a polynomial, known as Lagrange interpolation polynomial, if the S-box is considered as permutation of a finite field.

These are three main representations that exists for every function. This section should not serve as explanation of theory behind methods for getting the algebraic normal form or Lagrange interpolation polynomial, but as description of how these representation look like for our S-box.

In the last part of the section we give a representation of π that covers all cases form Perrin's definition of TKlog within one single formula.

3.2.1 S-box π as a vectorial Boolean function

S-box π is defined by a value table $\mathbb{Z}_{256} \to \mathbb{Z}_{256}$. In the value table we have record such as $i \mapsto j$ and if we want transfer the table to a value table over $\mathbb{F}_2^8$ we have to transfer both preimage and image from $\mathbb{Z}_{256}$ into Boolean vectors of dimension 8 by natural transfer into Boolean representation, done also by function V_8 from Definition 4, we get a lookup table $\mathbb{F}_2^8 \to \mathbb{F}_2^8$. Vectorial Boolean function with such value table can be expressed in algebraic normal form.

There exists a procedure how to find the algebraic normal form from a lookup table. Some of methods of converting representations of a Boolean function are described in [WF16], however we followed a modified procedure for vectorial Boolean function:

1. Write the look up table as Boolean matrix where columns are Boolean representation of images
$$T = \Big(\mathrm{Int}_8(\pi(0)) |\, \mathrm{Int}_8(\pi(1)) | \ldots |\, \mathrm{Int}_8(\pi(255)) \Big)$$
in our case we get a matrix of dimension 8×256,

2. Construct a matrix M_8 defined as
$$M_1 = \begin{pmatrix} 1 \end{pmatrix}, M_2 = \begin{pmatrix} 1 & 1 \\ 0 & 1 \end{pmatrix}, \ldots, M_k = \begin{pmatrix} M_{k-1} & M_{k-1} \\ 0 & M_{k-1} \end{pmatrix},$$
in our case we get a matrix of dimension 256×256,

3. From product of matrices $T \cdot M_8$ take columns as vector coefficients $a_{\overline{u}}$
$$T \cdot M_8 = \Big(a_{\mathrm{Int}_8(0)} | a_{\mathrm{Int}_8(1)} | \ldots | a_{\mathrm{Int}_8(255)} \Big),$$

4. Write the algebraic normal form for π as
$$\pi_{ANF} \sum_{\overline{u} \in \mathbb{F}_2^8} a_{\overline{u}} x^{\overline{u}}.$$

As for other computation we chose SageMath, a computer algebra system based on programming language Python. We got the result from the code and its output that can be seen in Attachment A.5. Output `[2]` represents 8 sequences of coefficients for 8 Boolean function $f_i, i = 0, \ldots, 7$ for vectorial Boolean function
$$\pi_{\mathrm{VBF}} = (f_0, \ldots, f_7).$$
Each sequence is sorted as coefficients of $x^{\overline{u}}$ for
$$\overline{u} = \mathrm{V}_8(i), i = 0, \ldots, 255.$$
This representation does not give any interesting information that can help us in our purpose but it is one of elementary representations.

3.2.2 S-box π as a Lagrange polynomial

Lagrange interpolation is a method of finding a polynomial from finite field's polynomial ring that satisfies given value table. There exists such polynomial as proved in [SB11][Theorem 6.2] and the polynomial is named Lagrange interpolation polynomial. We give an explicit definition in Definition 33.

Definition 33. *Let $a_1, \dots, a_n \in \mathbb{F}_q$ with $a_i \neq a_j$ for $i \neq j$ and $b_1, \dots, b_n \in \mathbb{F}_q$ elements of finite field $\mathbb{F}_q$. Then we define a* ***Lagrange interpolation polynomial*** *as*

$$L(x) = \sum_{j=1}^{n} b_j l_j(x),$$

where l_j are elements of ***Lagrange basis*** *$\{l_1, \dots, l_n\}$, where*

$$l_j(x) = \prod_{i=1, i\neq j}^{n} \frac{x - a_i}{a_j - a_i}.$$

To get a value table for π over $\mathbb{F}_{2^8}$ we have to first convert image and preimage into Boolean vectors by mapping V_8 from Notation 4 and then transfer the Boolean vectors to element of the finite field $\mathbb{F}_{2^8}$ by mapping F_V from Notation 4 as well. Then if we compute the polynomial by Lagrange interpolation of this value table for our S-box π, we get a polynomial that can provide us some information.

We used computer algebra system SageMath based on programming language Python to compute the Lagrange polynomial and we got a polynomial of degree 254 with 4 factors each of them in first power. Code can be seen in Attachment A.6. Explicitly we get π_{poly} and f_1, f_2, f_3, f_4 such that

$$\pi_{\text{poly}} = \pi_1 \cdot \pi_2 \cdot \pi_3 \cdot \pi_4,$$

with

$$\deg(\pi_{\text{poly}}) = 254, \deg(\pi_1) = 1, \deg(\pi_2) = 3, \deg(\pi_3) = 19, \deg(\pi_4) = 231$$

This result does not provide us any useful information that can get us closer to a representation of S-box π as a fractional q-projective function.

An interesting result would be the case where π_{poly} has a factor π_i of degree 17. If additionally the inverse of product of all other factors

$$\pi_1 \cdot \pi_{\text{poly}}^{-1} \mod x^{q^2} - x$$

is of degree 17 or lower, we can get exactly the representation of a fractional q-projective function.

Unfortunately, as we can see in the output of the code in Attachment A.6, it is not the case because none of the factors is of degree 17 and if we take the factor with closest degree to 17, the factor f_3 with degree 19 as numerator, combinations of the remaining factors does not have degree much lower then 255 and

$$\pi_{\text{poly}} = \pi_1 \cdot \pi_2 \cdot \pi_3 \cdot \pi_4 = \frac{\pi_3}{\pi_1^{-1} \cdot \pi_2^{-1} \cdot \pi_4^{-1}}.$$

is not a q-projective polynomial nor a fractional polynomial with similar degree of numerator and denominator.

3.2.3 One single formula for S-box π

TKlog as a structure found in our S-box is defined by κ and a permutation s. The definition gives formula for $\{0\}, \langle\alpha^{17}\rangle$ and $(\mathbb{F}_{2^8} \setminus \langle\alpha^{17}\rangle) \setminus \{0\}$ separately. We will follow now Perrin's notation that omit auxiliary transformations between algebraic structures, while explicit algorithm for π is described in Section 2.2.1. We can get rid of the separation by reorganisation

$$\pi(x) = \begin{cases} \kappa(x) & x = 0 \\ \kappa(16-j) & x = \alpha^k \text{ for } k = i + 17j \text{ and } i = 0 \\ \kappa(16-i) + \alpha^{17 \cdot s(j)} & x = \alpha^k \text{ for } k = i + 17j \text{ and } i \neq 0 \end{cases}$$

into one single formula

$$\pi(x) = g_0 \cdot \kappa(0) + g_1 \cdot \kappa(16-j) + g_2 \cdot \Big(\kappa(16-i) + \alpha^{17 \cdot s(j)}\Big),$$

where $x = 0$ or $x = \alpha^k$ with $k = i + 17j$ and g_1, g_2 defined as

$$g_0 = 1 - x^{255}, g_1 = 1 - (1-\alpha^i)^{255}, g_2 = 1 - \alpha^{15k}.$$

We take actually 3 formulas from Perrin's definition and put them together in one and only formula while only one of original formulas is active for x belonging to one of 3 original subsets since g_0, g_1 and g_2 are defined so that they fullfill the following.

$$g_0(x) = \begin{cases} 1 & x = 0 \\ 0 & x = \alpha^k \text{ for } k = i + 17j \text{ and } i = 0 \\ 0 & x = \alpha^k \text{ for } k = i + 17j \text{ and } i \neq 0 \end{cases}$$

$$g_1(x) = \begin{cases} 0 & x = 0 \\ 1 & x = \alpha^k \text{ for } k = i + 17j \text{ and } i = 0 \\ 0 & x = \alpha^k \text{ for } k = i + 17j \text{ and } i \neq 0 \end{cases}$$

$$g_2(x) = \begin{cases} 0 & x = 0 \\ 0 & x = \alpha^k \text{ for } k = i + 17j \text{ and } i = 0 \\ 1 & x = \alpha^k \text{ for } k = i + 17j \text{ and } i \neq 0 \end{cases}$$

Moreover we can use the structure of mapping κ to simplify the formula. Mapping κ is an affine mapping consisting of a linear part represented by multiplication by a matrix A and shift by a vector b. Since $\pi(0) = \kappa(0) = b$ and $\kappa(x) = A(x) + b$ for $x \neq 0$ we can simplify the formula as

$$\pi(x) = b + g_1(x) \cdot A(q-j) + g_2(x) \cdot \Big(A(q-i) + \alpha^{17 \cdot s(j)}\Big).$$

If we add the auxiliary transformations in the formula we get a function corresponding to the value table over $\mathbb{F}_{2^8}$

$$\pi(x) = \mathrm{F_V}(b) + g_1(x) \cdot \mathrm{F_V}(A(\mathrm{V}_4(q-j))) + g_2(x) \cdot \Big(\mathrm{F_V}(A(\mathrm{V}_4(q-i))) + \alpha^{17 \cdot s(j)}\Big).$$

3.3 Möbius transform and invariants

Paper [Gö22] provides us a classification of all fractional q-projective function by projective equivalency. We get that every fractional q-projective function is projectively equivalent to one of the unique forms with any valid choice of free parameters. This means that if our S-box π has the form of fractional q-projective function we should be able to find that form by inspection of all projective equivalents of ψ_1 and ψ_2 for fixed $\epsilon_q, \epsilon_2, \epsilon_1$ and δ.

Inspection of all instances of fractional q-projective functions is not doable in feasible time and therefore we have to choose another approach of how to compare two functions. We lower our goal to find exactly the representation of π to find only representation of a function that can be obtained from π by affine transformations. This pays off by need of inspection of affine invariants of lower number of functions, always one representative from each affine equivalency class.

Finding that form by bruteforce could be theoretically successful but in real execution we would have to go through all $2 \times ((q^2+1)q^2(q^2-1))^2 \approx 2^{49}$ choices for μ_1, μ_2 that provide the projective equivalency. Additionally, a run through all of possible choices of μ_1, μ_2 contains also inspection of some affine equivalent functions to a fractional projective function.

We can include also those function in our experiment, because if the S-box π is not exactly a fractional q-projective function, but a function affine equivalent to them, it can provide similar information as it would be fractional q-projective function itself.

If we want to inspect all projective equivalent function to ψ_1, ψ_2 and all affine equivalent function to them we should start with an experiment where we go through all mappings of form

$$L_2 \circ \mu_2 \circ \psi_1 \circ \mu_1 \circ L_1,$$

or

$$L_2 \circ \mu_2 \circ \psi_2 \circ \mu_1 \circ L_1,$$

with L_1, L_2 affine permutations and μ_1, μ_2 Möbius transformation. Together possibly up to $\approx 2^{49} \times 2^{32} \approx 2^{81}$, since L_1, L_2 both can be of form $ax+b$.

We developed a way how to reduce number of choices we will have to go through. It combines multiple aspects of permutations over affine and projective space $\mathbb{A}^1(L)$ and $\mathbb{P}^1(\mathbb{L})$ with properties of affine equivalent permutations.

We will dedicate separate Section 3.3.1, 3.3.2 and 3.3.3 to every step in our reduction. We unify the notation for used Möbius transforms μ_1, μ_2 in a way that we want to find such μ_1, μ_2 that we get:

$$\mu_2 \circ \Pi \circ \mu_1(x) \sim_{EA} \pi(x)$$

for $\Pi = \psi_1$ or $\Pi = \psi_2$ from Theorem 16 and

$$\mu_1 = \frac{a_1x+b_1}{c_1x+d_1}, \mu_2 = \frac{a_2x+b_2}{c_2x+d_2}.$$

3.3.1 Affine equivalency

With general bruteforce the simplest way how to compare two mappings is to test the trivial invarinat - whether their value tables matches i.e. whether

$$\mu_2 \circ \Pi \circ \mu_1(x) = \pi(x) \ \forall x \in \mathbb{F}_{2^8}.$$

Another way how to compare them is to inspect invariants other then the functional value e.g. invariants from Section 1.2. For general bruteforce it is not really efficient because computation of invariants takes more time then computation of functional value and moreover we get only a set of candidates for affine equivalent mappings to π.

On the other hand we can use this vagueness as an advantage and take it as aim to find a set of candidates for affine equivalent functions to π i.e. candidates for π up to affine equivalency. This will allow us to omit L_1, L_2 from the formula and all possibilities that are affine equivalent to another. Moreover we can then consider also all $\mu_2 \circ \Pi \circ \mu_1$ up to affine equivalency and this leads us also to reduce the set of all possibilities for μ_1, μ_2.

In the set of Möbius transfers there exists transformations that are at the same time affine mappings. It is those μ_1, μ_2 such that $c_1, c_2 = 0$, because then we have

$$\mu_1(x) = \frac{a_1x + b_1}{d_1} = \frac{a_1}{d_1}x + \frac{b_1}{d_1},\ \mu_2(x) = \frac{a_2x + b_2}{d_2} = \frac{a_2}{d_2}x + \frac{b_2}{d_2},$$

which are affine mappings. We will therefore forbid choice $c_1 = 0$ and $c_2 = 0$.

Let us now specify how we can enforce μ_1, μ_2 to be a non-affine transformation. We know that if we choose $c_1, c_2 = 0$ we get affine transformations. Let us then choose $c_1, c_2 \neq 0$. Especially without lost of generality we can choose $c_1, c_2 = 1$ because every choice of μ_1, μ_2 with $c_1, c_2 \neq 0$ can be reorganised so that

$$\mu_1(x) = \frac{a_1'x + b_1'}{c_1'x + d_1'} = \frac{\frac{a_1'}{c_1'}x + \frac{b_1'}{c_1'}}{x + \frac{d_1'}{c_1'}} = \frac{a_1x + b_1}{x + d_1},$$

$$\mu_2(x) = \frac{a_2'x + b_2'}{c_2'x + d_2'} = \frac{\frac{a_2'}{c_2'}x + \frac{b_2'}{c_2'}}{x + \frac{d_2'}{c_2'}} = \frac{a_2x + b_2}{x + d_2}.$$

With the condition on $a_1'd_1' - b_1'c_1' \neq 0$, $a_2'd_2' - b_2'c_2' \neq 0$ or now equivalently $a_2d_2 - b_2 \neq 0$, $a_2d_2 - b_2 \neq 0$ we get all possibilities to go through.

The idea of reduction of the set of all possibilities for μ_1, μ_2 is built up on the fact that value of invariants for $\Pi \in \{\psi_1, \psi_2\}$ and mappings affine equivalent to Π will be that same. Therefore comparison invariants for Π and π will give us the same information as comparison invariants for a mapping affine equivalent to Π with invariants for π.

Example. Let us take property ρ as invariant with respect to affine equivalency. Let $\Pi \sim_{EA} \Pi'$ and π our S-box.

Since $\Pi \sim_{EA} \Pi'$ we know that $\rho(\Pi) = \rho(\Pi')$ and therefore we get

$$\rho(\Pi) = \rho(\pi) \iff \rho(\Pi') = \rho(\pi)$$

and while we meet Π or Π' in the inspection of all possibilities and compare its ρ to $\rho(\pi)$ we already know how the comparison of the invariant will turn out for the other variant.

Moreover we know that $\Pi \sim_{EA} \Pi'$ and we can find it later by inspecting all affine equivalent mapping to get Π or Π' from each other.

In following section we describe how we can reduce the number of functions to be inspected if we focus only on one representative from every class of affine equivalency.

3.3.2 Composition with affine transformations

Result of the experiment should be a set of candidates for affine equivalent functions to the S-box π. In the result set we always need at least one representative from a class of affine equivalency with same invariants. However we need at most one representative from each class of affine equivalency, because we know that the other will have same value of invariants as the one representative. This provides us opportunity to exclude those μ_1, μ_2 that we can obtain as a composition of another Möbius transform with an affine transform.

With Notation 7, according to Observation 6 every μ_1 and μ_2 can be obtained as composition

$$\begin{aligned}\mu_1(x) =& \frac{ax+b}{cx+d}\\ =& f_{T,\frac{a}{c}} \circ f_{D,\frac{bc-ad}{c^2}} \circ f_I \circ f_{T,\frac{d}{c}}(x)\\ =& \Big(x+\frac{a}{c}\Big) \circ \Big(\frac{ad-bc}{c^2}x\Big) \circ \frac{1}{x} \circ \Big(x+\frac{d}{c}\Big)\end{aligned}$$

and if we choose $c = 1$ as we already commented in Section 3.3.1 we get

$$\frac{ax+b}{x+d} = \mu_1(x) = \Big(x+a\Big) \circ \Big((ad-b)x\Big) \circ \frac{1}{x} \circ \Big(x+d\Big).$$

The part of the composition that is on the right side from transformation $\frac{1}{x}$ can be considered as a part of transformation L_1 and not a part of μ_1.

Another reduction can be considered. We focus on part of the composition $((ad-b)x) \circ (\frac{1}{x})$ and we reformulate it to form

$$\Big((ad-b)x\Big) \circ \frac{1}{x} = \frac{ad-b}{x} = \frac{1}{x} \circ \Big(\frac{1}{ad-b}x\Big),$$

where we have guarantied that $ad - b \neq 0$ from the original condition for Möbius transformations. Again we can move the part on the right side of $\frac{1}{x}$ to the affine mapping L_1 and all together we get that we have to run through all possible μ_1 of form

$$\mu_1(x) = \Big(x+a\Big) \circ \frac{1}{x}$$

and the number of such μ_1 is equal to 2^8.

Analogously we get that μ_2 of general form

$$\begin{aligned}\mu_2(x) =& \frac{ax+b}{cx+d}\\ =& f_{T,\frac{a}{c}} \circ f_{D,\frac{bc-ad}{c^2}} \circ f_I \circ f_{T,\frac{d}{c}}(x)\\ =& \Big(x+\frac{a}{c}\Big) \circ \Big(\frac{ad-bc}{c^2}x\Big) \circ \frac{1}{x} \circ \Big(x+\frac{d}{c}\Big)\end{aligned}$$

can be reorganised so that with $c = 1$ we will have to go through all 2^8 possibilities of form

$$\mu_2(x) = \frac{1}{x} \circ \Big(x+d\Big),$$

since everything on the left of $\frac{1}{x}$ can be considered as a part of L_2.

All together we will go through all choices of $a, d \in \mathbb{F}_{2^8}$ in expressions

$$\begin{aligned}
L_2 \circ \frac{1}{x} \circ \Big(x+d\Big) \circ \psi_1 \circ \Big(x+a\Big) \circ \frac{1}{x} \circ L_1 \\
L_2 \circ \psi_1 \circ \Big(x+a\Big) \circ \frac{1}{x} \circ L_1 \\
L_2 \circ \frac{1}{x} \circ \Big(x+d\Big) \circ \psi_1 \circ L_1 \\
L_2 \circ \psi_1 \circ L_1
\end{aligned}$$

and

$$\begin{aligned}
L_2 \circ \frac{1}{x} \circ \Big(x+d\Big) \circ \psi_2 \circ \Big(x+a\Big) \circ \frac{1}{x} \circ L_1 \\
L_2 \circ \psi_2 \circ \Big(x+a\Big) \circ \frac{1}{x} \circ L_1 \\
L_2 \circ \frac{1}{x} \circ \Big(x+d\Big) \circ \psi_2 \circ L_1 \\
L_2 \circ \psi_2 \circ L_1
\end{aligned}$$

together $2 \times (2^{16} + 2^8 + 2^8 + 1) = 2^{18} + 2$ possibilities which is already a number of functions to be inspected, that can be considered as low enough.

3.3.3 Point at infinity

We are interested in a connection between S-box π and fractional projective functions. S-box π is a permutation of affine space $\mathbb{F}_{2^8} = \mathbb{A}^1(\mathbb{F}_{2^8})$ while fractional q-projective functions are by default defined to be permutations of projective space $\mathbb{P}^1(\mathbb{F}_{2^8}) = \mathbb{F}_{2^8} \cup \{\infty\}$. This is something that has to be clarified before we start the experiment.

As we defined evaluation of fractional polynomials in Definition 18 and we get

$$\psi_1\Big(\mathbb{A}^1(\mathbb{F}_{2^8})\Big) = \mathbb{A}^1(\mathbb{F}_{2^8}), \psi_1(\infty) = \infty$$

as well as

$$\psi_2\Big(\mathbb{A}^1(\mathbb{F}_{2^8})\Big) = \mathbb{A}^1(\mathbb{F}_{2^8}), \psi_2(\infty) = \infty$$

for every ψ_1 and ψ_2.

Composition with non linear Möbius transforms μ_1, μ_2 does not preserve this property, on the other hand composition with affine transformation does and we have to be careful in comparisons of Π transformed ψ_1 and ψ_2. Specially we have to take care of those that does not map ∞ to ∞.

3.3.3.1 Restriction on affine space

One of approaches is to focus only on those that does map ∞ to ∞ by forcing $d = \psi_i(a)$ for $i \in \{1, 2\}$ because then we get

$$\begin{aligned}
\Pi(\infty) &= \left(\frac{1}{x} \circ \Big(x + \psi_i(a)\Big) \circ \psi_i \circ \Big(x + a\Big) \circ \frac{1}{x}\right)(\infty) \\
&= \left(\frac{1}{x} \circ \Big(x + \psi_i(a)\Big) \circ \psi_i \circ \Big(x + a\Big)\right)(0) \\
&= \left(\frac{1}{x} \circ \Big(x + \psi_i(a)\Big) \circ \psi_i\right)(a) \\
&= \left(\frac{1}{x} \circ \Big(x + \psi_i(a)\Big)\right)(\psi_i(a)) \\
&= \left(\frac{1}{x}\right)(0) \\
&= \infty
\end{aligned}$$

for both $i = 1, 2$. This approach would consider only variants listed at the end of Section 3.3.2 that has μ_1, μ_2 on both sides of formula and variants without μ_1, μ_2 on both sides:

$$L_2 \circ \frac{1}{x} \circ \Big(x + d\Big) \circ \psi_1 \circ \Big(x + a\Big) \circ \frac{1}{x} \circ L_1$$
$$L_2 \circ \psi_1 \circ L_1$$

and

$$L_2 \circ \frac{1}{x} \circ \Big(x + d\Big) \circ \psi_2 \circ \Big(x + a\Big) \circ \frac{1}{x} \circ L_1$$
$$L_2 \circ \psi_2 \circ L_1$$

This approach may be too restrictive with only $2^9 + 2$ possibilities. We run the experiment with this approach with code in Attachment A.10 in SageMath, computer algebra system based on programming language Python. The code did not give us any fractional q-projective function with same differential spectrum as π has and therefore we develop also another approach how to deal with the case that ∞ is mapped to an element of $\mathbb{L}$.

3.3.3.2 Fractional jump

We will explore another approach, that is a bit outside of the idea of finding a representation of S-box π as a fractional q-projective function. The idea is to force the image of the function by additional manipulation to cover all elements from $\mathbb{L}$. Therefore if we meet the case where ∞ is mapped to an element from $\mathbb{L}$. we can force the function to map the preimage of ∞ to the image of ∞ and fulfill the nature of permutation of the affine space by that.

Suppose that we have Π such that ∞ is not mapped to ∞. We can apply simple manipulation for such function $\Pi : \mathbb{P}^1(\mathbb{L}) \longrightarrow \mathbb{P}^1(\mathbb{L})$ to obtain a function $\Pi_{\mathbb{A}} : \mathbb{L} \longrightarrow \mathbb{L}$

$$\Pi(x) = \frac{f_1}{f_2}(x) \rightsquigarrow \Pi_{\mathbb{A}}(x) = f_1 \cdot f_2^{2^r-2}(x) \mod x^r - x, \tag{3.1}$$

that gives us

$$\Pi(x) = \Pi_{\mathbb{A}}(x),\ \forall x \in \mathbb{L} \text{ and } \Pi(x) = \infty \Rightarrow \Pi_{\mathbb{A}}(x) = 0$$

and since Π is a permutation we get

$$\Pi_{\mathbb{A}}(\mathbb{L}) \subsetneq \mathbb{L}, \text{ because } \Pi_{\mathbb{A}}(\mathbb{L}) = \mathbb{L} \setminus \{\Pi(\infty)\},$$

since Π_A has two preimages of 0 and element $\Pi(\infty) \in \mathbb{L}$ does not have a preimage in $\mathbb{L}$. Therefore we have to straighten this situation. Overall we want to define a similar function that will be a permutation of the affine space.

Forcing f' to map the preimage of ∞ to the image of ∞ is the most direct way how to ensure that the function is permuting the affine space with minimal impact on the structure of images up to one preimage.

$$\left.\begin{matrix} f : \infty \mapsto \gamma \\ f : \beta \mapsto \infty \end{matrix}\right\} \longrightarrow f' : \beta \mapsto \gamma \tag{3.2}$$

Remark. We can get a function that permutes the affine space by forcing it to map the preimage of ∞ to an element of affine space while the preimage of that element is forced to be mapped to the image of ∞ and analogously with arbitrary many tuples image/preimage in the middle steps

$$\left.\begin{matrix} f : \infty \mapsto \gamma \\ f : \chi \mapsto \zeta \\ f : \beta \mapsto \infty \end{matrix}\right\} \longrightarrow \begin{cases} f' : \beta \mapsto \zeta \\ f' : \chi \mapsto \gamma \end{cases}, \tag{3.3}$$

maybe especially in our case with 17 middle steps that will correspond to the TKlog structure, but the impact on the structure grows with every middle step and therefore we choose to use the most direct modification 3.2.

Manipulation with aim to change image of one and only preimage is covered by a method called **fractional jump** construction. This method is described in [GM18, Chapter 2] for arbitrary dimension of projective space. We need the fractional jump for dimension 1 and the concrete form that provides us the manipulation is expressed in Observation 17.

Observation 17. *Let $f : \mathbb{F}_r \to \mathbb{F}_r$ for $\mathbb{F}_r$ a finite field. For every $\beta, \gamma \in \mathbb{F}_r$ let $f' = f + \big(\gamma - f(\beta)\big)\big(1 - (x-\beta)^{r-1}\big)$. It holds that*

$$f'(x) = \begin{cases} \gamma & x = \beta \\ f(x) & x \neq \beta. \end{cases} \tag{3.4}$$

Proof. If we develop $f'(x)$ we get

$$f'(x) = f(x) + \big(\gamma - f(\beta)\big)\big(1 - (x-\beta)^{r-1}\big)$$

and if $x = \beta$

$$f'(\beta) = f(\beta) + \Big(\gamma - f(\beta)\Big)\Big(1 - (\beta - \beta)^{q^2-1}\Big) = f(\beta) + \gamma - f(\beta) = \gamma$$

and if $x \neq \beta$

$$f'(x) = f(x) + \Big(\gamma - f(\beta)\Big)\Big(1 - (x - \beta)^{q^2-1}\Big) = f(x) + \Big(\gamma - f(\beta)\Big)(1-1) = f(x)$$

□

Construction of fractional jump together with modification 3.1 will preserve images of all affine points and preimage of ∞ will be mapped to image of ∞. Final manipulation will look like

$$\Pi(x) = \frac{f_1}{f_2}(x) \rightsquigarrow \Pi'(x) = f_1(x) \cdot f_2^{2^r-2}(x) + \gamma \cdot \Big(1 - (x - \beta)^{r-1}\Big).$$

Example. Let us analyse β and γ for each type of formula that we will need on its own

1. $\frac{1}{x} \circ \Big(x + d\Big) \circ \psi_1 \circ \Big(x + a\Big) \circ \frac{1}{x}$:
   maps $\infty \mapsto \frac{1}{\psi_i(a)+d}$ and $\frac{1}{\psi_i^{-1}(d)+a} \mapsto \infty$ and therefore
   $$\gamma = \frac{1}{\psi_i(a) + d} \text{ and } \beta = \frac{1}{\psi_i^{-1}(d) + a}$$
2. $\psi_1 \circ \Big(x + a\Big) \circ \frac{1}{x}$:
   maps $\infty \mapsto \psi_i(a)$ and $0 \mapsto \infty$ and therefore
   $$\gamma = \psi_i(a) \text{ and } \beta = 0$$
3. $\frac{1}{x} \circ \Big(x + d\Big) \circ \psi_1$:
   maps $\infty \mapsto 0$ and $\psi_i^{-1}(d) \mapsto \infty$ and therefore
   $$\gamma = 0 \text{ and } \beta = \psi_i^{-1}(d)$$
4. ψ_1 maps $\infty \mapsto \infty$ already

Those are point that we have to connect by force, by adding a formula from Observation 17.

Before we build in the method of fractional jump in our experiment, we have to first check how it commutes by composition with affine transformations from the right and the left side. Let us formulate this assumption as a theorem.

Theorem 18. *Let* $\mathrm{FJ}_{\beta,\gamma}$ *be a transformation of a function*

$$\mathrm{FJ}_{\beta,\gamma}(f) = f + \Big(\gamma - f(\beta)\Big)\Big(1 - (x - \beta)^{r-1}\Big)$$

forcing $\Big(\mathrm{FJ}(f)\Big)(\beta) = \gamma$ *for* $f : \mathbb{F}_r \to \mathbb{F}_r$ *such that* $f(\beta) = 0$.

Let $l_1, l_2 : \mathbb{F}_r \longrightarrow \mathbb{F}_r$ *be affine permutations* $l_1 : x \mapsto ax + b, l_2 : x \mapsto cx + d$ *for* $a, b, c, d \in \mathbb{F}_r$. *Then*

$$\Big(l_2 \circ \mathrm{FJ}_{\beta,\gamma}(f) \circ l_1\Big)(x) = \Big(\mathrm{FJ}_{l_1^{-1}(\beta), l_2(\gamma)}(l_2 \circ f \circ l_1)\Big)(x)$$

for every $x \in \mathbb{F}_r$.

Proof. We have to actually prove only

$$\Big(l_2 \circ \mathrm{FJ}_{\beta,\gamma}(f) \circ l_1\Big)(l_1^{-1}(\beta)) = l_2(\gamma) \iff \mathrm{FJ}_{l_1^{-1}(\beta),l_2(\gamma)}(l_2 \circ f \circ l_1)(l_1^{-1}(\beta)) = l_2(\gamma)$$

because for every other point $\delta \neq l_1^{-1}(\beta)$ it hold that

$$\Big(l_2 \circ \mathrm{FJ}_{\beta,\gamma}(f) \circ l_1\Big)(\delta) = l_2(\mathrm{FJ}_{\beta,\gamma}(f)(l_1(\delta))) = l_2(f(l_1(\delta)))$$

since the argument of $\mathrm{FJ}_{\beta,\gamma}(f)$ is $l_1(\delta) \neq \beta$ as well as

$$\mathrm{FJ}_{l_1^{-1}(\beta),l_2(\gamma)}(l_2 \circ f \circ l_1)(\delta) = l_2(f(l_1(\delta)))$$

since the argument of $\mathrm{FJ}_{\beta,\gamma}(l_2 \circ f \circ l_1))$ is $\delta \neq l_1^{-1}(\beta)$.

For the point $l_1^{-1}(\beta)$ we can prove 4 independent statements:

1. $\mathrm{FJ}_{\beta,\gamma}(f) \circ (ax) = \mathrm{FJ}_{\beta\cdot a^{-1},\gamma}(f \circ (ax))$
2. $\mathrm{FJ}_{\beta,\gamma}(f) \circ (x+b) = \mathrm{FJ}_{\beta+a,\gamma}(f \circ (x+b))$
3. $(cx) \circ \mathrm{FJ}_{\beta,\gamma}(f) = \mathrm{FJ}_{\beta,c\cdot\gamma}((cx) \circ f)$
4. $(x+d) \circ \mathrm{FJ}_{\beta,\gamma}(f) = \mathrm{FJ}_{\beta,\gamma+d}((x+d) \circ f)$

Each point follows from definition of FJ as we can see in the following.

1. By definition of $\mathrm{FJ}_{\beta,\gamma}$

$$\begin{aligned}\Big(\mathrm{FJ}_{\beta,\gamma}(f) \circ (ax)\Big)(\beta \cdot a^{-1}) &= \Big(\mathrm{FJ}_{\beta,\gamma}(f)\Big)(\beta \cdot a \cdot a^{-1}) \\ &= \Big(\mathrm{FJ}_{\beta,\gamma}(f)\Big)(\beta) \\ &= \gamma\end{aligned}$$

   By definition of $\mathrm{FJ}_{\beta\cdot a^{-1},\gamma}$

$$\Big(\mathrm{FJ}_{\beta\cdot a^{-1},\gamma}(f \circ (ax))\Big)(\beta \cdot a^{-1}) = \gamma$$

2. By definition of $\mathrm{FJ}_{\beta,\gamma}$

$$\begin{aligned}\Big(\mathrm{FJ}_{\beta,\gamma}(f) \circ (x+b)\Big)(\beta + b) &= \Big(\mathrm{FJ}_{\beta,\gamma}(f)\Big)(\beta + b + b) \\ &= \Big(\mathrm{FJ}_{\beta,\gamma}(f)\Big)(\beta) \\ &= \gamma\end{aligned}$$

   By definition of $\mathrm{FJ}_{\beta+b,\gamma}$

$$\Big(\mathrm{FJ}_{\beta+b,\gamma}(f \circ (x+b))\Big)(\beta + b) = \gamma$$

3. By definition of $\mathrm{FJ}_{\beta,\gamma}$

$$\begin{aligned}\Big((cx) \circ \mathrm{FJ}_{\beta,\gamma}(f)\Big)(\beta) &= (cx) \circ \Big(\mathrm{FJ}_{\beta,\gamma}(f)\Big)(\beta) \\ &= c \cdot \gamma\end{aligned}$$

   By definition of $\mathrm{FJ}_{\beta,c\cdot\gamma}$

$$\Big(\mathrm{FJ}_{\beta\gamma}((cx) \circ f)\Big)(\beta) = c \cdot \gamma$$

4. By definition of $\mathrm{FJ}_{\beta,\gamma}$

$$\begin{aligned}\Big((x+d)\circ \mathrm{FJ}_{\beta,\gamma}(f)\Big)(\beta) &= (x+d)\circ \Big(\mathrm{FJ}_{\beta,\gamma}(f)\Big)(\beta)\\ &= \gamma + d\end{aligned}$$

   By definition of $\mathrm{FJ}_{\beta,\gamma+d}$

$$\Big(\mathrm{FJ}_{\beta,\gamma+d}((x+d)\cdot f)\Big)(\beta+b) = \gamma + d$$

We just proved that composition with dilation and translation from both sides fulfill the statement. Since affine transformations are composed by a dilation and a translation, we proved the statement also for composition with affine transformations.

□

Remark. We can interpret Theorem 18 as an assurance that two following formulas give the same function.

1. We apply first the fractional jump on a function f and then apply affine transformations
$$l_2 \circ \mathrm{FJ}_{\beta,\gamma}(f) \circ l_1$$
   or

2. we first apply the affine transformations and then the fractional jump with modified parameters
$$\mathrm{FJ}_{l_1^{-1}(\beta),l_2(\gamma)}(l_2 \circ f \circ l_1).$$

Modified parameters in fractional jump for second case correspond to composition with the affine mappings because we have to take care always of the problematic pair β, γ and therefore we chose $l_1(\beta), l_2(\gamma)$ to avoid the case when the final image is ∞ and ensure that something is mapped to $l_2(\gamma)$. It is important because we want to preserve the property of permuting the affine space and we omit affine transformations in the experiment.

After we rearrange the function to be permutation of the affine space, we get a function that is not a fractional q-projective function but is very similar, because it has the value table the same up to one value. Since it seems like the S-box of interest is not a fractional q-projective permutation, we want to explore similar exposition as well, but we have to consider this in processing of data we will get in the course of the experiment. We describe actual procedure of the experiment with the methodology of evaluation of generated data.

3.4 Main experiment

In previous sections and especially in Section 3.3 we prepared the stage for a run of an experiment. The experiment has aim to locate the S-box π in one of affine-equivalency classes of set of fractional q-projective functions defined in Definition 30.

In following subsections we give first the description step-by-step of procedure of the experiment and then also a methodology of evaluation of generated data. We use the computer algebra system SageMath based on programming language Python. All used code is given in attachment and commented in this section. The code may require to import libraries, define constants or auxiliary function also described in Notation 4 or Definition 25. All of them are listed in Attachment A.2 as a base repository to be used anytime later in the code. Moreover if we define a variable once, we may refer to it later in a code.

Additionally we admit that one can always write a code that will be more effective or less complicated with better notation, but our aim is not to explore the best way how to run the experiment, we only need a code executable in reasonable time that fulfill our purposes to find a representation or show that there is no such one. Moreover there exist maybe better environments where we can execute the experiment such as Magma Computational Algebra System or we can even build the structure on our own in the C programming language, but SageMath is after all enough for our purposes.

3.4.1 Summary of procedure

We start the description of the experiment at the point where we already got through theoretic understanding of the settings described in Section 1.4 and the motivation in Section 3.1 with an overall analysis described in Section 3.3. Then the procedure can be divided into 3 parts:

1. Preparatory tasks
2. Run through all possibilities
3. Afterprocessing of generated data

Last part of the experiment is discussed in Section 3.4.2 and we focus here on first two.

The preparatory tasks consist of generating multiple data that will be used later. First of all we have to generate parameters $\epsilon_q, \epsilon_2, \epsilon_1, \delta$ described in Section 3.1 to be able to construct ψ_1, ψ_2 described as well in Section 3.1. These will be the initial functions that will be composed then with projective transformations.

We used the code that can be seen in Attachment A.7 to generate them. We already mentioned a method of generation of needed constants at the end of Section 3.1. We can add here a comment on the code. Although SageMath has useful libraries and inbuilt functions that are able to manipulate algebraic structures as finite fields and polynomials, we had to cover transitions between a field and its subfield by our own. We can find such precedures in cell `[4]` of Attachment A.7 where for example function `element_E_L` takes an element from field `E` and gives corresponding element from field `L` if it is possible. As is it define it in Attachment A.2 we take a chain of subfields

$$\mathbb{K} = \mathbb{F}_{2^4} \subset \mathbb{L} = \mathbb{F}_{2^8} \subset \mathbb{E} = \mathbb{F}_{2^{16}}$$

We fix one choice of needed constants once we generate them at random within boundaries

- $\epsilon_2 \in \{e \in \mathbb{F}_{2^8} : \mathrm{tr}_{\mathbb{L}\setminus\mathbb{F}_2}(e) = 1\}$ arbitrary,
- $\omega \in \{e \in \mathbb{F}_{2^{16}} : e^2 + e = \epsilon_2\}$ arbitrary,
- $\epsilon_q = \omega^q + \omega$,
- $\delta = 0$
- $\epsilon_1 \in \{e \in \mathbb{F}_{2^8} : \mathrm{tr}_{\mathbb{D}\setminus\mathbb{F}_2}(e) = 1\}$ arbitrary

with values as printed in Attachment A.7

$$\begin{aligned}\epsilon_2 &= U^7 + U^5 + U^3 + U^2 + U\\ \epsilon_q &= U^4 + U^3\\ \delta &= 0\\ \epsilon_1 &= U^6 + U^2\end{aligned}$$

where U is used as a generator of $\mathbb{L}$.

Next step is to figure out how we will evaluate functions that we generate. The first approach was to generate ψ_1, ψ_2 in its polynomial representation and compose them in every iteration with current μ_1, μ_2 as we run through all of $\mu_1 = \frac{1}{x} + a$ and $\mu_2 = \frac{1}{x+d}$. Then we could manipulate the function to get the form we discussed in Section 3.3.3, the fractional jump with manipulation of the denominator, and then evaluate it for any purpose such as computation of invarinats.. This approach would consume a lot of time since manipulation with polynomials of high degree over a finite field takes SageMath a lot of time and that could be limiting for the experiment. Moreover we would compute a lot of times the same or similar thing instead of remembering them.

We choose another approach. Since we have fixed ψ_1, ψ_2, we can once generate their value table and then always only search in a dictionary defined over the finite field $\mathbb{L}$. Composition with translation is done by additional manipulation with elements from the finite field such as addition and inverse, that is again store in a dictionary. Such manipulations does not consume a lot of time as multiplication of a polynomials of high degree.

That is the motivation of generating dictionaries `psi_1`, `psi_1_inv`, `psi_2`, `psi_2_inv` and `inverses`, to remember data instead of calculate them again and again. Corresponding code is in Attachment A.9 in cell `[2]`.

We also leave a comment about the evaluation of ψ_1, ψ_2 itself. We straightforwardly follow the Definition 18. Corresponding code is in Attachment A.9 in cell `[1]`.

The last thing that we prepare before we run the experiment itself is a function that generates values of invariants. We chose to generate only the differential spectrum as defined in Definition 5 since it is easy and fast to compute. The degree of a function can be difficult to compute on our own and the Walsh spectrum would take a lot of time. These other invariants can be computed maybe later for those of which the differential spectrum fulfill the requirements. SageMath has a library `sage.crypto.sbox` that can directly compute cryptographical properties of an S-box given by a value table over integer ring, but for our purposes it has several defects.

Firstly, it accepts only value tables over integer rings as input. Converting our value table in dictionary over the finite field to a value table over integer ring takes some time. Secondly, during the computation with by the library function, a lot of RAM memory is consumed and it can led to a dead of SageMath kernel.

Therefore we compute the differential spectrum by our own code by function `my_dspectrum`, that gives the same result as the SageMath library function but suits our purposes better. Corresponding code is in Attachment A.9 in cell `[3]`. Especially, instead of computing the multiset as

$$\Delta_F = \{\#\{x \in \mathbb{F}_{2^8} | F(x+a) + F(x) = b\} | a, b \in \mathbb{F}_{2^8}\}$$

which takes $(2^8)^3$iterations, we generate the multiset as

$$\Delta_F = \{\#\{F(x+a) + F(x) | x \in \mathbb{F}_{2^8}\} | a \in \mathbb{F}_{2^8}\}$$

which gives the same result and takes only $(2^8)^2$ iterations.

Let us move on to the run through all iteration. We give 5 variants of the computation. All of them are discussed in Section 3.3.1 and 3.3.3. Firstly, we run through all

$$\frac{1}{x} \circ \Big(x + \psi_i(a)\Big) \circ \psi_i \circ \Big(x + a\Big) \circ \frac{1}{x}$$

for $\psi \in \{\psi_1, \psi_2\}$. Corresponding code is in Attachment A.10. We ensure that it goes through permutation of the affine space directly, without any additional manipulation, by choosing $d = \psi(a)$. We therefore run through both ψ_1, ψ_2, with inner for loop that run thourgh all parameter `a` and compute appropriate value table.

Value table is computed so that we make the less possible of manipulations and therefore from an element `l` we compute its preimage of μ_1 and store the value in variable `point`. Then we find the image of `l` by ψ and then its image for μ_2 and store the value in variable `value`. Then we simply assign value of `value` to the key `point` in the dictionary `value_table`.

After we have the whole value table generated we provide it to the function `my_dspectrum` and it computes the differential spectrum for a function with such value table. The differential spectrum is then compared with the one of the S-box π and stored in a file according to the result of comparison. Methodology of the comparison is described in Section 3.4.2.

We execute the run for 4 other variants of the experiment with modifications described in Section 3.3.3.2 in a similar way. For every of variants

$$\begin{aligned}
&L_2 \circ \psi \circ L_1 \\
&L_2 \circ \psi \circ \Big(x + a\Big) \circ \frac{1}{x} \circ L_1 \\
L_2 \circ \frac{1}{x} \circ \Big(x + d\Big) \circ \psi \circ\ &L_1 \\
L_2 \circ \frac{1}{x} \circ \Big(x + d\Big) \circ \psi \circ\ &\Big(x + a\Big) \circ \frac{1}{x} \circ L_1
\end{aligned}$$

we successively chose $\psi \in \{\psi_1, \psi_2\}$ and generate all value tables for all choices of parameters `a` and `d`. Code for all of them can be found in Attachment A.11 with the same order as above in cells `[2]`, `[3]`, `[4]`, `[5]`.

The process of computation is the same as for the special case with $d = \psi(a)$, but we had to add a steps in the middle that cover the manipulation around the point at infinity. First we have to identify the preimage of ∞ as we do it in the Example in Section 3.3.3 and then force the function to map that point to the point we want. As we work with the value table, it is quite easy, since we can write it directly to the dictionary and during the for loop that assign values, we can just avoid the critical point. Manipulation at the level of explicit representation can be done after, in the interpretation of data we generate.

In each cell from Attachment A.10 and A.11 we print the overall and half time consumed by the computation in the cell. Separate measured values with precision on seconds can be seen in following table.

variant	number of iterations	consumed time
$\frac{1}{x} \circ \Big(x + \psi(a)\Big) \circ \psi \circ \Big(x + a\Big) \circ \frac{1}{x}$	$2 \cdot 2^8$	0:02:34
ψ	2	<0:00:01
$\psi \circ \Big(x + a\Big) \circ \frac{1}{x}$	$2 \cdot 2^8$	0:02:44
$\frac{1}{x} \circ \Big(x + d\Big) \circ \psi$	$2 \cdot 2^8$	0:02:47
$\frac{1}{x} \circ \Big(x + d\Big) \circ \psi \circ \Big(x + a\Big) \circ \frac{1}{x}$	$2 \cdot 2^{16}$	3:37:50

Table 3.1: time complexity of the experiment

3.4.2 Interpretation of the result

In previous section, we described the procedure of the experiment. We run through all functions that can be obtained as a projective equivalent to chosed ψ_1, ψ_2. We want to find whether there is one such function that corresponds to our S-box π. To decide which ones are more similar to π than the others, we use comparison of the differential spectrum of those function. We store all differential spectrum in files so that we can attach them to the thesis and some of them are printed also in Attachment A.12.

By the first run of the experiment we found out that there is no such function with exactly the same differential spectrum. There is also one aspect that we did not considered yet. As we modify the majority of functions by the fractional jump, we might change the differential spectrum by that as well.

Let us take a look on how we compute the differential spectrum of a function and how it can change if we change one image. Generally, we get the differential spectrum of a function F as a multiset

$$\Delta = \{\#\{F(x + a) + F(x) | x \in \mathbb{F}_{2^8}\} a \in \mathbb{F}_{2^8}\}$$

and we store it in a dictionary in a form that corresponds to a multiset. By one change in the value table, forcing e.g. $\beta \mapsto \gamma$ we might change the value of

$$F(\beta + a) + F(\beta) \text{ and } F(\beta + a + a) + F(\beta + a)$$

for every $a \in \mathbb{F}_{2^8}$. This gives us up to $2 \cdot 2^8$ changed values which is . We can define a metric how to review the similarity.

Definition 34. *We define the* ***distance of differential spectrums*** $\Delta_{F_1}, \Delta_{F_2}$ *as*

$$\vartheta(\Delta_{F_1}, \Delta_{F_2}) = \sum_{b \in \mathbb{F}_{2^8}} |b'_1 - b'_2|$$

with

$$\Delta_{F_i} = \{(b, b'_i) | b \in \mathbb{F}_{2^8}, b' = \sum_{a \in \mathbb{F}_{2^8}} \delta_{F_i}(a, b)\}$$

for $i = 1, 2$*.*

We use the distance of differential spectrum of π and a function F as a metric of similarity. In case

$$\vartheta(\Delta_F, \Delta_\pi) \leq 512$$

we categorize F as a function similar to π. We can see in the code in Attachment A.10 and A.11, that we store the records about found differential spectrum in multiple files and we sort them by value of $\vartheta(\Delta_F, \Delta_\pi)$. However, no function F with $\vartheta(\Delta_F, \Delta_\pi) \leq 512$ was found and therefore files `invariants matches.txt`, `invariants similar.txt` as well as `invariants matches special.txt` and `invariants similar special.txt` stays empty and we do not even attach them to the thesis. Files ”invariants.txt” and ”invariants special.txt” contains all found differential spectrum, the file ”invariants special.txt” contains records that are also contained in file ”invariants.txt”.

Conclusion

The main goal of this thesis was to find whether the S-Box π used in the cipher Kuznyechik and hash function Streebog is affine equivalent to a fractional q-projective permutation which was introduced in [Gö22] and has a simple form. Such an equivalence could be exploited to devise an attack on the cipher. The motivation was mainly based on a previously published paper [Per19], where a significant structure was found. We described in Chapter 2 the procedure of the block cipher Kuznyechik and the hash function Streebog and we summarised findings of [Per19] around the partition-preserving property and cryptographical properties of S-box π that are uncovered by the TKlog structure.

As it seemed that the structure has a connection with fractional q-projective functions from [Gö22], we designed an experiment in Chapter 3, in which we want to discover whether there is a fractional q-projective function that represents a function affine-equivalent to the S-box π. We developed a way how to make the experiment doable in reasonable time and how to deal with various aspects of fractional polynomials and equivalencies over the affine and projective space. The experiment itself did not find any fractional q-projective representation, but the generated data may be analysed to find a fractional q-projective function that has similar properties.

We give also elementary representations of S-box π in Section 3.2 as an algebraic normal form, the Lagrange interpolation polynomial or one single formula, that encapsulates the definition of TKlog by Perrin, given as a formula of 3 parts. None of those representation did give us any information that can be used in an attack.

Another idea that is not mentioned in the thesis but can be related to the topic, is to reformulate Perrin's definition of TKlog, with help of elements from a subgroup of all (2^q+1)-th roots of 1, the set $\langle \alpha^{2^4-1} \rangle$ instead of a subgroup of all (2^q-1)-th roots of 1, the set $\langle \alpha^{2^4+1} \rangle$ or their combinations.

Bibliography

[BPU16] Alex Biryukov, Léo Perrin, and Aleksei Udovenko. Reverse-Engineering the S-Box of Streebog, Kuznyechik and STRIBOBr1. *Cryptology ePrint Archive, Report 2016/071*, 2016.

[DD13] Vasily Dolmatov and Alexey Degtyarev. GOST R 34.11-2012: Hash Function. RFC 6986, August 2013.

[Dol16] Vasily Dolmatov. GOST R 34.12-2015: Block Cipher ”Kuznyechik”. RFC 7801, March 2016.

[eS49] Claude e. Shannon. Communication theory of secrecy systems. *Bell Systems Technical Journal 28*, pages 656 – 715, 1949.

[GM18] Federico A. Guidi and Giacomo Micheli. Fractional jumps: Complete characterisation and an explicit infinite family. *arXiv:1805.11658*, 2018.

[Gö22] F. Göloglu. Classification of fractional projective permutations over finite fields. *Finite Fields Appl.*, 2022.

[KR11] Lars Knudsen and M Robshaw. *The Block Cipher Companion.* Information Security and Cryptography. Springer, 2011.

[MvOV01] Alfred J. Menezes, Paul C. van Oorschot, and Scott A. Vanstone. *Handbook of Applied Cryptography.* Fifth Printing. CRC Press, 2001.

[Per19] Léo Perrin. Partition in the S-box of Streebog and Kuznyechik. *IACR Transaction on Symmetric Cryptography*, pages 302–329, 2019.

[PU16] Léo Perrin and Aleksei Udovenko. Exponential S-Boxes: a Link Between the S-Boxes of BelT and Kuznyechik/Streebog. *IACR Transaction on Symmetric Cryptography*, pages 99–124, 2016.

[SB11] D. Stanovský and L. Barto. *Počítačová algebra.* Druhé opravené vydání. Matfyzpress, Praha, 2011.

[WF16] Chuan-Kun Wu and Dengguo Feng. *Boolean Functions and Their Applications in Cryptography.* Springer, Berlin, 2016.

[Wil08] Fulton William. *Algebraic Curves.* Third preface. Addison-Wesley, 2008.

List of Figures

List of Tables

A. Attachments

A.1 S-box π as a lookup table

```
pi=[
252,238,221,17,207,110,49,22,251,196,250,218,35,197,4,77,
233,119,240,219,147,46,153,186,23,54,241,187,20,205,95,193,
249,24,101,90,226,92,239,33,129,28,60,66,139,1,142,79,
5,132,2,174,227,106,143,160,6,11,237,152,127,212,211,31,
235,52,44,81,234,200,72,171,242,42,104,162,253,58,206,204,
181,112,14,86,8,12,118,18,191,114,19,71,156,183,93,135,
21,161,150,41,16,123,154,199,243,145,120,111,157,158,178,177,
50,117,25,61,255,53,138,126,109,84,198,128,195,189,13,87,
223,245,36,169,62,168,67,201,215,121,214,246,124,34,185,3,
224,15,236,222,122,148,176,188,220,232,40,80,78,51,10,74,
167,151,96,115,30,0,98,68,26,184,56,130,100,159,38,65,
173,69,70,146,39,94,85,47,140,163,165,125,105,213,149,59,
7,88,179,64,134,172,29,247,48,55,107,228,136,217,231,137,
225,27,131,73,76,63,248,254,141,83,170,144,202,216,133,97,
32,113,103,164,45,43,9,91,203,155,37,208,190,229,108,82,
89,166,116,210,230,244,180,192,209,102,175,194,57,75,99,182]
```

A.2 Auxiliary functions and used modules in SageMath

```
[1]: import sage.rings.finite_rings
     L = GF(2^8,'U'); LL=list(L)
     R.<x>=PolynomialRing(L)
     import sage.matrix
     import random
     from datetime import datetime
     from sage.crypto.sbox import SBox
```

```
[2]:
     #auxiliary functions
     carrierset=[i[1] for i in enumerate(R)]
     def log_GF(a):
         return carrierset.index(a)
     def exp_GF(a):
         if a==0:
             return 1
         else:
             return carrierset[a]
     def Int_to_Vec(a,n):
         vec_a=Integer(a).digits(2)
         vec_a.reverse()
         while len(vec_a)<n:
             vec_a.insert(0,0)
         return vec_a
     def Vec_to_Int(a):
         r=0
         for i in range(len(a)):
             if a[len(a)-i-1]:
                 r+=(2^i)
         return r
     def Vec_to_GF(x):
         r=0
         for i in range(len(x)):
             if x[len(x)-i-1]:
                 r+=exp_GF(i)
         return r
     def GF_to_Vec(x):
         r=[]
         for i in range(7):
             if x>=exp_GF(7-i):
                 r.append(1)
                 x-=exp_GF(7-i)
             else:
                 r.append(0)
         r.append(x)
         return r
```

A.3 Linear layer of Kuznyechik

$$\begin{aligned}
R :& (\mathbb{F}_2^8)^{16} \longrightarrow (\mathbb{F}_2^8)^{16} \\
& R(a_{15}||\dots||a_0) = l(a_{15},\dots,a_0)||a_{15}||\dots||a_1 \text{ where} \\
l :& (\mathbb{F}_2^8)^{16} \longrightarrow \mathbb{F}_2^8 \\
& l(a_{15},\dots,a_0) = \nabla(l'(\Delta(a_{15}),\dots,\Delta(a_0))) \text{ where} \\
l' :& Q^{16} \longrightarrow Q \\
& l'(a'_{15},\dots,a'_0) = \langle \overline{b}, \overline{a'} \rangle \\
& \text{with } \overline{a'} = (a'_{15},\dots,a'_0) \\
& \overline{b} = (148, 32, 133, 16, 194, 192, 1, 251, 1, 192, 194, 16, 133, 32, 148, 1), \\
\Delta :& \mathbb{F}_2^8 \longrightarrow Q \\
& \Delta(a) = \Delta(a_7,\dots,a_0) = \Sigma_{i=0}^{7} a_i \theta^i \\
\nabla :& Q \longrightarrow \mathbb{F}_2^8 \\
& \nabla = \Delta^{-1}
\end{aligned}$$

A.4 S-box π as a TKlog in SageMath

```
[1]: def Kappa(vec_a): #vec_a in Z_2^4
         b=[1,1,1,1,1,1,0,0]
         A=Matrix(GF(2),[
         [0,0,1,1, 0,0,0,0],
         [0,0,1,0, 0,1,0,0],
         [0,0,1,0, 0,1,1,0],
         [0,0,0,1, 0,0,1,0]
                ]).transpose()

         return (A*vector(GF(2),vec_a))+vector(GF(2),b)

     def perm_s(a):
         s=[0,12,9,8,7,4,14,6,5,10,2,11,1,3,13]
         return s[a]

[2]:
     def TKlog(int_a): #int_a in Z_{2^8}
         a=Vec_to_GF(Int_to_Vec(int_a,8))
         if a==0:
             return Vec_to_Int(
                 Kappa(
                     Int_to_Vec(0,4)
                 ))
         else:
             k=log_GF(a)
             i=k%17
             j=k//17

             if i==0:
                 return Vec_to_Int(
                     Kappa(
                         Int_to_Vec(16-j,4)
                     ))
             else:
                 return Vec_to_Int(
                     GF_to_Vec(
                         Vec_to_GF(
                             Kappa(
                                 Int_to_Vec(16-i,4)
                             ))
                         +
                         exp_GF(17*perm_s(j))
                     ))
```

A.5 Algebraic normal form of π in SageMath

```
[1]: def Moebius_matrix(k):
         if k==0:
             return matrix(GF(2),[1])
         else:
             m=Moebius_matrix(k-1)
             return block_matrix(GF(2),[[m,m],[0,m]],subdivide=False)
```

```
[2]:
     matrix_of_values=[]
     for i in range(len(Pi)):
         matrix_of_values.append(Int_to_Vec(Pi[i],8))
     matrix_of_values=matrix(matrix_of_values).transpose()

     M=Moebius_matrix(8)
     ANF=matrix_of_values*M
     ANF.str()
```

```
[2]:
     [1 0 0 1 0 1 1 0 0 0 0 1 1 0 1 1 0 1 0 0 0 1 1 0 1 1 1 0 1 1 0 1 0
     1 1 0 0 1 0 1 0 0 0 1 1 0 1 1 1 1 1 1 1 1 0 0 1 0 0 0 0 0 0 0 0 1
     1 0 0 0 1 0 0 0 0 0 1 1 0 0 0 1 0 0 1 1 0 1 1 1 1 1 0 0 0 1 1 1 1
     1 0 1 0 1 1 1 0 0 1 0 1 0 1 0 0 1 1 1 1 1 0 1 1 1 1 1 1 0 0 0 1 0
     1 0 0 1 0 1 1 1 1 0 1 0 0 0 1 0 0 0 1 1 1 1 1 1 1 1 0 0 0 1 1 1 0
     0 0 0 1 0 0 1 0 0 0 1 1 1 0 1 1 1 0 1 0 0 0 0 1 0 1 1 1 1 1 0 0 1
     0 1 0 1 0 0 1 1 0 0 1 1 0 1 1 1 1 0 1 1 0 1 1 0 1 1 1 1 0 0 1 0 0
     1 1 1 0 0 0 0 0 1 0 0 0 1 0 1 0 1 1 1 1 1 1 1 1 0]

     [1 0 0 1 0 0 1 1 0 0 0 1 1 1 1 1 0 0 0 1 1 0 1 1 1 0 1 0 0 0 1 1 0
     1 0 0 0 1 1 1 1 1 0 1 1 0 1 1 1 1 0 0 0 1 0 1 0 1 0 1 0 1 1 1 0 1
     1 1 0 1 0 0 0 0 1 1 1 0 0 1 1 0 1 1 1 0 1 1 1 0 0 0 0 1 0 1 1 0 1
     0 0 1 0 0 0 0 1 0 0 1 0 0 0 0 1 1 1 0 1 0 0 1 1 0 0 1 0 0 0 0 1 1
     1 0 1 1 0 0 1 1 0 0 0 0 0 1 1 0 0 0 0 1 1 1 1 0 1 0 1 1 1 1 0 0 1
     1 1 1 1 1 0 1 1 0 1 0 1 1 1 1 1 1 1 1 0 1 0 0 1 0 1 0 1 0 0 1 1 0
     0 1 0 1 1 1 0 1 1 1 0 1 1 0 0 1 1 1 0 0 0 1 0 0 0 1 0 0 0 0 1 1 1
     0 1 1 0 0 1 0 1 0 0 0 0 1 0 1 1 1 0 1 0 0 0 0 0 0]

     [1 0 1 0 1 1 0 0 0 1 1 0 1 1 1 1 0 0 1 1 0 0 0 1 1 0 0 0 0 1 0 1 0
     1 1 0 1 1 0 1 1 0 0 1 1 1 0 1 1 1 1 0 1 0 1 0 0 1 0 1 0 0 0 1 0 0
     1 1 1 0 1 1 0 1 1 1 1 0 1 0 0 0 0 0 1 1 1 1 1 0 0 1 0 1 1 0 1 0 1
     1 1 0 1 1 0 0 0 1 0 1 1 0 0 0 1 0 0 1 0 1 1 0 0 1 0 1 1 0 1 1 0 1
     0 0 0 1 0 1 0 1 1 1 1 1 1 0 0 0 1 1 0 1 0 0 0 1 0 1 1 1 1 1 0 0 1
     1 1 0 0 0 0 1 1 0 0 1 0 1 1 1 1 0 0 1 1 1 1 1 1 1 1 1 0 1 1 1 0 0
     0 1 1 1 1 1 0 1 1 0 0 1 1 1 1 1 0 0 0 1 1 1 0 0 1 0 1 0 1 0 1 1 0
     1 1 0 0 1 0 1 1 0 1 0 0 1 0 1 0 0 1 1 1 1 0 0 1 0]

     [1 1 0 1 1 1 1 1 0 0 0 0 0 0 1 0 1 0 1 0 0 1 0 1 1 1 1 1 1 1 0 0 0
     1 1 0 0 0 0 1 1 1 0 1 1 0 1 0 0 0 0 1 1 0 1 1 0 0 1 1 1 1 0 1 1 0
     0 1 1 0 1 1 1 0 1 1 0 0 1 1 0 1 0 1 1 0 0 0 0 1 1 0 0 1 0 0 1 1 1
     0 0 0 0 0 0 0 1 1 1 1 1 1 1 0 1 0 0 0 0 0 0 0 1 0 0 0 1 1 0 1 1 1
     1 0 1 0 0 0 1 0 0 0 1 1 0 0 0 1 1 0 0 1 0 0 1 0 1 1 1 1 1 0 0 0 1
     1 1 1 0 0 1 0 1 1 0 0 1 1 1 1 1 0 1 1 0 0 1 1 1 0 0 0 0 1 0 1 1 0
     1 0 0 1 0 1 1 0 0 1 1 1 0 1 0 0 1 0 0 1 1 0 1 1 1 1 0 0 1 1 1 1 1
     0 0 0 0 1 1 1 0 0 1 1 1 1 1 0 0 1 1 1 0 0 1 0 1 0]

     [1 0 0 1 0 0 1 1 0 1 0 0 1 1 1 1 0 1 1 1 1 0 1 0 1 0 1 1 0 0 0 1 0
     0 1 0 1 1 1 0 1 0 0 1 1 0 0 1 1 1 0 1 0 0 0 0 0 0 0 1 0 0 1 0 1
     0 1 0 1 1 1 1 1 1 0 0 1 0 1 1 0 0 0 0 1 1 1 1 1 0 1 0 0 0 1 1 1 1
     1 1 1 0 1 0 0 0 0 1 1 0 0 0 0 0 1 0 1 0 1 1 0 0 0 0 0 0 0 0 1 1 1
     0 1 1 0 1 1 1 1 0 0 1 1 1 1 1 0 0 1 1 0 1 1 1 0 0 0 0 0 1 1 0 0 0
     1 0 0 1 0 1 1 0 1 1 0 1 0 1 1 1 0 0 0 1 1 0 1 0 0 0 0 1 1 1 0 0 0
     0 1 0 0 1 1 0 0 1 0 0 0 0 0 0 1 0 0 0 1 1 0 0 0 0 1 0 1 0 0 1 1 0
     0 1 1 1 0 0 1 1 0 1 0 0 1 0 1 0 1 1 1 0 1 0 0 0 0]

     [1 0 0 1 0 0 1 0 1 1 0 0 0 0 0 0 1 1 0 0 0 0 1 0 0 0 1 1 0 0 1 1 1
     0 1 0 0 1 1 1 1 0 0 1 0 0 0 1 0 1 0 0 1 1 1 1 0 0 0 1 1 1 0 1 1 1
     1 1 0 1 0 0 1 0 1 0 1 0 1 1 0 1 1 1 1 1 1 1 0 1 1 0 0 1 0 1 0 0 0
     0 1 1 0 0 0 0 1 0 1 1 0 0 0 1 1 1 0 0 0 0 0 1 0 0 1 1 0 0 0 0 0 0
     0 1 0 0 1 0 0 0 0 1 0 0 0 0 1 0 0 1 1 1 1 1 1 1 0 1 0 0 1 0 0 1 0
     1 0 1 0 1 1 1 1 0 0 0 1 0 0 1 1 1 1 1 0 0 1 1 1 1 0 1 1 0 0 1 0 1
     0 1 0 1 0 1 1 1 1 1 0 1 1 0 0 1 1 0 1 1 1 0 0 1 0 1 1 1 1 0 0 1 0
     0 0 1 1 0 0 0 1 0 0 0 0 1 1 1 1 0 0 0 1 1 0 1 0 0]

     [0 1 0 1 1 1 1 0 1 0 0 0 1 1 0 0 0 0 0 1 0 0 0 1 0 1 1 1 1 0 1 1 0
     1 0 0 0 0 1 1 1 0 0 0 1 1 0 1 0 0 1 0 0 1 1 1 1 1 1 1 0 1 1 0 1 0
     1 0 1 1 1 1 1 1 0 0 0 0 0 0 1 1 0 0 0 0 0 0 1 0 0 1 1 0 1 1 1 0 0
     0 0 1 1 0 0 0 0 1 1 1 0 1 0 0 1 1 0 0 0 0 0 0 0 0 0 1 0 1 1 0 1 0
     1 1 0 1 1 0 1 0 0 1 1 0 1 0 1 0 1 0 0 1 0 0 0 1 0 0 1 0 0 0 0 0 0
     1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 1 1 0 1 1 1 1 0 1 1 0 0 1 1 1
     1 0 0 1 0 0 1 1 0 1 1 1 0 1 1 0 0 0 0 1 0 1 1 1 1 0 0 0 1 1 0 1 1
     1 0 1 0 1 1 1 0 0 1 1 0 0 1 0 1 0 0 1 0 0 1 0 1 0]

     [0 0 1 0 1 1 1 0 1 1 0 1 1 0 1 0 1 0 0 1 1 0 0 1 1 0 1 1 0 1 1 1 1
     1 1 0 0 0 0 0 1 1 1 0 0 0 0 0 1 0 1 0 0 1 0 0 0 0 0 1 0 1 0 1 1 1
     0 0 0 0 0 1 0 0 1 1 1 0 1 0 1 0 0 0 1 0 0 1 0 1 1 1 0 0 1 0 1 0 1
     1 0 0 0 0 0 0 0 0 1 0 1 0 0 1 1 1 0 1 1 0 0 1 0 1 0 0 0 0 1 0 0 1
     0 1 1 1 1 1 0 0 1 0 1 1 0 1 1 1 0 1 0 1 1 1 1 1 0 1 1 0 1 1 1 0 0
     0 1 0 0 1 0 0 1 1 1 0 0 1 0 1 0 1 0 0 0 0 1 1 0 0 1 1 1 0 1 1 0 1
     1 0 1 0 0 0 0 1 0 0 0 0 1 0 0 1 1 0 1 0 1 1 1 0 0 1 0 0 0 1 0 0 1
     0 1 1 1 1 1 0 0 1 0 1 0 1 0 1 0 0 0 0 1 1 0 1 0 0]
```

A.6 Lagrange polynomial of π in SageMath

```
[1]: inv=2^8-2
     zeropoly=x^(2^8-1)-x
     def lagrange_pol(i,R):
         f=1
         for j in R:
             if i!=j:
                 f=(f*(x-j)*((i-j)^inv))%zeropoly
         return f
     def inverse(pol):
         b=pol^inv
         b=b%zeropoly
         return b
```

```
[2]: Pi_as_Lagrange=0
     zacatek=datetime.now()
     for prvek in LL:
         fc_value=Vec_to_GF(Int_to_Vec(
         Sbox_Pi[Vec_to_Int(GF_to_Vec(prvek))]
         ,8))
         Pi_as_Lagrange+=lagrange_pol(prvek,LL)*fc_value
     print("consumed time:",datetime.now()-zacatek)
```

```
consumed time: 0:04:56.769572
```

```
[3]: print("degree of the Lagrange polynomial for pi :",Pi_as_Lagrange.
      ↪degree())
     factors=list(Pi_as_Lagrange.factor())
     print("number of factors :",len(factors))
     for i in range(len(factors)):
         print("degree of factor f{0} : {1} of power:{2}".
      ↪format(i,factors[i][0].degree(),factors[i][1]))
     print("inverses :")
     for i in range(len(factors)):
         factor_inverse=inverse(factors[i][0])
         print("degree of inverse of factor f{0} : {1}".
      ↪format(i,factor_inverse.degree()))
```

```
degree of the Lagrange polynomial for pi : 254
number of factors : 4
degree of factor f0 : 1 of power:1
degree of factor f1 : 3 of power:1
degree of factor f2 : 19 of power:1
degree of factor f3 : 231 of power:1
inverses :
degree of inverse of factor f0 : 254
degree of inverse of factor f1 : 254
degree of inverse of factor f2 : 254
degree of inverse of factor f3 : 254
```

A.7 Setting of constants for main experiment in SageMath

```
[1]: m=4; n=2*m
     q=2^m; r=2^n; ee=r^2

     E=GF(ee,'V'); V=E.gen()
     L=GF(r,'U'); U=L.gen()
     K=GF(q,'u'); u=K.gen()
```

```
[2]:
     carrierset_K=[]
     for i in enumerate(K):
         if i[0]==0:
             carrierset_K.append(0)
         else:
             carrierset_K.append(u^i[0])

     def log_GF_K(a):
         return carrierset_K.index(a)

     carrierset_L=[]
     for i in enumerate(L):
         if i[0]==0:
             carrierset_L.append(0)
         else:
             carrierset_L.append(U^i[0])

     def log_GF_L(a):
         return carrierset_L.index(a)

     carrierset_E=[]
     for i in enumerate(E):
         if i[0]==0:
             carrierset_E.append(0)
         else:
             carrierset_E.append(V^i[0])

     def log_GF_E(a):
         return carrierset_E.index(a)
```

```
[3]:
     def Trac(xx,order_of_field1,order_of_field2):
         value=0
         l=log(order_of_field1,2)
         d=log(order_of_field2,2)
         for j in range(l/d):
             value+=xx^(2^(d*j))
         return value
```

[4]:
```
def element_E_in_L(a):
    index_a_L=log_GF_E(a)
    if index_a_L%(r+1)==0:
        index_a_K=index_a_L//(r+1)
        return U^index_a_K
    else:
        return False

def element_L_in_K(a):
    index_a_L=log_GF_L(a)
    if index_a_L%(q+1)==0:
        index_a_K=index_a_L//(q+1)
        return u^index_a_K
    else:
        return False

def element_K_in_L(a):
    index_a_K=log_GF_K(a)
    index_a_L=index_a_K*(q+1)
    return U^index_a_L

def element_L_in_E(a):
    index_a_L=log_GF_L(a)
    index_a_E=index_a_L*(r+1)
    return V^index_a_E
```

[5]:
```
U_1=[X for X in L if Trac(X,r,2)==1]
e_2=random.choice(U_1); print("e_2 =",e_2)

e_2_E=element_L_in_E(e_2)
W=[Y for Y in E if Y^2+Y==e_2_E]
w=random.choice(W)

e_q_E=w^q+w
e_q=element_E_in_L(e_q_E); print("e_q =",e_q)

delta=0; print("delta =",delta)

V_1=[X for X in K if Trac(X,q,2)==1]
e_1_K=random.choice(V_1)
e_1=element_K_in_L(e_1_K); print("e_1 =",e_1)
```

```
e_2 = U^7 + U^5 + U^3 + U^2 + U
e_q = U^4 + U^3
delta = 0
e_1 = U^6 + U^2
```

A.8 Invariants of π in SageMath

[1]:
```
pi_SBox=SBox(pi)
```

[2]:
```
def DDT_multiset(SBox_for_f):
    DDT_f=SBox_for_f.difference_distribution_table().
 ↪coefficients()
    d={i:0 for i in set(DDT_f)} #dictionary with 0 values
    for i in DDT_f:
        d[i] += 1
    return(d)

def LAT_multiset(SBox_for_f):
    LAT_f=SBox_for_f.linear_approximation_table().coefficients()
    l={abs(i):0 for i in set(LAT_f)} #dictionary with 0 values
    for i in LAT_f:
        l[abs(i)] += 1
    return(l)
```

[3]:
```
zerotime=datetime.now()
print("Degree :",pi_SBox.max_degree(),"\nconsumed time :
 ↪",datetime.now()-zerotime)
zerotime=datetime.now()
print("Differential spectrum :",DDT_multiset(pi_SBox),"\nconsumed␣
 ↪time :",datetime.now()-zerotime)
zerotime=datetime.now()
print("Walsh spectrum :",LAT_multiset(pi_SBox),"\nconsumed time :
 ↪",datetime.now()-zerotime)
```

```
Degree : 7
consumed time : 0:00:00.495818
Differential spectrum : {256: 1, 2: 22454, 4: 4377, 6: 444, 8: 25}
consumed time : 0:00:00.045100
Walsh spectrum : {128: 1, 2: 11645, 4: 10761, 6: 10166, 8: 8793,␣
 ↪10: 6804, 12:
4474, 14: 2796, 16: 1693, 18: 971, 20: 535, 22: 219, 24: 91, 26:␣
 ↪39, 28: 14}
consumed time : 0:00:00.193512
```

A.9 Auxiliary function for main experiment

```
[1]: def Evaluate(poly_up,poly_down,point):
         if point=="inf":
             if poly_up.degree()==poly_down.degree():
                 value_res=poly_up.list()[poly_up.degree()]*(poly_down.
     ↪list()[poly_down.degree()]^inv)
             elif poly_up.degree()>poly_down.degree():
                 value_res="inf"
             else:
                 value_res=0
         else:
             value_up=poly_up(point)
             value_down=poly_down(point)
             if value_down==0:
                 value_res="inf"
             else:
                 value_down_inv=value_down^inv
                 value_res=value_up*value_down_inv
         return value_res
```

```
[2]:
     psi_1_up=x^17 + (e_q+1)*x + e_2 + delta + e_1
     psi_2_up=x^17 + (e_q+1)*x + e_2 + delta
     psi_down=x^16 + x + e_q
     psi_1={a:Evaluate(psi_1_up,psi_down,a) for a in LL}
     psi_1_inv={Evaluate(psi_1_up,psi_down,a):a for a in LL}
     psi_2={a:Evaluate(psi_2_up,psi_down,a) for a in LL}
     psi_2_inv={Evaluate(psi_2_up,psi_down,a):a for a in LL}
     inverses={a:a**inv for a in LL}
```

```
[3]:
     def my_dspectrum(value_table):
         d_spectrum={}
         for a_GF in LL:
             DDT={}
             for i_GF in LL:
                 d=value_table[L(i_GF)]
                 d+=value_table[L(i_GF)+L(a_GF)]
                 if d in DDT:
                     DDT[d]+=1
                 else:
                     DDT[d]=1
             for d_s in DDT:
                 if DDT[d_s] in d_spectrum:
                     d_spectrum[DDT[d_s]]+=1
                 else:
                     d_spectrum[DDT[d_s]]=1
             del(DDT)
         del(value_table)
         return(d_spectrum)
```

A.10 Special case with $d = \psi(a)$ in SageMath

```
[1]: f1=open("invariants special.txt","a")
     f2=open("invariants matches special.txt","a")
     zacatek=datetime.now()

     for i in range(2):
         if i==0:
             psi=psi_1
             psi_inv=psi_1_inv
         else:
             psi=psi_2
             psi_inv=psi_2_inv
         for a in LL:
             d=psi[a]
             value_table={}
             for l in LL:
                 point=inverses[l+a]     #1/(x+a)=(mu_1)^-1
                 image=inverses[psi[l]+d]
                 value_table[point]=image
             tmp=my_dspectrum(value_table)
             if tmp==pi_dspectrum:
                 f2=open("invariants match special.txt")
                 f2.write("(1/x)(x+{0})psi{1}(x+{2})(1/x), {3};\n".
      ↪format(d,i+1,a,tmp))
                 f2.close()
             else:
                 similarity=0
                 for b in pi_dspectrum.keys():
                     if b in tmp.keys():
                         similarity+=abs(pi_dspectrum[b]-tmp[b])
                     else:
                         similarity+=pi_dspectrum[b]
                 for b in tmp.keys():
                     if b not in pi_dspectrum.keys():
                         similarity+=tmp[b]
                 if similarity<=512:
                     f3=open("invariants similar special.txt","a")
                     f3.write("psi{0}(x+{1})(1/x), {2};\n".
      ↪format(i+1,a,tmp))
                     f3.close()
                 else:
                     f1.write("(1/x)(x+{0})psi{1}(x+{2})(1/x), {3};\n".
      ↪format(d,i+1,a,tmp))
         print(datetime.now()-zacatek)
     f1.close()
```

```
0:01:17.650238
0:02:34.724217
```

A.11 Main experiment in SageMath

```
[1]: pi_GF={preimage:
      ↪Vec_to_GF(Int_to_Vec(pi[Vec_to_Int(GF_to_Vec(preimage))],8))␣
      ↪for preimage in LL}
     pi_dspectrum=my_dspectrum(pi_GF)
```

```
[2]:
     #variant psi_1, psi_2

     f1=open("invariants.txt","a")
     zacatek=datetime.now()

     for i in range(2):
         if i==0:
             psi=psi_1
             psi_inv=psi_1_inv
         else:
             psi=psi_2
             psi_inv=psi_2_inv
         tmp=my_dspectrum(psi)
         if tmp==pi_dspectrum:
             f2=open("invariants match.txt","a")
             f2.write("psi{0}, {1};\n".format(i+1,tmp))
             f2.close()
         else:
             similarity=0
             for b in pi_dspectrum.keys():
                 if b in tmp.keys():
                     similarity+=abs(pi_dspectrum[b]-tmp[b])
                 else:
                     similarity+=pi_dspectrum[b]
             for b in tmp.keys():
                 if b not in pi_dspectrum.keys():
                     similarity+=tmp[b]
             if similarity<=512:
                 f3=open("invariants similar.txt","a")
                 f3.write("psi{0}, {1};\n".format(i+1,tmp))
                 f3.close()
             else:
                 f1.write("psi{0}, {1};\n".format(i+1,tmp))
         print(datetime.now()-zacatek)

     f1.close()
```

```
     0:00:00.327935
     0:00:00.629967
```

```
[3]: #variant psi_1 mu_1, psi_2 mu_1

     f1=open("invariants.txt","a")
     zacatek=datetime.now()

     for i in range(2):
         if i==0:
             psi=psi_1
         else:
             psi=psi_2
         for a in LL:
             value_table={}
             beta=0
             gamma=psi[a]
             value_table[beta]=gamma
             for l in LL:
                 point=inverses[l+a]     #1/(x+a)=(mu_1)^-1
                 image=psi[l]
                 value_table[point]=image
             tmp=my_dspectrum(value_table)
             if tmp==pi_dspectrum:
                 f2=open("invariants match.txt","a")
                 f2.write("psi{0}(x+{1})(1/x), {2};\n".
      ↪format(i+1,a,tmp))
                 f2.close()
             else:
                 similarity=0
                 for b in pi_dspectrum.keys():
                     if b in tmp.keys():
                         similarity+=abs(pi_dspectrum[b]-tmp[b])
                     else:
                         similarity+=pi_dspectrum[b]
                 for b in tmp.keys():
                     if b not in pi_dspectrum.keys():
                         similarity+=tmp[b]
                 if similarity<=512:
                     f3=open("invariants similar.txt","a")
                     f3.write("psi{0}(x+{1})(1/x), {2};\n".
      ↪format(i+1,a,tmp))
                     f3.close()
                 else:
                     f1.write("psi{0}(x+{1})(1/x), {2};\n".
      ↪format(i+1,a,tmp))
         print(datetime.now()-zacatek)
     f1.close()
```

```
0:01:21.962485
0:02:44.569614
```

[4]:
```
#variant mu_2 psi_1, mu_2 psi_2
f1=open("invariants.txt","a")
zacatek=datetime.now()
for i in range(2):
    if i==0:
        psi=psi_1
        psi_inv=psi_1_inv
    else:
        psi=psi_2
        psi_inv=psi_2_inv
    for d in LL:
        value_table={}
        beta=psi_inv[d]
        gamma=0
        value_table[beta]=gamma
        for l in LL:
            point=l
            image=inverses[psi[l]+d]
            value_table[point]=image
        tmp=my_dspectrum(value_table)
        if tmp==pi_dspectrum:
            f2=open("invariants matches.txt","a")
            f2.write("(1/x)(x+{0})psi{1}, {2};\n".
 ↪format(d,i+1,tmp))
            f2.close()
        else:
            similarity=0
            for b in pi_dspectrum.keys():
                if b in tmp.keys():
                    similarity+=abs(pi_dspectrum[b]-tmp[b])
                else:
                    similarity+=pi_dspectrum[b]
            for b in tmp.keys():
                if b not in pi_dspectrum.keys():
                    similarity+=tmp[b]
            if similarity<=512:
                f3=open("invariants similar.txt","a")
                f3.write("(1/x)(x+{0})psi{1}, {2};\n".
 ↪format(d,i+1,tmp))
                f3.close()
            else:
                f1.write("(1/x)(x+{0})psi{1}, {2};\n".
 ↪format(d,i+1,tmp))
    print(datetime.now()-zacatek)
f1.close()
```

```
0:01:25.116888
0:02:47.124847
```

[5]:
```
f1=open("invariants.txt","a")     #variant mu_2 psi_1 mu_1, mu_2␣
 ↪psi_2 mu_1
zacatek=datetime.now()
for i in range(2):
    if i==0:
        psi=psi_1
        psi_inv=psi_1_inv
    else:
        psi=psi_2
        psi_inv=psi_2_inv
    for a in LL:
        for d in LL:
            value_table={}
            beta=inverses[psi_inv[d]+a]
            gamma=inverses[psi[a]+d]
            value_table[beta]=gamma
            for l in LL:
                if l!=psi_inv[d]:
                    point=inverses[l+a]     #1/(x+a)=(mu_1)^-1
                    image=inverses[psi[l]+d]
                    value_table[point]=image
            tmp=my_dspectrum(value_table)
            if tmp==pi_dspectrum:
                f2=open("invariants matches.txt","a")
                f2.write("(1/x)(x+{0})psi{1}(x+{2})(1/x), {3};\n".
 ↪format(d,i+1,a,tmp))
                f2.close()
            else:
                similarity=0
                for b in pi_dspectrum.keys():
                    if b in tmp.keys():
                        similarity+=abs(pi_dspectrum[b]-tmp[b])
                    else:
                        similarity+=pi_dspectrum[b]
                for b in tmp.keys():
                    if b not in pi_dspectrum.keys():
                        similarity+=tmp[b]
                    if similarity<=512:
                        f3=open("invariants similar.txt","a")
                        f3.write("(1/x)(x+{0})psi{1}(x+{2})(1/
 ↪x), {3};\n".format(d,i+1,a,tmp))
                        f3.close()
                    else:
                        f1.write("(1/x)(x+{0})psi{1}(x+{2})(1/
 ↪x), {3};\n".format(d,i+1,a,tmp))
            print(datetime.now()-zacatek)
f1.close()
```

```
1:48:26.770032
3:37:50.116593
```

A.12 Example of differential spectrum

```
(1/x)(x+U^5 + U^4 + U^2 + U)psi1(x+0)(1/x), {256: 2, 4: 14464, 16:
↪ 448};
(1/x)(x+U^7 + U^5 + U^4 + U^2 + 1)psi1(x+U)(1/x), {256: 2, 4:␣
↪14464, 16: 448};
(1/x)(x+U^7 + U^6 + U^5 + U^4 + 1)psi1(x+U^2)(1/x), {256: 2, 16:␣
↪448, 4: 14464};
(1/x)(x+U^7 + U^6 + U^5 + 1)psi1(x+U^3)(1/x), {256: 2, 4: 14464,␣
↪16: 448};
(1/x)(x+U^5 + U^4 + U^3 + 1)psi1(x+U^4)(1/x), {256: 2, 4: 14464,␣
↪16: 448};
(1/x)(x+U^7 + U^5 + U^3 + U^2 + 1)psi1(x+U^5)(1/x), {256: 2, 4:␣
↪14464, 16: 448};
(1/x)(x+U^2 + U + 1)psi1(x+U^6)(1/x), {256: 2, 4: 14464, 16: 448};
(1/x)(x+U^7 + U^6 + U^4 + U^3 + U)psi1(x+U^7)(1/x), {256: 2, 4:␣
↪14464, 16: 448};
(1/x)(x+U + 1)psi1(x+U^4 + U^3 + U^2 + 1)(1/x), {256: 2, 4:␣
↪14464, 16: 448};
(1/x)(x+U^5 + U^2 + 1)psi1(x+U^5 + U^4 + U^3 + U)(1/x), {256: 2,␣
↪4: 14464, 16: 448};
(1/x)(x+U^7 + U^3 + U + 1)psi1(x+U^6 + U^5 + U^4 + U^2)(1/x),␣
↪{256: 2, 4: 14464, 16: 448};
(1/x)(x+U^6 + U^2 + 1)psi1(x+U^7 + U^6 + U^5 + U^3)(1/x), {256:␣
↪2, 4: 14464, 16: 448};
(1/x)(x+U^7 + U^4 + U^2)psi1(x+U^7 + U^6 + U^3 + U^2 + 1)(1/x),␣
↪{256: 2, 16: 448, 4: 14464};
(1/x)(x+U^7 + U^5 + U^4 + U^3 + U)psi1(x+U^7 + U^2 + U + 1)(1/x),␣
↪{256: 2, 16: 448, 4: 14464};
(1/x)(x+U^7 + U^5 + U^4 + U^3 + 1)psi1(x+U^4 + U + 1)(1/x), {256:␣
↪2, 4: 14464, 16: 448};
(1/x)(x+U^7 + U^6 + U^5 + U^4 + U^3 + U^2 + U)psi1(x+U^5 + U^2 +␣
↪U)(1/x), {256: 2, 16: 448, 4: 14464};
(1/x)(x+U^7 + U^5 + U^3)psi1(x+U^6 + U^3 + U^2)(1/x), {256: 2, 4:␣
↪14464, 16: 448};
(1/x)(x+U^4 + U^2)psi1(x+U^7 + U^4 + U^3)(1/x), {256: 2, 4:␣
↪14464, 16: 448};
(1/x)(x+U^6 + U^3 + U + 1)psi1(x+U^5 + U^3 + U^2 + 1)(1/x), {256:␣
↪2, 4: 14464, 16: 448};
(1/x)(x+U^6 + U^5 + U^2)psi1(x+U^6 + U^4 + U^3 + U)(1/x), {256:␣
↪2, 16: 448, 4: 14464};
(1/x)(x+U^5 + U^3 + U)psi1(x+U^7 + U^5 + U^4 + U^2)(1/x), {256:␣
↪2, 4: 14464, 16: 448};
(1/x)(x+U^7 + U^3 + U)psi1(x+U^6 + U^5 + U^4 + U^2 + 1)(1/x),␣
↪{256: 2, 4: 14464, 16: 448};
```