University of Trento

Department of Mathematics

On some computational aspects for hidden sums in Boolean functions

Carlo Brunetta

Supervisor: Prof. Massimiliano Sala
PhD. Marco Calderini

Academic Year 2015/2016

University of Trento

Department of Mathematics

On some computational aspects for hidden sums in Boolean functions

Thesis by:

Carlo Brunetta

Supervisor:

Prof. Massimiliano Sala

PhD. Marco Calderini

Academic Year 2015/2016

If a man's wit be wandering,
let him study the mathematics.

Francis Bacon

Contents

Introduction

We live in an age and society that surrounds us with information, and where our day-to-day lives increasingly depend upon this information and our ability to manipulate it.

For example, it is often taken for granted that we can control our bank accounts from almost anywhere in the world using a combination of satellite and cellular phone networks to talk to bank representatives, specialised wired ATM networks to withdraw money, and the Internet for online banking services.
Sadly, whenever there are services for manipulating information that has value, there will be unscrupulous elements in society that will seek to subvert these services for their own benefit. This has led to the development of research into information security.

Block ciphers combine simple operations to construct a complex encryption transformation. This tradition has its roots in Shannon's paper [Sha45] connecting cryptography with information theory.

Shannon suggested building a strong cipher system out of simple components that substantiate the so-called *confusion* and *diffusion* of data applying these components iteratively in a number of rounds.
Each of these components, seen as a single function, would be cryptographically weak and only their composition can be strong. Feistel [Fei73] were the first to introduce a practical architecture based on Shannon's concepts. The most prominent example of a Feistel type cipher is probably the Data Encryption Standard (DES) [EG83].

Most modern block ciphers are built using components whose cryptographic strength is evaluated in terms of the resistance offered to attacks on the whole cipher.
In particular, linear and differential properties of Boolean functions are studied for the S-Boxes to thwart linear and differential cryptanalysis.
Little is known on similar properties to avoid trapdoors in the design of the block cipher.

By a trapdoor we mean the presence of a secret that, if known, allows to disclose the cipher, i.e. to read a ciphertext without knowing the key, or to compute the encryption key.

In the DES algorithm, no trapdoors have been found in more than 20 years, but many users are still suspicious about the DES S-boxes. The discussion of trapdoor issues has been directed towards individuating trapdoors in known ciphers.

A way to consider trapdoors is to employ (permutation) group theory: an iterated block cipher can be regarded as a set of permutations of a message space.
Some properties of the group generated by the round functions of such a cipher are known to be of cryptanalytic interest.

Kenneth Paterson [Pat99] has considered iterated block ciphers in which the group generated by the one-round functions acts imprimitively on the message space, with the aim of exploring the possibility that this might lead to the design of trapdoors.
In particular, Paterson constructed an example of a DES-like cipher where the group generated by the one-round functions is imprimitive.

In [CDVS09] the authors investigated the minimal properties for the S-Boxes (and the mixing layer) of an AES-like cipher (more precisely, a translation based cipher, or tb cipher) to thwart the trapdoor coming from the imprimitivity action. More refined group theory can be used to insert additional trapdoors.

In [Li03], Li observed that if V is a vector space over a finite field $\mathbb{F}_p$, the symmetric group $\mathrm{Sym}(V)$ will contain many isomorphic copies of the affine group $\mathrm{AGL}(V)$, which are its conjugates in $\mathrm{Sym}(V)$. So there are several structures $(V, \circ)$ of a $\mathbb{F}_p$-vector space on the set V , where $(V, \circ)$ is the abelian additive group of the vector space. Each of these structure will yield in general a different copy $\mathrm{AGL}_\circ(V)$ of the affine group within $\mathrm{Sym}(V)$.
Thus, if the group generated by the one-round functions of a block cipher is contained in a copy of $\mathrm{AGL}(V)$ this might lead to the design of trapdoors coming from alternative vector space structure, which we call hidden sums.

The thesis main goal is to improve the bound given in Proposition 2.1.27 of [Cal15] that counts the number of different abelian regular elementary subgroups of $\mathrm{AGL}(V)$ that can generate a hidden sum with constrains on subspaces dimension. We developed a new representation of hidden sums that simplify the searching and analysing algorithms and, with this representation, a lower bound is given in Proposition 3.3.1.

The thesis will be developed in three main chapter:

1. **Preliminaries** where there will be a thorough description of the algebraic structures, theorems and definition to achieve the cryptography definitions needed to understand the meaning of a trapdoor, as a concept and as a mathematical construction. There will be presented the latest results that can guarantee the absence of the hidden sum.

2. **Hidden sum cardinality** in which we will concentrate on mathematical questions about the hidden sums that can be created in ciphers. The thesis construct a simpler approach using linear algebra and construct optimized algorithms that permits to count the hidden sums, fixing different constructional dimensions of the cipher.

3. **Hidden sum algorithms** where there will be presented different optimized algorithms to build and archive new operations that are directly connected with hidden sums.
   As an additional research, we will analyse part of the modern cipher called PRESENT ([BKL+07]) that, in principle, could hide a hidden sum trapdoor.

In the appendix additional Magma code for the different algorithms presented in the thesis will be presented.

Notation and preliminaries

In this chapter will be presented all the notation used in the thesis and the preliminaries that define its starting point.
Details can be found in [Cal15], [Str09], [Lan93], [LN97], [Sha49], [Car11], [Car07],

The preliminaries will be divided into:

- **Algebraic preliminaries** including all the linear algebra and group theory used. It will introduce all the Boolean notation to complete, correctly define and describe the hidden sums and relative attack.
- **Cryptography preliminaries**, that will define the cryptographic primitivities and the concept of security in an algebraic sense.
- **Hidden sum** which will contain the description of the theory developed in [Cal15] to construct and describe the properties of the different sums used in the hidden sum attack. There will be an example of the attack as a working proof-of-concept.

2.1 Linear algebra and group theory terminology

In this subsection all the basic linear algebra, group theory definition and properties used in the thesis will be presented.

We denote with $1..n = \{1, \dots, n\}$ and with $\mathbb{F}_q$ the finite fields with q elements, where q is a power of a prime.
If not differently indicated, we denote $\mathbb{F} = \mathbb{F}_2$.
The cardinality of A will be denoted by $\#A$ or $|A|$.

With $\mathbb{F}_q^{n\times k}$ we denote the sets of all the $n \times k$ matrices with entries in $\mathbb{F}_q$.
Let M be a $n \times m$ matrix. The element in the i-th row and j-th column will be denoted by $M_{i,j}$ or $M[i][j]$.

e_i will represent the i-th canonical vector in $\mathbb{F}_q$ and the vectorial (sub)space generated by $v_1, \ldots, v_k$ will be denoted by $\text{Span}\{v_1, \ldots, v_k\}$.

$\text{Sym}(V)$ and $\text{Alt}(V)$ will indicate the symmetric and alternating group acting on V.
With $\langle g_1, \ldots, g_k \rangle$ we will denote the group generated by $g_1, \ldots, g_k$ in $\text{Sym}(V)$.
The action of g on an element x will be denoted by $(x)g$ or xg if there are no ambiguities.

With $\text{GL}(V)$ and $\text{AGL}(V)$ we will denote the linear and affine group of V.

Definition 2.1.1. Let G be a group acting on V.
G is called **transitive** if for all $x, y \in V$, exists $g \in G$ such that $xg = y$.

G is called **regular** if for all $x, y \in V$, exists unique $g \in G$ such that $xg = y$.

Note 2.1.1. G is regular if and only if G transitive and $\#G = \#V$

Definition 2.1.2. A partition $\mathcal{B}$ of V is G-invariant if for any $B \in \mathcal{B}$ and $g \in G$, we have $Bg \in \mathcal{B}$.
A partition $\mathcal{B}$ is **trivial** if $\mathcal{B} = \{V\}$ or $\mathcal{B} = \{\{v\} | v \in V\}$.

If $\mathcal{B}$ is not trivial and G-invariant then $\mathcal{B}$ is a **block system** for the action of G on V.
If there exists a block system, then we say that G is **imprimitive** in its action on V.
If G is not imprimitive (and it is transitive), then we say that G is **primitive**.

Definition 2.1.3. An element r of a ring R is called **nilpotent** if $r^n = 0$ for some $n \geq 1$.
$r \in R$ is called **unipotent** if $r - 1$ is nilpotent, so $(r-1)^n = 0$ for some $n \geq 1$.

Note 2.1.2. Let $G \subseteq \text{GL}(V)$ be a subgroup of unipotent permutations. Then G is called **unipotent**.

Definition 2.1.4. An element $k \in \text{GL}(V)$ is said **upper triangular** in a basis $v_1, \ldots, v_n$ if

$$\forall i \in 1..n. \quad v_i k - v_i \in \text{Span}\{v_{i+1}, \ldots, v_n\}$$

The upper triangular matrices in the canonical basis are called **upper unitriangular matrices**. We denote with $\mathcal{U}(V)$ the upper unitriangular matrices group.

Theorem 2.1.1. *Let G be a group consisting of unipotent matrices. Then there is a basis in which all elements of G are upper triangular.*

Define the set of invertible and symmetrical $n \times n$ matrices over a finite field $\mathbb{K}$ with null-diagonal as

$$\text{Sym0-GL}_n(\mathbb{K}) := \left\{ x \in \text{GL}_n(\mathbb{K}) \;\middle|\; \begin{array}{c} x \text{ has null diagonal, and} \\ x \text{ is symmetric} \end{array} \right\}$$

Theorem 2.1.2 (Thm. 2 [Mac69])**.** *Let $\mathbb{F}_q$ be a finite fields with $q = p^k$ and p prime, $k > 0$.*
Then the number of symmetric invertible matrices is

$$\text{sym}(n) = q^{\binom{n+1}{2}} \prod_{j=1}^{\lceil \frac{n}{2} \rceil} \left(1 - q^{1-2j}\right)$$

Theorem 2.1.3 (Thm. 3.3 [LLM+10])**.** *Let $\mathbb{F}_q$ be a finite fields with $q = p^k$ and p prime, $k > 0$.*
When n is even, the number of $(n-1) \times (n-1)$ symmetric invertible matrices is equal to the number of $n \times n$ symmetric invertible matrices with zero diagonal.

$$\text{sym}(n-1) = \text{sym}_0(n)$$

Definition 2.1.5. Let $M \in \mathbb{F}_q^{n \times n}$ be a $n \times n$ matrix with entries in $\mathbb{F}_q$.
M will be called **skew-symmetric** if $M_{i,j} = -M_{j,i}$ for all $i, j \in 1..n$.
If $\text{Char}(\mathbb{F}_q) = 2$, then M need to have $M_{i,i} = 0$ for $i \in 1..n$.

Observe that the first condition impose that the principal diagonal is null for field with characteristic different from 2.

Remark 2.1.1. There is no skew-symmetric invertible matrix M of dimension n odd over any field F.

Proof. The proof is described in [Bi14] and formally in [MMMM13].
From the Leibniz formula for determinants we have

$$\det M = \sum_{\sigma \in \mathrm{Sym}(n)} \mathrm{sgn}(\sigma) \prod_{i=1}^{n} M_{i,\sigma(i)}$$

Now, we have

$$\prod M_{i,\sigma^{-1}(i)} = \prod M_{\sigma(i),i} = (-1)^n \prod M_{i,\sigma(i)}$$

and that every permutation has the same sign of its inverse. So for n odd, every permutation cancels with its inverse in the Leibniz expansion except for the permutation that are their own inverse.
These are the product of disjoint transposition and, since n being odd, there must be a fixed point. So there will be a $M_{i,i}$ in the expansion and so, from the fact that M has the diagonal equals zero, it will not contribute in the expansion and we obtain $\det M = 0$. □

Remark 2.1.2.
In a field $\mathbb{F}_q$ with characteristic 2, skew-matrices are symmetric invertible matrices.

Proof.
Note that in characteristic 2, we have that $x = -x$ for every $x \in \mathbb{F}_q$.
From the definition of skew-matrices, we have that for M skew-matrix is invertible and

$$M_{i,j} = -M_{j,i} = M_{j,i}$$

and so M is also symmetric and invertible. □

Corollary 2.1.1. *Let n odd. There are no symmetric invertible $n \times n$ matrix with null diagonal and entries in $\mathbb{F}_q$ with characteristic 2.*

Proof.
Suppose exists M being a symmetric invertible $n \times n$ matrix in $\mathbb{F}_q^{n \times n}$.
From the previous observation, M is a invertible skew-matrix. Because n is odd and the Remark 2.1.1, there are no invertible □

Example 2.1.1. The reason to request the null-diagonal is mandatory to allow the correctness of the observations done.
Many books do not consider the cases where $\mathrm{Char}(\mathbb{F}_q) = 2$ and so they only consider

as a defining condition for a skew-matrix A that $A_{i,j} = -A_{j,i}$. But for characteristic equals 2, we have to add the null-diagonal condition to allow the general skew-matrices theorems and observation to be true.

For example, let $n = 3$ and $\mathbb{F}_q = \mathbb{F}_2$.
Consider the matrix

$$\pi = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{pmatrix}$$

It is trivial that π is a symmetric invertible matrix with not null-diagonal.
If we drop the null-diagonal condition, as a lot of books do, it is also skew symmetric.

We obtain some contradiction as π is a skew-matrix with n odd because, if we consider the proof of Remark 2.1.1, the determinant does not nullify with M because $M_{1,1} \neq 0$.

2.2 Boolean functions terminology

Boolean functions are a basic tool that permits to describe cryptography-primitives in a strong algebraic form.
In this subsection some basic definitions and notions will be reported.

Definition 2.2.1. Let $n \in \mathbb{N} \setminus \{0\}$.
A **Boolean function** (B.f.) is a function $f : \mathbb{F}^n \to \mathbb{F}$.
The set of all the Boolean function will be denoted by $\mathcal{B}_n$.

Each B.f. can be written in an unique way as a polynomial in $\mathbb{F}[X] = \mathbb{F}[x_1, \cdots, x_n]$ as

$$f(X) = \sum_{S \subseteq 1..n} a_S X_S \qquad X_S = \prod_{i \in S} x_i$$

This representation is called **Algebraic Normal Form** (ANF).

· The **algebraic degree** of a B.f. f coincides with the degree of its ANF

$$\deg(f) = \max\{\#S : a_S \neq 0\}$$

- Let $\mathcal{A}_n$ be the set of all affine functions $\mathbb{F}^n \to \mathbb{F}$ as the subset of the function in $\mathcal{B}_n$ with degree less than or equal to 1.
  The ANF of a function $\alpha \in \mathcal{A}_n$ is $\alpha = \sum_{i=1}^{n} \alpha_i x_i + \alpha_0$
- Let $\mathbb{F}^{2^n} = \{v_1, \ldots, v_{2^n}\}$. $\overline{f} = (f(v_1), \ldots, f(v_{2^n})) \in \mathbb{F}^{2^n}$ is called the **value vector** of f.
- The distance between two B.f. is the Hamming distance between their values vectors
$$d(f, g) = \#\{i | f(v_i) \neq g(v_i)\}$$
- The **non-linearity** of a B.f. f is the minimum distance between f and any affine function
$$N(f) = d(f, \mathcal{A}_n)$$
- **Covering Radius Bound**
$$N(f) \leq 2^{n-1} - \frac{1}{2} 2^{\frac{n}{2}}$$
- If a function f equals the covering radius bound, it is called **bent**.
- f is **balanced** if $\#f^{-1}(0) = \#f^{-1}(1) = 2^{n-1}$

Definition 2.2.2. Let $n, m \in \mathbb{N} \setminus \{0\}$.
A **vectorial Boolean function** (v.B.f.) is a function $F : \mathbb{F}^n \to \mathbb{F}^m$.

If we consider $v \in \mathbb{F}^n$, we call the v **component of** F as $x \to < v, f(x) >$ where $< \cdot, \cdot >$ is the standard scalar product.
Every component is a Boolean function and will denote the v component of F as F_v.

- F a v.B.f.
$$\deg F = \max_{v \in \mathbb{F}_2^n} \deg(F_v)$$
- The non-linearity of a v.B.f F is
$$N(f) = \min_{v \in \mathbb{F}^n} N(F_v)$$
- F v.B.f as $F : \mathbb{F}^n \to \mathbb{F}^m$ with $n, m \geq 1$.
  F is called **balanced** if for any $a, b \in \mathbb{F}^m$
$$\#F^{-1}(a) = \#F^{-1}(b) = 2^{n-m}$$

· A v.B.f. is balanced if and only if all the components are balanced.
A permutation is always balanced.

Definition 2.2.3. Let f be a v.B.f..

$$\hat{f}_u(x) := f(x+u) + f(x)$$

is called the **derivative** of f with respect to u.

We report some different measures of non-linearity.

· Let $m, n \geq 1$. $f : \mathbb{F}^n \to \mathbb{F}^m$. For any $a \in \mathbb{F}^n$ and $b \in \mathbb{F}^m$ we define

$$\delta_f(a,b) := \#\{x \in \mathbb{F}^n | \hat{f}_a(x) = b\}$$

The **differential uniformity** of f is

$$\delta(f) = \max_{a \in \mathbb{F}^n \setminus \{0\}, b \in \mathbb{F}^m} \delta_f(a,b)$$

f is said to be δ-differentially uniform if $\delta = \delta(f)$.

The smaller the δ, the highest the non-linearity of the function.

· If $\delta = 2$, then the function is called **Almost Perfect Non-linear** (APN)

· Let f a v.B.f.. f is **weakly δ-differentially uniform** if

$$\forall a \in \mathbb{F}^n \setminus \{0\}. \qquad \#\operatorname{Im}(\hat{f}_a) > \frac{2^{n-1}}{\delta}$$

If f is 2-weakly differential, f is called **weakly APN**.

· If f v.B.f. is δ-differentially uniform, then f is weakly δ-differentially uniform.

Definition 2.2.4. Let f v.B.f. such that $f(0) = 0$.
A function f is **l-anti-invariant** if for any subspace $U \subseteq \mathbb{F}_2^n$ such that $f(U) = U$ we have $\dim(U) < n - l$ or $U = \mathbb{F}_2^n$

Definition 2.2.5. Let f v.B.f. such that $f(0) = 0$.
A function f is **strongly l-anti-invariant** if for any two subspace $U, W \subseteq \mathbb{F}_2^n$ such that $f(U) = W$ we have $\dim(U) = \dim(W) < n - l$ or $U = W = \mathbb{F}_2^n$

Definition 2.2.6. Let f a v.B.f..

$$\hat{n}(f) = \max_{a \in \mathbb{F}_2^n \setminus \{0\}} \#\{v \in \mathbb{F}_2^n \setminus \{0\} | \deg(< v, \hat{f}_a >) = 0\}$$

It is possible to define different equivalence relations over v.B.f..

Definition 2.2.7. Two permutations $f, g : \mathbb{F}^n \to \mathbb{F}^n$ are **affine equivalent** if there exists $\gamma_1, \gamma_2 \in \mathrm{AGL}(V)$ such that $(x)g = (x)\gamma_2 f \gamma_1$.
Properties that are invariant under the action of the affine group are called **affine invariant**. For example, the following are affine invariant:

- Non-linearity
- Algebraic Degree
- Differential uniformity
- Weakly differential uniformity
- $\hat{n}(f)$

Definition 2.2.8. A v.B.f f is called **anti-crooked (AC)** if for each $a \in V \setminus \{0\}$ the set

$$\mathrm{Im}(\hat{f}_a) = \{f(x+a) + f(x) | x \in V\}$$

is not an affine subspace of V.

In [Cal15] are present a detailed description of properties that an AC function has with respect the non linearity.

Corollary 2.2.1 (3.1.7 [Cal15])**.**
Let f be a vBf. If $\hat{n}(f) = 0$ then f is AC.

Lemma 2.2.1 (3.1.10 [Cal15])**.**
If f is AC, then f^{-1} is not necessarily AC.

2.3 Cryptographic preliminaries

In this section, block ciphers will be defined along with the meaning of a *secure* block cipher.
References about Cryptography can be found in [Sti02], [RD82], [CW09].
It is usal to formally describe a cryptosystem as the abstract concept of *what* a system needs to have to be a cryptographic system.

Definition 2.3.1. A **cryptosystem** is a pair $(\mathcal{M}, \mathcal{K})$ where

- $\mathcal{M}$ is a finite set of possible messages; the union of $\mathcal{P}$ for plaintexts and $\mathcal{C}$ for ciphertexts.
- $\mathcal{K}$ is a finite set of possible keys
- for any $k \in \mathcal{K}$, we have an encryption and decryption function

$$\phi_k : \mathcal{P} \to \mathcal{C} \quad \psi_j : \mathcal{C} \to \mathcal{P} \quad \phi_k, \psi_j \in \mathrm{Sym}(\mathcal{M}) \quad \psi_k = \phi_j^{-1}$$

The general cryptosystem definition can be divided into two formal classes:

- **Symmetric key** in which $j = k$ and so $\phi_k = \psi_k^{-1}$.
  In this class the encryption and decryption has to be made with the same key.
- **Asymmetric key** in which $j \neq k$ and so $\phi_k \neq \psi_k^{-1}$.
  This class is meant to have a couple of keys working together: one encrypts and the other decrypts.

This division allows to develop different use of the cryptosystem.
For example, in an asymmetric key cryptosystem we can publish ϕ_k and allow everyone to crypt some information while only the owner of ψ_j can decrypt the message.

2.3.1 Block Ciphers

Block ciphers form an important class of cryptosystem in symmetric key cryptography.

Following the most used structure in modern ciphers,the plaintext space can be considered as coinciding with the ciphertext space.
Without lost of generality, it is possible to consider $\Omega = \mathcal{P} = \mathcal{C} = \mathbb{F}_q^r$ and $\mathcal{K} = \mathbb{F}_q^l$ with $l \geq r \geq 1$. Thus adapting the previous definitions as follows:

Definition 2.3.2. Let l, r be natural numbers. Let ϕ be any function as

$$\phi : \mathbb{F}_q^r \times \mathbb{F}_q^l \to \mathbb{F}_q^r$$

For any $k \in \mathbb{F}_q^l$, denote ϕ_k as the function

$$\phi_k : \mathbb{F}_q^r \to \mathbb{F}_q^r \qquad \phi_k(x) = \phi(x, k)$$

ϕ is an **algebraic block cipher** (or just block cipher) if ϕ_k is a permutation of $\mathbb{F}_q^r$ for all $k \in \mathbb{F}_q^l$.

By this definition:

Definition 2.3.3. A **block cipher** is a indexed set of permutation $\mathbb{F}_q^l \to \mathrm{Sym}(\mathbb{F}_q^r)$. Any key $k \in \mathcal{K}$ induces a permutation ϕ_k on $\mathcal{M}$.

It is possible to define a more general structure:

Definition 2.3.4. Let ϕ an algebraic block cipher. Let $\xi : \mathbb{F}_q^r \times \mathbb{F}_q^l \to \mathbb{F}_q^r$ a indexed set of permutation. Let $N \in \mathbb{N} \setminus 0$.
An **iterated block cipher** is a block cipher in which

$$\phi_k = \bigcirc_{i=1}^{N} \xi_{k_i} = \xi_{k_1} \cdots \xi_{k_N}$$

and k_i are obtained in an unique way from k.
It is also defined as a N-round iterated block cipher.

Most of the modern block ciphers are iterated ciphers: they are obtained by composing a finite number N of rounds.
In each round[1] the iterated ciphers perform a non-linear substitution operation, that we call S-Box, on disjoint parts of the input. This provide the *"confusion"* by Shannon terminology.

After that, the cipher does a permutation, usually a linear transformation, on the whole data that provide *"diffusion"*.

Defining the concept of *confusion and diffusion* is hard.
The main idea of *diffusion* consists in spreading the influence of a part of the input (the plaintext and key) to all the parts of the ciphertext.

[1]Except possibly for a couple which may be different.

The main idea of *confusion* consist in eliminating any clue between plaintext-key and the ciphertext.

This process is called **round** and the process performed in a round forms the **round function**.

For every $j \in 1..N$, the j-th round take as input the $j-1$-th round output and a sub-key $k^{(j)}$ derived from a **master key** k. These keys are generated by a **key schedule** that is a public algorithm, which strongly depends on the cipher, that construct $N+1$ sub-keys.

It is possible to formalize block ciphers in different ways. Let us consider the definition given in [CDS08] that can define a class large enough to include some common ciphers such as PRESENT ([BKL+07]), AES ([DR99],[DR02]), SERPENT ([BAK98]) but with enough algebraic structure to allow security proofs.

Let $V = \mathbb{F}^r$ with $r = mb$ and $b \geq 2$. V is a direct sum $V = V_1 \oplus \cdots \oplus V_b$ where each V_i has dimension m.
For any $v \in V$, we will write $v = v_1 \oplus \cdots \oplus v_b$ with $v_i \in V_i$ and we will consider the projection $\pi_i : V \to V_i$ mapping $v \to v_i$.

Any $\gamma \in \mathrm{Sym}(V)$ that acts as $x\gamma = x_1\gamma_1 \oplus \cdots \oplus x_b\gamma_b$ for some $\gamma_i \in \mathrm{Sym}(V_i)$, is a **bricklayer transformation** or **parallel map** and any γ_i is a **brick**.
Traditionally, γ_i is called S-Box and γ is a parallel S-Box.

A linear map $\lambda : V \to V$ is usually called Mixing-Layer.
Let $\sigma_v : x \to x + v$ the translation by v.
For any $I \subset 1..b$ with $I \neq \emptyset$ or $1..b$; we define $\oplus_{i \in I} V_i$ a **wall**.

Definition 2.3.5. A linear map $\lambda \in \mathrm{GL}(V)$ is a **proper mixing layer** if no wall is invariant under λ.

Now it is possible to characterize the **translation-based** class by the following definition.

Definition 2.3.6. A block cipher $\mathcal{C} = \{\phi_k | k \in \mathcal{K}\}$ over $\mathbb{F}_2$ is called **translation based (tb)** if:

- it is the composition of a finite number of rounds, such that any round $\rho_{k,h}$ can be written as $\gamma\lambda\sigma_u$ where
  - γ is a round-dependent bricklayer trasformation which does not depend on k
  - λ is a round-dependent linear map which does not depend on k
  - u is in V and depends on both k and the round. u is called **round key**
- For at least one round, at the same time
  - λ is proper
  - the map $\mathcal{K} \to V$ given by $k \to u$ is surjective

  It is called a **proper round**.

It is now important to define *"when a block cipher is secure"*. Here are several criteria that contribute to the evaluation of a cipher:

- **Security**
  The security of a block cipher is highly dependent on the properties of the different components such as the substitution layer and the linear (or affine) transformation.
  However, there is no mathematical method to prove the security of a given block cipher, although it is sometimes possible to prove the insecurity of such a cipher.
  For this reason, to evaluate security it is often considered the **practical security**. According to this concept, a cipher is secure if the best-known attack requires too many resources with respect a suitable and acceptable margin. Testing the cipher with known attack(s) and pondering the different outcomes will result in a global security index.
  It is impossible to predict the security of a cipher with respect to yet unknown attacks.
- **Efficency**
  It refers to the amount of computation and memory used to perform ϕ or ψ.
  In fact, the goal is having really optimized computation that require the minimum amount of memory and time and are easily implementable if hardware or software.

- **Flexibility**
  Flexibility is the capacity of the block cipher to be used in different context maintaining its security.
  It is important since a block cipher can be used as a building block in various cryptographic constructions like a hash function, an authentication code or a stream cipher. Or it can be necessary to expand or reduce the plaintext/key space for practical reasons.

In a cryptoanalytic scenario, it is vital to define all the possible attacks that a block cipher is able to withstand..
We can subdivide the attacks by *modes* or *types* of attack:

- by **modes** of attack, we define the strategy of a possible attacker. Some of these modes, ordered from the most practical to the most hypothetical, are
  - *Ciphertext-only*: the attacker tries to deduce information about the key (or plaintext) starting from the knowledge of several ciphertext and, usually, assuming some properties about the distribution of the plaintext. This is a unlikely scenario for modern block cipher.
  - *Known-plaintext*: the attacker knows a certain set of couple plain-ciphertext. In this scenario, the attacker can search for redundancy in the pairs or it can analyse the distribution of the cipher. Linear cryptoanalysis is a typical example of such an attack (see [Mat94]).
  - *Chosen-plaintext or chosen-ciphertext*: the attacker has the ability to construct the set of known-pair. The aim is to exploit the block cipher and obtain information from the chosen-pairs. Differential cryptoanalysis is a typical example (see [AC09]).
  - *Adaptive chosen-plaintext or ciphertext*: the attacker has the ability to chose new plaintext (or ciphertext) to cipher depending on the information gained during the attack.
  - *Combined chosen-plaintext and chosen-ciphertext*: is ad adaptive attack where the attacker can encrypt or decrypt arbitrary messages as he desire.
  - *Related-key*: the attacker knows (or can impose) additionally mathematical relations between the keys used for encryption, but not their values. It can be practical in a scenario where the block cipher is used as a building brick, such in hash functions.
- by **types** of attack, we describe the output or reason of the attack. Graded from the least favourable to the most, they are:

- *Distinguishing attack*: the attacker is able to tell whether the attacked block cipher is a random permutation or it is a permutation described by a ϕ_k.
  Modern block cipher are designed to model a random permutation. If a block cipher is weak to a distinguishing attack, it may be present a structural flaws of the cipher that might be transformed into a more powerful attack.
- *Local deduction*: the attacker is able to find the plaintext of an intercepted ciphertext which he did not obtain from the legitimate sender. In other words, the attacker knows a set of *likely plaintext*: if this set is small, the block cipher can be broken.
- *Partial Key Recovery*: the attacker can get some information abount the key k like some bits or the relations and structure of the key.
- *Global deduction*: the attacker is able to construct ϕ_k, ψ_k without the knowledge of the key k.
- *Key recovery*: the attacker is able to recover the master key k. The cipher is totally broken.

Some additional parameters are basilar to compare the effectiveness of the attack:

- *Time complexity*: it measure the computational processing required to perform an attack. Usually, the choice of the computational unit is done to compare the attack with an exhaustive key search.
- *Data complexity*: it is the number of collected data (ciphertexts, known or chosen plaintexts) required to perform an attack according to the model.
- *Success probability*: it measures the frequency at which the attack is successful when repeated a certain number of times in a statistically independent way
- *Memory complexity*: it measures the amount of memory units necessary to store pre-computed/obtained data necessary to perform the attack

An attack is considered successful if the time/data/memory complexity is significantly smaller than 2^l with $\#\mathcal{K} = \mathbb{F}^l$ and a success probability close to 1.

2.3.2 Trapdoors and group theoretic properties

In this subsection we will describe *"what is a trapdoor"* and then define the algebraic properties that exist in the block cipher model used.
Other references can be found in [RP97], [MPW94], [Pat99].

We can define a **trapdoor** for a block cipher is a hidden structure that, with the knowledge of this structure, allows an attacker to obtain information on the key or to decrypt certain ciphertexts.
Trapdoors are usually dived into:

- **Full trapdoor** is some secret information that allows an attacker to acquire the key, or a global deduction of it, by using a small number of known plaintexts, no matter which they are or what the key is.
- **Partial trapdoor** does not necessarily work with all the keys or plaintexts.
- **Detectable (or undetectable) trapdoor** define the trapdoor that are computationally feasible (or infeasible) to be found even if one knows the general form of the trapdoor.

The hidden sum trapdoor is directly constructed over some group properties that now we will describe and explain.

Let $\mathcal{C} = \{\phi_k | k \in \mathcal{K}\}$ be a tb cipher with plaintext space $V = \mathbb{F}^d$ for some $d \in \mathbb{N}$. The goal is to determine the group $\Gamma(\mathcal{C}) =< \phi_k | k \in \mathcal{K} > \subseteq \mathrm{Sym}(V)$ generated by the permutation ϕ_k.

Unfortunately, for many classical cases such AES, SERPENT, DES ([Des77]), this appears to be a difficult problem.

For this reason, we define for each h, the round function

$$\Gamma_h(\mathcal{C}) = \langle \phi_{k,h} | k \in \mathcal{K} \rangle \subseteq \mathrm{Sym}(V) \qquad \phi_{k,h} = \gamma_h \lambda_h \sigma_{k,h}$$

and

$$\Gamma_\infty(\mathcal{C}) = \langle \Gamma_h(\mathcal{C}) | h \in 1..l \rangle$$

Remark 2.3.1.

$$\Gamma_\infty(\mathcal{C}) = \langle \gamma_h \lambda_h \sigma_{k,h} | h \in 1..l, k \in \mathcal{K} \rangle$$

As we can observe, $\langle \sigma_{k,h} \rangle = T_+$ and so we have $T_+ \subseteq \Gamma_\infty(\mathcal{C})$.

It is decisive to change the goal stated before, to determine $\Gamma_\infty(\mathcal{C})$.
That is the permutation group generated by its round functions with the key varying in the key space.
A weakness of that group might reveal weaknesses of the cipher.

Paterson, in [Pat99], showed that if this group is imprimitive, then it is possible to embed a trapdoor in the cipher.
He developed a DES-like cipher with such a trapdoor.
The attack that can be constructed is a chosen-plaintext.

Paterson's attack needs that the round ξ of an n-round iterated block cipher ϕ acts imprimitively on the space message $\mathcal{M}$ (seen as a vector space) and let $Y_1, \ldots, Y_r$ a not-trivial block system for ξ.
Suppose that it is easy, given a message $m \in \mathcal{M}$, to compute the i-th block in which $m \in Y_i$.
Let $k_1, \ldots, k_t$ the key used in the cipher on the different rounds.

Suppose we choose a set m_i such that $m_i \in Y_i$ for every i and obtain the ciphertext c_i.
Now, from the imprimitivity of ϕ we have

$$c_i = m_i\phi \in Y_j \qquad Y_j = Y_i\phi$$

Now, given any other ciphertext c, we compute the l in which $c \in Y_l$. We have that $m \in Y_l\phi^{-1}$.
So r chosen plaintexts determine that the message corresponded to any ciphertext must lie in a set of dimension $\frac{\#\mathcal{M}}{r}$.

For a translation based cipher, in [CDS08] the authors provided conditions on the S-boxes which ensure that the group Γ_∞ is primitive using boolean functions properties.

Theorem 2.3.1 ([CDS08])**.**
Let $\mathcal{C}$ be a tb cipher, with h a proper round and $r \in 1..\frac{m}{2}$ and $r < \frac{m}{2}$.
If any brick of γ_h is

- *weakly $2r$-uniform*
- *strongly r-anti-invariant*

then $\Gamma_h(\mathcal{C})$ *is primitive (and hence* $\Gamma_\infty(\mathcal{C})$ *is primitive).*

In [CDVS09], some additional condition on S-boxes of a tb cipher are established such that $\Gamma_\infty(\mathcal{C})$ is either $\mathrm{Alt}(V)$ or $\mathrm{Sym}(V)$ obtaining the following theorem.

Theorem 2.3.2 ([CDVS09])**.**
Let $d = mn$ *with* $m, n > 1$*. Let* $\mathcal{C}$ *be a tb cipher such that*

- $\mathcal{C}$ *satisfies the hypothesis of the theorem in [CDS08]*
- γ *is Anti-Crooked, so for all non-zero* $a \in V_i$, $\mathrm{Im}(\hat{\gamma}_{i_a})$ *is not a coset of a subspace of* V_i

Then the group $\Gamma_\infty(\mathcal{C})$ *is either* $\mathrm{Alt}(V)$ *or* $\mathrm{Sym}(V)$*.*

In particular, we are interested on creating an algebraic attack that will be constructed in the next sections.
The theorem gives a strong characterisation that ensure us that the cipher is Alt or Sym.

Corollary 2.3.1. *Let* $\mathcal{C}$ *be a* N*-round translation based cipher with* γ_h *permutation box of the round* h*.*
If the round h *has* γ_h *AC, then* $\Gamma_\infty(\mathcal{C}) \nsubseteq \mathrm{AGL}_\circ$ *for any operation* $\circ$*.*

2.4 Hidden Sum

The concept behind the research of hidden sum trapdoor is connected by the first point of the Li theorem ([Li03]), which is a particular case of the O'Nan-Scott theorem.

Theorem 2.4.1 ([Li03])**.** *Let* G *be a primitive group of degree* p^b *with* $b > 1$*. Suppose* G *contains a regular abelian subgroup* T*. Then* G *is one of the following.*

- *Affine.* $G \subseteq \mathrm{AGL}(e, p)$ *for some prime* p *and* $e \geq 1$
- *Wreath product*
- *Almost simple*

We are interested in the affine case because Li observed that if V is a vector space over a finite field $\mathbb{F}_p$, $\mathrm{Sym}(V)$ contains many isomorphic copies of $\mathrm{AGL}(V)$ which are conjugates in $\mathrm{Sym}(V)$. A new abelian additive group $(V, \circ)$ will yield in general a different copy $\mathrm{AGL}(V, \circ)$ of the affine group.
This new structure $(V, \circ)$ will be our **hidden sum**.

Note that if h is a proper round of a tb cipher $\mathcal{C}$, then $\Gamma_h(\mathcal{C}) =< \gamma_h\lambda_h, T(V) >$ where $T(V)$ is the translation group.
Thus, it could be that Γ_∞ is contained in a isomorphic copy of $\mathrm{AGL}(V)$ and so happens that the abelian additive group $(V, \circ)$ is a **hidden sum trapdoor**.

2.4.1 Radical ring

For abelian regular subgroups of the affine group, in [CDS05] the authors give an easy description of these in terms of commutative associative algebra that one can impose on the vector space $(V, +)$.

Definition 2.4.1. A Jacobson radical ring is a ring $(V, +, \cdot)$ in which every element is invertible with respect to the circle operation

$$x \circ y = x + y + x \cdot y$$

so that $(V, \circ)$ is a group.
The operation $\circ$ may induce, or not, a vector space structure on V.

Theorem 2.4.2 ([CDS05])**.** *Let $\mathbb{K}$ be any field and $(V, +)$ a vector space of any dimension over $\mathbb{K}$.*
Then there is a one-to-one correspondence between

1. *(not necessarily elementary) abelian regular subgroups T of* $\mathrm{AGL}(V, +)$
2. *commutative, associative $\mathbb{K}$-algebra structure $(V, +, \cdot)$ that one can impose on the vector space structure $(V, +)$ such that the resulting ring is radical*

In this correspondence, isomorphism classes of $\mathbb{K}$-algebras corresponds to conjugacy classes of abelian-subgroups of $\mathrm{AGL}(V, +)$ *where the conjugation is under the action of* $\mathrm{GL}(V, +)$.

We will denote with T_+ the translation group and a $\sigma_a \in T_+$ acts as $x\sigma_a = x + a$.

We have a relation between $T_\circ$ and $\mathrm{AGL}(V,\circ)$ as that $\mathrm{AGL}(V,\circ)$ is the normalizer of $T_\circ$ with respect to $\mathrm{Sym}(V)$.
Indeed, we have $\mathrm{AGL}(V,+)$ to be the normalizer of T_+ and $\mathrm{AGL}(V,+)$ and T_+ are the isomorphic images of $\mathrm{AGL}(V,\circ)$ and $T_\circ$ respectively.
With 1_V we will denote the identity map of V. We clearly have that $1_V \in \mathrm{AGL}(V,\circ)$ for any operation $\circ$.

As the affine group is the semi-direct product $\mathrm{AGL}(V,+) = \mathrm{GL}(V,+) \ltimes T_+$, every $\tau_v \in \mathrm{AGL}_+(V)$ can be written as $\tau_v = k_v\sigma_v$ with $k \in \mathrm{GL}(V,+)$ and $\sigma_v \in T_+$.
We can define $\Omega(T_\circ) = \{k_a | a \in V\} \subseteq \mathrm{GL}(V,+)$ and the set

$$U(T) = \{a | \tau_a = \sigma_a\}$$

We have that $U(T)$ is a subspace of V and if $T = T_\circ$ for some operation $\circ$, then $U(T_\circ)$ is not empty by the lemma:

Lemma 2.4.3 ([CDS05]). *Let $T \subseteq \mathrm{AGL}_+(V)$ be a regular subgroup. Then if V is finite, $T_+ \cap T$ is non-trivial.*

and follows

Proposition 2.4.4 (2.1.6 in [Cal15]). *Let $T \subseteq \mathrm{AGL}(V,+)$ be an elementary abelian regular subgroup. If $T \neq T_+$, then $\dim(U(T)) \in 1..(n-2)$.*

The structure of $T_\circ$ that we need to find to create the hidden sums, has to respect

Lemma 2.4.5 (2.1.11 in [Cal15]). *Let $V = \mathbb{F}^n$ and $T \subseteq \mathrm{AGL}(V,+)$ be an elementary abelian regular subgroup. Then for each $a \in V$, k_a has order 2 and it is unipotent. In particular $\Omega(T)$ is a unipotent subgroup of $\mathrm{GL}(V,+)$.*

So we have that $k_a^2 = 1_V$.

Lemma 2.4.6 (2.1.13 in [Cal15]). *Let $V = \mathbb{K}^n$, $\mathbb{K}$ any field. Let $G \subseteq \mathrm{GL}(V)$ be a unipotent subgroup and let $W \subseteq V$ be a subspace such that for all $v \in W$ and $g \in G$, $vg = v$; so G is contained in the stabilizer of W.*
Then all elements of G are upper triangular in a basis $\{v_1, \ldots, v_{n-k+1}, \ldots, v_n\}$ where $\{v_{n-k+1}, \ldots, v_n\}$ is any basis of W.

So we have that k_a is upper triangular for every $a \in V$.

Corollary 2.4.1. *If we have T such that $\Omega(T)$ is a unipotent group, then all elements of $\Omega(T)$ are upper triangular in a basis $\{v_1, \ldots, v_{n-k+1}, \ldots, v_n\}$ where $\{v_{n-k+1}, \ldots, v_n\}$ is any basis of $U(T)$.*

Remark 2.4.1 (Cor. 2.1.16 [Cal15]). We can conjugate T and obtain a subgroup T^* such that
$\Omega(T^*) \subseteq \mathcal{U}(V)$ and $U(T^*) = \mathrm{Span}\{e_{n-k+1}, \ldots, e_n\}$ where $k = \dim(U(T^*))$.

So we have that exists an element in the group that conjugate the group T such that k_a's are upper triangular and has the subspace $U(T^*)$ generated by the last k vector of the canonical basis.

Theorem 2.4.7 (2.1.21 [Cal15]). *Let $V = \mathbb{F}_p^{n+k}$ with $n \geq 2, k \geq 1$. $T_\circ \subseteq \mathrm{AGL}(V, +)$ be such that $U(T_\circ) = \mathrm{Span}\{e_{n+1}, \ldots, e_{n+k}\}$.*
Then $T_+ \subseteq \mathrm{AGL}(V, \circ)$ if and only if for all $k_y \in \Omega(T_\circ)$ there exists a matrix $B_y \in \mathbb{F}_p^{n \times k}$ such that

$$k_y = \begin{pmatrix} I_n & B_y \\ 0 & I_k \end{pmatrix}$$

This theorem describes the form of the k_y and imposes a condition to guarantee $T_+ \subseteq \mathrm{AGL}(V, \circ)$.

Note 2.4.1.
Imposing the form of Theorem 2.4.7 for the B_{e_i} we have that the canonical basis **is** a basis even in the operation $\circ$ and that we have that $T_\circ \subseteq \mathrm{AGL}_+$ **and** $T_+ \subseteq \mathrm{AGL}_\circ$.

2.4.2 Hidden Sum Attack

To embed a hidden sum trapdoor in a n-bit block cipher, we need $\Gamma_\infty \subseteq \mathrm{AGL}(V, \circ)$ for some hidden sum $\circ$.
As $T_+ \subseteq \Gamma_\infty$, the first condition we need to have is $T_+ \subseteq \mathrm{AGL}(V, \circ)$. Now consider $T_\circ \subseteq \mathrm{AGL}(V, +)$ such that $T_+ \subseteq \mathrm{AGL}(V, \circ)$.

Consider $\dim(U(T_\circ)) = k \geq 1$ and let $g \in \mathrm{GL}(V, +)$ such that

$$U(T_\circ)g = \mathrm{Span}(\{e_{n-k+1}, \ldots, e_n\}) = U(T_\triangle)$$

where $T_\triangle = g^{-1}T_\circ g$.
g is an isomorphism between $(V, \circ)$ and $(V, \triangle)$.

Now, from the condition on the basis, we have that $\{e_i\}_{i\in 1..n}$ is basis of $(V, \triangle)$ and we can write $v \in V$ as a linear combination with respect to the sum $\triangle$

$$v = \triangle_{i=1}^{n} \lambda_i e_i$$

We can find the λ_i using the following algorithm described in [Cal15]:

Algorithm 2.4.1. *Let as input $v = (v_1, \cdots, v_n) \in V$.*
Let $\lambda_i = v_i$ for every $i \in 1..(n-k)$. Now consider $v\prime = v\tau_{e_1}^{\lambda_1} \dots \tau_{e_{n-k-1}}^{\lambda_{e_{n-k-1}}}$ and let

$$\lambda_i = v_i' \quad i \in (n-k+1)..n$$

Return $[v] = [\lambda_i]$.

Let consider $[v] = [\lambda_1, \dots, \lambda_n]$.
Let $\mathcal{C} = \{\phi_k | k \in \mathcal{K}\}$ be a tb cipher such that $\Gamma_\infty \subseteq \mathrm{AGL}(V, \circ)$ for some operation $\circ$ and $T_\circ \subseteq (V, +)$. Let $\dim(U(T_\circ)) = k$. Let g as before.

Let ϕ be the encryption function with a given unknown session key k.

Mount the attack: the hidden sum attack is a global deduction, chosen plaintext.

Chose the plaintext $0\phi, v_1\phi, \dots, v_n\phi$ where $v_i = e_i g^{-1}$.
Compute $[0\phi g], [v_1\phi g], \dots, [v_n\phi g]$.

Since $[0\phi g] = [t]$ is the translation vector and $[e_i\phi g] + [t]$ are the matrix rows, we have

$$[v\phi g] = [vg] \cdot M + [t] \qquad [v\phi^{-1} g] = ([vg] + [t])M^{-1}$$

for all $v \in V$.

We are able to crypt and decrypt because we reconstructed the function ϕ as an affine transformation.

Note that we only need $n+1$ plaintexts to reconstruct the cipher from $v = 0\tau_{e_1}^{\lambda_1} \cdots \tau_{e_n}^{\lambda_n}$ and the computational cost of executing the attack is extremely low.

For this reason, we can claim

Theorem 2.4.8 (2.4.1 in [Cal15]). *Hidden sum trapdoors coming from translation groups such that $T_\circ \subseteq \mathrm{AGL}(V, +)$ are practical full trapdoors.*

Proof.
To prove the theorem, we can compare the computational complexity of the attack with respect a brute force attack where we initially have a ciphertext and the block cipher as a black box that just encrypts the input given.

Let assume that the input of the cipher is a n bit message block.

We assume that the encryption has a linear cost with respect the input, so $\mathcal{O}(n)$.[2]. We consider to have constant cost $\mathcal{O}(1)$[3]:

- Check the equality of element in the message space
- Generating matrices
- Linear algebra, i.e. multiplying matrices, sum vectors, inversion of a matrix.
- Generating the $[\lambda_i]$, as the fact that it is just the composition of affine transformation

Trivially, the brute force attack has to check and encrypt all the message space and so it has a total cost of $\mathcal{O}(n2^n)$.

Now consider the hidden sum attack:

· Find $[x]$ for the $n+1$ plaintext. Cost $\mathcal{O}(n+1)$.

· Encrypt $n+1$ plaintext in the form $[\lambda]$. Cost $\mathcal{O}((n+1) \cdot n)$

· Generate the affine transformation and invert the matrix. Cost $\mathcal{O}(1)$.

· Multiply the ciphertext and recover the plaintext. Cost $\mathcal{O}(1)$.

[2]This assumption is consistent with reality. A block cipher on a bigger message space has to encrypt *"slower"* than the same block cipher with less round or smaller message space.

[3]All this algorithm have a polynomial complexity $\mathcal{O}(n^k)$ for some $k \in \mathbb{N}$

So we obtain an approximate total cost of

$$\mathcal{O}((n+1)+(n+1)n+1+1) \sim \mathcal{O}(n^2)$$

which is negligible with respect the brute force attack. □

Remark 2.4.2.
To use the hidden sum attack, we need to know which is the hidden sum $\circ$.

From the Corollary 2.3.1, if we have a round with all the γ_i permutation boxes to be AC, we have that does not exists any hidden sum $\circ$ such that Γ_∞ is contained in $\mathrm{AGL}_\circ$ and so we cannot use the hidden sum attack.

Remark 2.4.3. For a real-case analysis:
AES cipher has primitive group Γ_∞ (see [SW08]) and its γ is Anti Crooked. So by Theorem 2.3.2 we have that AES is either Alt or Sym and so is unaffected by the hidden sum attack.

SERPENT cipher has some round with permutation box that is AC and other not. Even so, it is either Alt or Sym by Corollary 2.3.1. Wernsdorf, in the NIST's AES Competition, proved that SERPENT is isomorphic to the alternate group (see [Wer00]).

It was proven that a round of DES generate the alternating group (see [EG83]).

PRESENT has a not AC permutation box and so the group Γ_∞ of PRESENT is primitive, but we cannot say that PRESENT has a hidden sum. By Li's Theorem 2.4.1, it can be $\mathrm{AGL}_\circ$ or one of the other choice of the theorem.

To prove that PRESENT can be attacked by the hidden sum attack, it is necessary to analyse the structure of Γ_∞ to see if it is $\mathrm{AGL}_\circ$ for some operation $\circ$.

Hidden sum cardinality

In this chapter we will focus on the question:
"how many operation $\circ$ *with dimension* $n+k$ *have exactly* $\dim U(T_\circ) = k$ *?"*.
We will start presenting the bound described in [Cal15] that uses the cardinality of a variety constructed with some properties that the operation has to respect to then rewrite the problem in finding the cardinality of particular sets of vectors in $\mathbb{F}_2^{kn}$.
We will give a lower bound and compare it with the Calderini's bound.

3.1 Variety bound

We start from the [Cal15]'s Lemma 2.1.26.

Let $n > 2$ and $0 < k < n-2$. For $i \in 1..n$, let k_i the matrix of τ_{e_i} of a hidden sum $\circ$, in the form of the Theorem 2.4.7, over $V = \mathbb{F}^{n+k}$. Let $k_{e_i} = k_i$ as

$$k_i = \begin{pmatrix} & b_{1,1}^{(i)} & \cdots & b_{1,k}^{(i)} \\ I_{n\times n} & \vdots & \ddots & \vdots \\ & b_{n,1}^{(i)} & \cdots & b_{n,k}^{(i)} \\ & & I_k & \end{pmatrix}$$

Lemma 3.1.1. *Let* $N := n+k$ *and* $V := \mathbb{F}^N$*, with* $n \geq 2$ *and* $k \geq 1$.
The elementary abelian regular subgroups $T_\circ \subseteq \mathrm{AGL}(V,+)$ *such that* $\dim(U(T)) = k$ *and* $T_+ \subseteq \mathrm{AGL}(V,\circ)$ *are*

$$\begin{bmatrix} N \\ k \end{bmatrix}_2 \cdot |\mathcal{V}(\mathcal{I}_k)|$$

where $\mathcal{I}_k$ *is generated by*

$$S_1 \cup S_2 \cup S_3$$

with

$$S_1 := \left\{ \prod_{i=1}^{n} \prod_{j=1}^{k} \left(1 + \sum_{s\in S} b_{i,j}^{(s)}\right) | S \subseteq [n], S \neq \emptyset \right\}$$

$$S_2 := \{ b_{i,j}^{(s)} - b_{s,j}^{(i)} | i, s \in [n], j \in [k] \}$$

$$S_3 := \{b_{i,j}^{(i)} | i \in [n], j \in [k]\}$$

$\mathcal{V}(\mathcal{I}_k)$ is the variety over $\mathbb{F}$ of $\mathcal{I}_k$ and $\begin{bmatrix} N \\ k \end{bmatrix}_q = \prod_{i=0}^{k-1} \frac{q^{N-i}-1}{q^{k-i}-1}$ is the Gaussian Binomial.

The three sets define the properties of the operation $\circ$ which are related to the group $T_\circ$:

1. S_1: $T_\circ$ has to have $U(T_\circ)$ dimension equals to k.
   In the form we are considering, we have that $e_{n+1}, \ldots, e_{n+k}$ are the elements that generate the subspace $U(T_\circ)$ and so we need to have that for any $v \in \langle e_1, \ldots, e_n \rangle$, $B_v \neq 0$. Otherwise we have that $v \in U(T_\circ)$ and so $\dim(U(T_\circ)) = k + 1$.

   To achieve this goal, we need to have that, for any $v \in \langle e_1, \ldots, e_n \rangle$, $B_v \neq 0$.
   To obtain B_v, consider $\tau_v = \tau_{e_1}^{\lambda_1} \cdots \tau_{e_n}^{\lambda_n}$ for $\lambda_i \in \mathbb{F}_2$. We obtain:

$$\begin{aligned}
\tau_v = \begin{pmatrix} I_n & B_v \\ & I_k \end{pmatrix} \sigma_v &= \tau_{e_1}^{\lambda_1} \cdots \tau_{e_n}^{\lambda_n} \\
&= (k_{e_1}\sigma_{e_1})^{\lambda_1} \cdots (k_{e_n}\sigma_{e_n})^{\lambda_n} \\
&= (k_{e_1}^{\lambda_1}\sigma_{e_1}^{\lambda_1}) \cdots (k_{e_n}^{\lambda_n}\sigma_{e_n}^{\lambda_n}) \\
&= \left(\prod_{i=1}^{n} k_{e_i}^{\lambda_i}\right) \left((\sigma_{e_1}^{\lambda_1}(k_{e_2}^{\lambda_2} \cdots k_{e_n}^{\lambda_n})) \cdots (\sigma_{e_{n-1}}^{\lambda_{n-1}}(k_{e_n}^{\lambda_n}))\sigma_{e_n}^{\lambda_n}\right) \\
&= \left(\prod_{i=1}^{n} k_{e_i}^{\lambda_i}\right) \left(\prod_{i=1}^{n} \left(\sigma_{e_i}^{\lambda_i} \prod_{j=i}^{n} k_{e_j}^{\lambda_n}\right)\right) \\
&= \left(\prod_{i=1}^{n} k_{e_i}^{\lambda_i}\right) \sigma_v \\
&= \left(\prod_{i=1}^{n} \begin{pmatrix} I_n & \lambda_i B_{e_i} \\ & I_k \end{pmatrix}\right) \sigma_v \\
&= \begin{pmatrix} I_n & \sum_{i=1}^{n} \lambda_i B_{e_i} \\ & I_k \end{pmatrix} \sigma_v
\end{aligned}$$

   and so we get that $B_v = \sum_{i=1}^{n} \lambda_i B_{e_i}$.
   We only need to check that every not null linear combination, of the B_{e_i}'s, is not null.

2. S_2: $T_\circ$ is abelian. This means that for $i, j \in 1..n$, we have $e_i\tau_{e_j} = e_j\tau_{e_i}$ and that means

$$e_i k_{e_j} + e_j = e_i \left(I_{n+k} + \begin{pmatrix} 0 & B_{e_j} \\ 0 & 0 \end{pmatrix}\right) + e_j = e_i + e_j + e_i \begin{pmatrix} 0 & B_{e_j} \\ 0 & 0 \end{pmatrix}$$

$$e_j k_{e_i} + e_i = e_j \left(I_{n+k} + \begin{pmatrix} 0 & B_{e_i} \\ 0 & 0 \end{pmatrix} \right) + e_i = e_i + e_j + e_j \begin{pmatrix} 0 & B_{e_i} \\ 0 & 0 \end{pmatrix}$$

and so the j-th row of B_{e_i} has to be equal to the i-th row of B_{e_j}

3. S_3: $T_\circ$ has to be elementary and so for every $i \in 1..n$ we need to have $e_i \tau_{e_i} = 0$. Similarly to the previous point, we get

$$0 = e_i k_{e_i} + e_i = e_i \left(I_{n+k} + \begin{pmatrix} 0 & B_{e_i} \\ 0 & 0 \end{pmatrix} \right) + e_i = e_i + e_i + e_i \begin{pmatrix} 0 & B_{e_i} \\ 0 & 0 \end{pmatrix} = e_i \begin{pmatrix} 0 & B_{e_i} \\ 0 & 0 \end{pmatrix}$$

and so the i-th row of B_{e_i} has to be null.

At this point, a bound is given in Proposition 2.1.27:

Proposition 3.1.2. *Let $\mathcal{I}_k$ defined as Lemma. Then*

$$|\mathcal{V}(\mathcal{I}_k)| \leq \left(2^{k\binom{n}{2}} - 1 - \sum_{r=1}^{n-2} \binom{n}{r} \prod_{i=1}^{\binom{n-r}{2}} \left(2^k - 1\right) \right) =: \mu(n,k)$$

To improve the bound, we search for a simpler construction that facilitate the count of the variety.

Remark 3.1.1.
We will always drop the $\begin{bmatrix} N \\ k \end{bmatrix}_2$ coefficient as we will always refer to the operations $\circ$ that has the form described in the preliminaries (see Theorem 2.4.7).
For this reason, we can concentrate only on the B_v part of the k_v matrices.

3.2 Thoughts on the matrix construction

Let consider $\circ$ in the form of the Theorem 2.4.7.
With this form, for any $v \in V$, $\tau_v = k_v \sigma_v$ where $k_v = \begin{pmatrix} I_n & B_v \\ & I_k \end{pmatrix}$.
In particular for that form, we have that the canonical basis $\{e_i\}_{i \in 1..(n+k)}$ is a basis for $T_\circ$ and so we can only considerate the matrices B_{e_i}'s for every $i \in 1..(n+k)$.
In addiction, we have $U(T_\circ) = \mathrm{Span}(\{e_{n+1}, \ldots, e_{n+k}\})$ and so

$$B_{e_{n+1}} = \cdots = B_{e_{n+k}} = 0$$

So we only need to concentrate on B_{e_i}'s with $i \in 1..n$.

Let $\{B_{e_i}\}_{i\in 1..n}$ the submatrices of the canonical basis $\{e_i\}_{i\in 1..n}$ from which it is possible to create the k_{e_i} and so it is possible to generate $T_\circ$.
We have that, for every i, B_{e_i} is a matrix in $\mathbb{F}_2^{n\times k}$. We can see the rows of the matrix as element in $\mathbb{F}_{2^k}$.
In this way, we can see every B_{e_i} as a vector in $\mathbb{F}_{2^k}^n$.

We can now consider the space generated by all the vectors B_{e_i}, meant as column vectors, in a single matrix $B_\circ$ in $\mathbb{F}_{2^k}^{n\times n}$.

$$\text{From } \{B_{e_i}\}_{i\in 1..n} \longrightarrow B_\circ := \begin{pmatrix} B_{e_1} & \cdots & B_{e_n} \end{pmatrix}$$

The matrix $B_\circ$ has all the information about the operation $\circ$.
We can even see this matrix as a matrix in $\mathbb{F}^{(kn)\times n}$ as

$$B_\circ = \begin{pmatrix} B_{e_1} & \cdots & B_{e_n} \end{pmatrix} = \begin{pmatrix} B_{e_1}^T[1] & \cdots & B_{e_n}^T[1] \\ B_{e_1}^T[2] & \cdots & B_{e_n}^T[2] \\ \vdots & \vdots & \vdots \\ B_{e_1}^T[n] & \cdots & B_{e_n}^T[n] \end{pmatrix}$$

where $B_{e_i}^T[j]$ is the transposition of the j-th row of B_{e_i}.

Example 3.2.1. Let $N = 5$ and $k = 2$ and consider the operation $\circ$ defined by these B_{e_i}'s:

$$\begin{cases} B_{e_1} = \begin{pmatrix} 0 & 0 \\ 1 & 1 \\ 1 & 1 \end{pmatrix} & B_{e_2} = \begin{pmatrix} 1 & 1 \\ 0 & 0 \\ 0 & 1 \end{pmatrix} \\ B_{e_3} = \begin{pmatrix} 1 & 1 \\ 0 & 1 \\ 0 & 0 \end{pmatrix} \\ B_{e_4} = \begin{pmatrix} 0 & 0 \\ 0 & 0 \\ 0 & 0 \end{pmatrix} & B_{e_5} = \begin{pmatrix} 0 & 0 \\ 0 & 0 \\ 0 & 0 \end{pmatrix} \end{cases}$$

Because we have that $n = N - k = 2$, we just need to focus on B_{e_1}, B_{e_2} and B_{e_3}.

If we see the elements of the matrices as elements of $\mathbb{F}_{2^2} = \{0, 1, \alpha, \alpha+1\}$ as the field extension where $\alpha^2 = \alpha + 1$, we can rewrite the matrices as

$$\begin{cases} B_{e_1} = \begin{pmatrix} 0 \\ \alpha+1 \\ \alpha+1 \end{pmatrix} & B_{e_2} = \begin{pmatrix} \alpha+1 \\ 0 \\ \alpha \end{pmatrix} \\ B_{e_3} = \begin{pmatrix} \alpha+1 \\ \alpha \\ 0 \end{pmatrix} \end{cases}$$

We have that

$$B_\circ = \begin{pmatrix} B_{e_1} & B_{e_2} & B_{e_3} \end{pmatrix} = \begin{pmatrix} 0 & \alpha+1 & \alpha+1 \\ \alpha+1 & 0 & \alpha \\ \alpha+1 & \alpha & 0 \end{pmatrix}$$

Starting from the fact that we can represent an element in $\mathbb{F}_{2^2}$ with an element in $\mathbb{F}_2^2$, we can rewrite the elements of $\mathbb{F}_{2^2}$ as vectors in the vector space $\mathbb{F}_2^2$. Just consider the polynomial representation of the elements in $\mathbb{F}_{2^2}$ and consider the coefficients as an element in $\mathbb{F}_2^2$.
In our example we have

$$\begin{array}{ll} 0 \longleftrightarrow (0,0) & 1 \longleftrightarrow (1,0) \\ \alpha \longleftrightarrow (0,1) & \alpha+1 \longleftrightarrow (1,1) \end{array}$$

and so we can rewrite $B_\circ$ as

$$B_\circ = \begin{pmatrix} B_{e_1}^T[1] & B_{e_2}^T[1] & B_{e_3}^T[1] \\ B_{e_1}^T[2] & B_{e_2}^T[2] & B_{e_3}^T[2] \\ B_{e_1}^T[3] & B_{e_2}^T[3] & B_{e_3}^T[3] \end{pmatrix} = \begin{pmatrix} 0 & 1 & 1 \\ 0 & 1 & 1 \\ 1 & 0 & 0 \\ 1 & 0 & 1 \\ 1 & 0 & 0 \\ 1 & 1 & 0 \end{pmatrix} = \begin{pmatrix} 0^T & (\alpha+1)^T & (\alpha+1)^T \\ (\alpha+1)^T & 0^T & \alpha^T \\ (\alpha+1)^T & \alpha^T & 0^T \end{pmatrix}$$

With this construction, we can translate the properties that the operation has to respect that can guarantee $\dim(U(T)) = k$:

1. **Zero-summable columns** as the property that for every non-null choice of the columns, their sum[1] has to be different from zero. Otherwise we obtain that $\dim(U(T)) = k + 1$.
   Even if we see the element of the matrix in $\mathbb{F}_{2^k}$, we are not actually working in

[1]The sum is the bitwise-XOR in F_2^k.

the field but use only its element representation.
For this reason, we prefer the representation in $M_{kn,n}(\mathbb{F})$ as, in this case, the property means that the columns are linearly independent.

2. No column has to be a null vector. Otherwise, $\dim(U(T)) = k + 1$. It's a particular case of the previous point.

3. In the form of a matrix in $M_{n,n}(\mathbb{F}_{2^k})$, the matrix $B_\circ$ is symmetric.
Indeed from the fact that $e_j B_{e_i} = e_i B_{e_j}$, we have that the two rows are identical and so they will be represented with the same element in $\mathbb{F}_{2^k}$.

4. In the form of a matrix in $M_{n,n}(\mathbb{F}_{2^k})$, the matrix $B_\circ$ has, in the diagonal, all zeros.
In fact from the fact that $e_i k_{e_i} = 0$, we have that the i-th row is zero and so the i-th row of B_{e_i}. So the elements in the diagonal has to be zero.

Remark 3.2.1. These condition are exactly the translation of the definitions of the variety $\mathcal{I}_k$:

1. The sum for every not-null set of columns is not null is the translation of the S_1 set of the ideal.

2. That the matrix is symmetric is the translation of the set S_2.

3. That the matrix has the null-diagonal is the translation of the set S_3.

Remark 3.2.2. We can represent the matrix $B_\circ$ generated by the B_{e_i}'s as

- an element of $M_{n,n}(\mathbb{F}_{2^k})$ matrix space. We have the merit of having a square, symmetric and with null-diagonal matrix, but with a hard time checking the zero-summable columns property.

- a matrix in $M_{nk,k}(\mathbb{F}_2)$. We have that the zero-summable property can be translated in the maximal rank of $B_\circ$ but we lose the square form that is more comfortable to check the symmetry and the null-diagonal.

For all this reason, we can state

Proposition 3.2.1. *Let V vector space of dimension $N = n + k$ and consider $k = \dim(U(T_\circ))$.*

Let consider M as the matrix obtained by the composition of the B_{e_i} as vectors in $\mathbb{F}_{2^k}^n$.
We have that M is a matrix in $M_{n,n}(\mathbb{F}_{2^k})$

Then we have

- *M has a all zeros diagonal*
- *M is symmetric*
- *M has no zero-summable columns*
  or we can see, M has maximal rank (in the $(nk) \times n$ matrix form)

and, if we define

$$\mathcal{M} := \{m \in M_{n,n}(\mathbb{F}^k) | m \text{ has the conditions listed } \}$$

we have

$$|\mathcal{V}(\mathcal{I}_k)| = |\mathcal{M}|$$

We basically reduced the original variety point count into the research of matrices with defined properties in a matrix space.

Problem 3.2.1. *Given $n \geq 2$ and $k \geq 1$.*
How many matrices $m \in M_{n,n}(\mathbb{F}^k)$ exist that has the conditions of the proposition?

Remark 3.2.3. The matrices $B_\circ$'s are not necessarily in $\mathrm{GL}_n(\mathbb{F}_{2^k})$ as we check the linear independence in $\mathbb{F}_2$ and not in $\mathbb{F}_{2^k}$.

Example 3.2.2.

Let $k = 2$ and $n = 3$. Consider

$$\begin{cases} B_{e_1} = \begin{pmatrix} 0 \\ \alpha+1 \\ \alpha+1 \end{pmatrix} & B_{e_2} = \begin{pmatrix} \alpha+1 \\ 0 \\ \alpha \end{pmatrix} \\ B_{e_3} = \begin{pmatrix} \alpha+1 \\ \alpha \\ 0 \end{pmatrix} \end{cases}$$

If we consider the compact representation in $\mathbb{F}_4$ and check the independence using the extended field, we get that $B_{e_1} = \alpha \cdot (B_{e_2} + B_{e_3})$ and so the vectors are linearly

dependent.
In our case, we are not allowed to *"multiply by element of the extended field"* but just to sum the vectors. The only scalar that we can use is the one in $\mathbb{F}_2$.
And so we have for all $c \in \mathbb{F}_2$, $B_{e_1} \neq c(B_{e_2} + B_{e_3})$. And so $B_{e_1}, B_{e_2}, B_{e_3}$ are non-zero summable.

In fact, if we see the $M_{nk,n}(\mathbb{F})$ representation, it is trivial that the rank is 3.

$$B_\circ = \begin{pmatrix} 0 & 1 & 1 \\ 0 & 1 & 1 \\ 1 & 0 & 0 \\ 1 & 0 & 1 \\ 1 & 0 & 0 \\ 1 & 1 & 0 \end{pmatrix}$$

Recall from the preliminaries, the definition of the invertible $n \times n$ matrices in $\mathbb{F}_{2^k}$ with null-diagonal:

$$\text{Sym0-GL}_n(\mathbb{K}) := \left\{ x \in \text{GL}_n(\mathbb{K}) \;\middle|\; \begin{array}{c} x \text{ has null diagonal, and} \\ x \text{ is symmetric} \end{array} \right\}$$

We can state the trivial inclusion

$$\text{Sym0-GL}_n(\mathbb{F}_{2^k}) \subseteq \mathcal{M}$$

as we know that, for every $m \in \text{GL}_n(\mathbb{F}_{2^k})$ we have $\text{Rk}\, m = n$ and in particular it means that no linear combination of null coefficient in $\mathbb{F}_{2^k}$, is null.

But we have that $\mathbb{F}_2 \subseteq \mathbb{F}_{2^k}$ and so there are no linear combination using coefficients in $\mathbb{F}_2$.
This means that if we represent m as a $(nk) \times n$ matrix, we have that m have maximal rank n. And so $m \in \mathcal{M}$.

3.3 Matrix Bound

We can exploit the possibility to interchange between the two representation. For this reason we can easily state a lower bound:

Proposition 3.3.1 (Matrix Lower Bound)**.** *Let $N := n+k$ and $V := \mathbb{F}^N$, with $n \geq 2$ and $k \geq 1$.*
Then we can define a lower bound $\nu(n,k) \leq |\mathcal{V}(\mathcal{I}_k)|$ by

$$\nu(n,k) := \begin{cases} q^{\binom{n}{2}} \prod_{j=1}^{\lceil \frac{n-1}{2} \rceil} (1 - q^{1-2j}) \ \textit{with } q = 2^k & n \textit{ even} \\ 0 & n \textit{ odd} \end{cases}$$

Proof.
If we consider the Remark 3.2.3, we have that $\text{Sym0-GL}_n(\mathbb{F}_{2^k}) \subseteq \mathcal{M}$.
By this

$$\nu(n,k) := \#\,\text{Sym0-GL}_n(\mathbb{F}_{2^k}) \leq \#\mathcal{M} = \#\mathcal{V}(\mathcal{I}_k)$$

To count: note that $\nu(n,k)$ is composed by invertible symmetric $n \times n$ matrix with null diagonal in $\mathbb{F}_{2^k}$.
So the case n even follows directly from Theorem 2.1.3.
For the case n odd, observe that we are in characteristic 2. This means that symmetric matrix with zero diagonal are also skew-symmetric.
And so, by Remark 2.1.1, the cardinality of the set of matrices is zero. □

Corollary 3.3.1.
If $k = 1$ or $n = 2$, we get that the lower bound is the exact number of solutions.

Proof. For the case of $k = 1$, we have that $\mathbb{F}_{2^k} = \mathbb{F}_2$. And so the conditions of the Proposition 3.2.1 give $\mathcal{M} = \text{Sym0-GL}_n(2)$.
Similarly, if $n = 2$, we have that $\mathcal{M} = \text{Sym0-GL}_2(\mathbb{F}^k)$ □

We can see in the Table 3.1, the behaviour of the lower bound with respect the exact number of operation and the variety bound.

The data that we can compare with the exact number is limited to $n = 7$ as the algorithm that counts the exact number of operation $\circ$ needs a consistent time to compute the result.

$N = n + k$	k	Variety Bound	Exact Value	Matrix Lower Bound
4	2	3	3	3
5	1	998	28	28
5	2	42	42	0
5	3	7	7	7
6	2	3969	3360	3024
6	3	462	462	0
6	4	15	15	15
7	1	32711	13888	13888
7	2	1044630	937440	0
7	3	260729	254968	228928
7	4	3990	3990	0
7	5	31	31	31

Table 3.1: Behaviour of the Matrix bound and Calderini's bound with respect the exact number of operations $\circ$ with fixed dimension N and k.

We recall a theorem on the productory convergence criteria, corollary of the monotone convergence theorem (more detail can be found in the book [Ste11]).

Lemma 3.3.2.
Let $\{a_n\}_{n\in\mathbb{N}} \subseteq \mathbb{R}$. *Then*

$$\exists \lim_{n\to\infty} \prod_{j=1}^{\infty} a_n \iff \exists \lim_{n\to\infty} \ln(a_n)$$

and if $a_n \geq 1$, *write it as* $a_n = 1 + p_n$. *Then*

$$1 + \sum_{i=1}^{\infty} p_n \leq \prod_{j=1}^{\infty} (1 + p_n) \leq e^{\sum_{i=1}^{\infty} p_n}$$

We now compare the Calderini's bound and the matrix lower bound.

Proposition 3.3.3.

Let $\mu(n,k)$ *be the variety bound and let* $\nu(n,k)$ *be the matrix bound. Let* $q = 2^k$. *Then*

$$\lim_{n\to\infty} \frac{\mu(n,k)}{\nu(n,k)} = \prod_{j=1}^{\infty} \frac{1}{(1-q^{1-2j})} \leq e^{\frac{1}{q-1}}$$

Proof.
Define

$$\mu_1(n,k) := q^{\binom{n}{2}} \qquad \mu_2(n,k) := \sum_{r=1}^{n-2} \binom{n}{r} \prod_{i=1}^{\binom{n-r}{2}} (q-1)$$

so we can rewrite

$$\mu(n,k) = \mu_1(n,k) - 1 - \mu_2(n,k) = \mu_1(n,k) \left(1 - \frac{1}{\mu_1(n,k)} - \frac{\mu_2(n,k)}{\mu_1(n,k)}\right)$$

We can now rewrite and see the μ_2's asymptotic equivalence by calculating

$$\begin{aligned} \frac{\mu_2(n,k)}{\mu_1(n,k)} &= \frac{\sum_{r=1}^{n-2} \binom{n}{r} \prod_{i=1}^{\binom{n-r}{2}} (q-1)}{\mu_1(n,k)} \\ &= \frac{\sum_{r=1}^{n-2} \binom{n}{r} (q-1)^{\binom{n-r}{2}}}{q^{\binom{n}{2}}} \end{aligned}$$

Consider the case $q = 2$ (so $k = 1$):

$$\begin{aligned}
\frac{\mu_2(n,k)}{\mu_1(n,k)} &= \frac{\sum_{r=1}^{n-2} \binom{n}{r} \prod_{i=1}^{\binom{n-r}{2}} (2-1)}{\mu_1(n,k)} \\
&= \frac{\sum_{r=1}^{n-2} \binom{n}{r} (2-1)^{\binom{n-r}{2}}}{2^{\binom{n}{2}}} \\
&= \frac{\sum_{r=1}^{n-2} \binom{n}{r}}{2^{\binom{n}{2}}} \\
&\leq \frac{\sum_{r=0}^{n} \binom{n}{r}}{2^{\binom{n}{2}}} \\
&= \frac{2^n}{2^{\binom{n}{2}}} \sim_\infty 0
\end{aligned}$$

For the case $q > 2$:

$$\begin{aligned}
\frac{\mu_2(n,k)}{\mu_1(n,k)} &=\leq \frac{\sum_{r=1}^{n-2} \binom{n}{r} (q)^{\binom{n-r}{2}}}{q^{\binom{n}{2}}} \\
&= \sum_{r=1}^{n-2} \binom{n}{r} q^{\binom{n-r}{2} - \binom{n}{2}}
\end{aligned}$$

From the fact that, for every $r \in 1..(n-2)$, we have $\binom{n-r}{2} \leq \binom{n}{2}$. We so can state that

$$q^{\binom{n-r}{2} - \binom{n}{2}} \leq q^{\binom{n-1}{2} - \binom{n}{2}} \leq 1 \qquad \forall n \in \mathbb{N}$$

Now:

$$\begin{aligned}
\frac{\mu_2(n,k)}{\mu_1(n,k)} &\leq \sum_{r=1}^{n-2} \binom{n}{r} q^{\binom{n-r}{2} - \binom{n}{2}} \\
&\leq \sum_{r=1}^{n-2} \binom{n}{r} q^{\binom{n-1}{2} - \binom{n}{2}} \\
&= q^{\frac{n^2-3n+2-n^2+n}{2}} \left(\sum_{r=1}^{n-2} \binom{n}{r} \right) \\
&\leq q^{-n+1} \left(\sum_{r=0}^{n} \binom{n}{r} \right) = q^{-n+1} 2^n = q \left(\frac{2}{q} \right)^n
\end{aligned}$$

The limit converge if and only if $|\frac{2}{q}| \leq 1$. In our case, $q > 2$ and so we have

$$\frac{\mu_2(n,k)}{\mu_1(n,k)} \leq q\left(\frac{2}{q}\right)^n \sim_\infty 0$$

From basic analysis, we have that for two successions a_n, b_n such that $a_n \ll_\infty b_n$ (a_n is negligible with respect to b_n), then $a_n + b_n \sim_\infty b_n$.
In our case we have

$$\mu_1(n,k) \sim_\infty q^{\binom{n}{2}} \qquad \mu_2(n,k) \ll_\infty \mu_1(n,k)$$

from which we have

$$\mu(n,k) = \mu_1 - 1 - \mu_2 \sim_\infty q^{\binom{n}{2}} = \mu_1(n,k)$$

And so we get

$$\frac{\mu(n,v)}{\nu(n,v)} \sim_\infty \frac{q^{\binom{n}{2}}}{q^{\binom{n}{2}}\prod_{j=1}^{\infty}(1-q^{1-2j})} = \prod_{j=1}^{\infty}\frac{1}{1-q^{1-2j}}$$

With the Lemma 3.3.2, we can rewrite

$$\frac{1}{1-q^{1-2j}} = \frac{q^{2j-1}}{q^{2j-1}-1} = 1 + \frac{q}{q^{2j}-q}$$

and define

$$p_n := \frac{q}{q^{2j}-q}$$

We trivially have that $p_n \to_\infty 0$ and $p_n \geq 0$.

From the lemma, we have that

$$1 + \sum_{i=1}^{\infty} p_n \leq \prod_{j=1}^{\infty}\frac{1}{(1-q^{1-2j})} \leq e^{\sum_{i=1}^{\infty} p_n}$$

We have for any $n > 0$

$$\frac{q}{(q^2)^n} \leq p_n \leq \frac{1}{q^n}$$

From this, we get

$$\frac{q}{1-q^2} \leq \sum_{i=1}^{\infty} p_n \leq \frac{1}{q-1}$$

and so

$$\prod_{j=1}^{\infty} \frac{1}{(1-q^{1-2j})} \leq e^{\frac{1}{q-1}}$$

□

Corollary 3.3.2.

$$\lim_{n,k\to\infty} \frac{\mu(n,k)}{\nu(n,k)} = 1$$

Proof.
The limit for $k \to \infty$ is equivalent to the limit for $q \to \infty$. After the limit $n \to \infty$, if we do an asymptotical equivalence with respect to q and get

$$\prod_{j=1}^{\infty} \frac{1}{(1-q^{1-2j})} \leq e^{\frac{1}{q-1}} \sim_{\infty} 1$$

□

With the lower bound and the different representation of $B_\circ$, we can extend the results of Calderini on the exact cardinality of the variety for $k = 1$ and rewrite the proof in a different way.

Proposition 3.3.4. *Let $N := n + k$ and $V := \mathbb{F}^N$, with $n \geq 2$ and $k \geq 1$. Then*

$$|\mathcal{V}(\mathcal{I}_k)| = N(n,k) = \begin{cases} 2^k - 1 & n = 2 \\ (2^k+3)(2^k-1)(2^k-2) & n = 3 \\ 2^{\binom{n}{2}} \prod_{j=1}^{\lceil \frac{n-1}{2} \rceil} (1-2^{1-2j}) & k = 1 \text{ and } n \text{ even} \\ 0 & k = 1 \text{ and } n \text{ odd} \end{cases}$$

Proof.

Case 3.3.1 ($n = 2$).
Just consider the matrices

$$M = \begin{pmatrix} 0 & x \\ x & 0 \end{pmatrix} \qquad x \in \mathbb{F}_{2^k} \text{ such that } M \text{ respect the condition of Prop. 3.2.1}$$

In this case, we get exactly $\mathrm{sym}_0(2)$ with $q = 2^k$ because the vectors of the matrix are orthogonal one with the other.

Case 3.3.2 ($n = 3$).
Consider the matrices

$$M = \begin{pmatrix} 0 & y & z \\ y & 0 & x \\ z & x & 0 \end{pmatrix} \qquad x, y, z \in \mathbb{F}_{2^k} \text{ such that } M \text{ respect the condition of Prop. 3.2.1}$$

Let consider the different cases:

- $x \neq 0$: $2^k - 1$ possible values
  - $y = 0$: so $z \notin \{0, x\}$ and so we get $2^k - 2$ possible values
  - $y = x$: so $z \notin \{0, x\}$ and so we get $2^k - 2$ possible values
  - $y \neq x$: so z can be any element, $y \notin \{0, x\}$. So $2^k(2^k - 2)$ possible pairs
- $x = 0$: $y \neq 0 \wedge z \neq \{0, y\}$ and so we get $(2^k - 1)(2^k - 2)$ possible pairs

Summing all the possible triple, we get

$$(2^k - 1)(2^k - 2 + 2^k - 2 + 2^k(2^k - 2)) + (2^k - 1)(2^k - 2) = (2^k - 1)(2^k - 2)(2^k + 3)$$

Case 3.3.3 ($k = 1$).
As we consider $\mathbb{F}_{2^k} = \mathbb{F}_2$, we have that $M_{n,n}(\mathbb{F}_2) = \text{Sym0-GL}_n(\mathbb{F}_2)$.
And so we are just counting the invertible, skew-symmetric invertible matrix over $\mathbb{F}$ with dimension n. They are

$$\text{sym}_0(n) = \begin{cases} 2^{\binom{n}{2}} \prod_{j=1}^{\lceil \frac{n-1}{2} \rceil} (1 - 2^{1-2j}) & n \text{ even} \\ 0 & n \text{ odd} \end{cases}$$

□

3.4 Exact number algorithm

In the appendix of [Cal15], it is presented an algorithm that counts the number of operations with the form of the Theorem 2.4.7 with fixed $N = n + k$ and k.
We present an algorithm that generate the translation group $T_\circ$ using the matrix representation and then check the dimension of $U(T_\circ)$ with the rank of the matrix $B_\circ$.

Algorithm 3.4.1. *Let $n, k \in \mathbb{N}$ with $n \geq 2$ and $k \geq 1$. The main goal is having $\dim U(T_\circ) = k$.*
We know from the matrix construction of an operation $\circ$ that we need n matrix of dimension $n \times k$.

Consider the matrix $B_\circ \in \mathcal{M}$ where we indicate with $*$ a possible element of $\mathbb{F}_{2^k}$:

$$B_\circ = \begin{pmatrix} 0 & * & \cdots & * \\ * & 0 & \ddots & \vdots \\ \vdots & \ddots & \ddots & * \\ * & \cdots & * & 0 \end{pmatrix}$$

As we can observe, $B_\circ$ have $n^2 - n$ elements $*$ in $\mathbb{F}_{2^k}$. For the symmetry of $B_\circ$, the number of $*$ is $\binom{n}{2}$.

We construct the B_{e_i}'s as:

1. *Construct B_{e_1} with $n-1$ elements in $\mathbb{F}_{2^k}$.*

2. *From the fact that $T_\circ$ is abelian, we have that*

$$e_1 \circ e_2 = e_1 k_{e_2} + e_2 = e_2 k_{e_1} + e_2 = e_2 \circ e_1$$

$$e_1 k_{e_2} + e_1 = e_2 k_{e_1} + e_2$$

$$e_1(k_{e_2} + I_{n+k}) = e_2(k_{e_1} + I_{n+k})$$

   and so the second row of $k_{e_2} + I_{n+k}$ has to be equal to the first row of $k_{e_1} + I_{n+k}$. Starting from the fact that we are considering the operation $\circ$ with form described in Theorem 2.4.7, we have that the second row of B_{e_1} is equal to the first row of B_{e_2}.
   Then, the second element of B_{e_2} is zero.
   Then we can append $n-2$ elements of $\mathbb{F}_{2^k}$.

3. *In the same way, we can create the $B_{e_{i+1}}$ vector using the $i+1$-th position of the B_{e_j} with $j < i+1$, append a zero that will be the $i+1$-th element of $B_{e_{i+1}}$ and then $n-(i+1)$ elements of $\mathbb{F}_{2^k}$.*

4. *To construct B_n, we take the n-th element of every B_{e_i} for $i \in 1..(n-1)$ and then append a 0.*

The construction permits us to easily create $B_\circ$ using $\binom{n}{2}$ elements of $\mathbb{F}_{2^k}$.
For this reason, consider the cartesian product of $\binom{n}{2}$ times $\mathbb{F}_{2^k}$, indexed with i and j:

$$O := \prod_{i=1}^{n-1}\prod_{j=1}^{n-i}\mathbb{F}_{2^k} = \underbrace{(\mathbb{F}_{2^k}\times\cdots\times\mathbb{F}_{2^k})}_{n-1\ elements}\times\underbrace{(\mathbb{F}_{2^k}\times\cdots\times\mathbb{F}_{2^k})}_{n-2\ elements}\times\cdots\underbrace{(\mathbb{F}_{2^k})}_{1\ element})$$

The set is divided in such a way that for every i, there are exactly $n-i$ element of $\mathbb{F}_{2^k}$.
For $u \in O$, we can generate an operation $\circ$ with the construction described before.

Every element in this set will generate an operation $\circ$ over $\mathbb{F}^{n+k}$ and the operation $\circ$ generates an abelian elementary subgroup $T_\circ \subseteq \mathrm{AGL}_+$.
The operations $\circ$ constructed in this way, will have, for the B_{e_i}'s, matrix dimension n and k and they will be in form of the Theorem 2.4.7 and we will have that $\dim(U(T_\circ)) \geq k$.

For every $u \in O$, we generate the matrix $B_\circ$ described in Section 3.2. $B_\circ$ will represent a single operation $\circ$.
We are now interested in searching only the operations $\circ$ that has $\dim(U(T_\circ)) = k$.
This is guaranteed if and only if $\mathrm{Rk}(B_\circ) = n$ when we consider $B_\circ$ in the form of a $(nk) \times n$ matrix over $\mathbb{F}_2$.

We compacted the construction idea in the diagram in Figure 3.1.

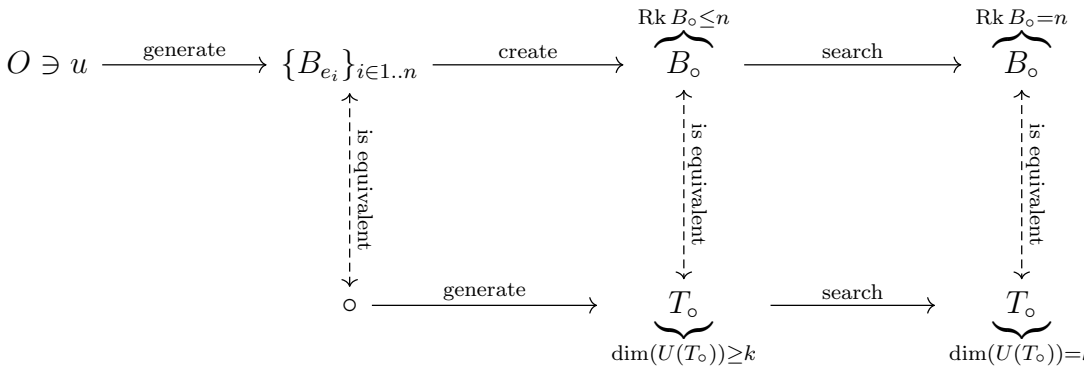

Figure 3.1: main idea from the set O to the operation $\circ$ that has $\dim(U(T_\circ)) = k$

We describe, in the Figure 3.2, the work-flow diagram of the matrix construction

algorithm:

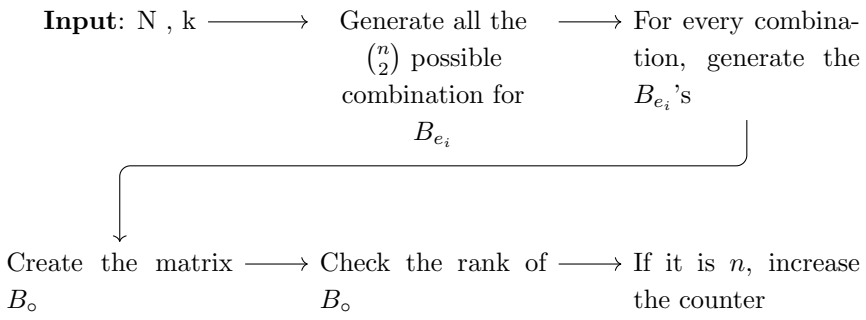

Figure 3.2: work-flow of the Matrix construction

Proposition 3.4.1. *Let $n \geq 2$ and $k \geq 1$.*
Assume that:

- *Generating O has cost $\mathcal{O}(1)$*
- *Creating the B_{e_i}'s for all i has cost $\mathcal{O}(n)$*
- *Check the rank of a matrix $n \times m$: using Gauss elimination as algorithm. Cost: $\mathcal{O}(\min(n,m) \cdot n \cdot m)$*

Then the computational complexity of the matrix construction algorithm is $\mathcal{O}\left(2^{k\binom{n}{2}}(n^3k)\right)$

Proof.
Following the algorithm:

+ Generate O. $\#O = 2^{k\binom{n}{2}}$. Cost 1
+ For every $o \in O$:
  - Generate $B_\circ$. Cost: n
  - Checking the rank for a single $\circ$. Cost: n^3k

So the complexity is

$$\mathcal{O}\left(1 + 2^{k\binom{n}{2}}(n + n^3k)\right) \quad \sim \quad \mathcal{O}\left(2^{k\binom{n}{2}}(n^3k)\right)$$

□

The main difference with respect to Calderini's algorithm is the use of linear algebra and not algorithms for the group representation and intersection.

Algorithm 3.4.2 (Calderini's counting algorithm)**.**
Let $n \geq 2$, $k \geq 1$. Let T_+ be the translation group of $\mathbb{F}^{n+k}$.

To count the operation $\circ$ such that the group $T_\circ$ has $\dim(U(T_\circ)) = k$, the algorithm proceeds:

1. *The construction of the set of B_{e_i}'s is fundamentally equivalent:*
   (a) *Generate O as before.*
   (b) *For every $u \in O$, generate the B_{e_i}'s and the k_{e_i}'s inserting B_{e_i}'s in the upper-right corner.*
2. *Create the group $T_\circ = \langle \{\tau_{e_i} = k_{e_i}\sigma_{e_i}\}_{i \in 1..(n+k)} \rangle$.*
3. *Intersect $T_\circ \cap T_+$ and check the cardinality. If it is 2^k, then increase the counter.*

In Figure 3.3, we describe the work-flow of Calderini's bound

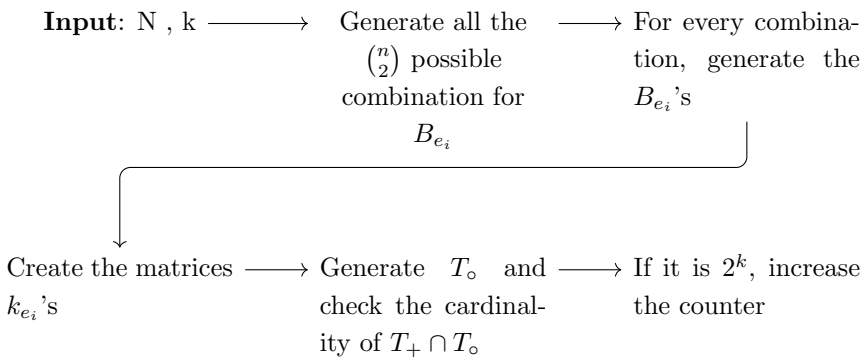

Figure 3.3: work-flow of Calderini's bound

Remark 3.4.1. The two algorithms work as a brute force and test **all** the possible operations $\circ$.
A first improvement should reduce the searching set O or construct it in a clever way.

We tested the algorithms and checked the correctness of the number of sums with fixed $N = n + k$ and k.

Calderini, in his Thesis [Cal15], present the exact number of operation $\circ$ only for $n \in \{3, 4, 5, 6\}$, with all the admissible k.
We extended the calculation even to $n = 7$ with the two algorithms.

The times and operations found are reported in Table 3.2.
To do the computation, we used a standard MacBook Pro (late 2013) with a Intel-i5 and 8 GB of RAM.

$N = n + k$	k	Number of operations	Calderini's time	Matrix time
3	1	1	0.000	0.010
4	1	0	0.000	0.000
4	2	3	0.000	0.000
5	1	28	0.040	0.010
5	2	42	0.080	0.010
5	3	7	0.000	0.000
6	1	0	1.310	0.220
6	2	3360	12.880	0.730
6	3	462	1.440	0.060
6	4	15	0.020	0.000
7	1	13888	99.580	9.240
7	2	937440	15771.740	239.150
7	3	254968	5947.070	45.720
7	4	3990	39.240	0.490
7	5	31	0.090	0.000

Table 3.2: computation time in seconds and comparing test of Calderini's and Matrix bound

As we can see, in this case, the use of linear algebra in the algorithm rather than finite represented group algorithms is a better choice.

Hidden sum algorithms

In this chapter different algorithm to generate, manage and search hidden sums will be presented.
We will start with an algorithm to generate a single operation $\circ$ and define a bijection between all the $\circ$ operation and a numerical space that can be easily software implemented to save and archive the operation.

Successively, we will start to create different algorithms to search hidden sums for the different blocks of a block cipher:

- **Mixing layer** as the $\circ$ that mantain the linearity of the mixing layer
- **Substitution Box** in the differential properties and the possibility to linearise them
- **Block Ciphers** seen as the composition of the Sbox and Mixing Layer

In this thesis we will concentrate on the mixing layer.
The properties and algorithms for the substitution box and the entire block cipher are not developed in this thesis.

4.1 Construct a generic hidden sum

In this section, there will be defined an algorithm that generate the operation $\circ$ from a set of input parameters.

Algorithm 4.1.1 (Construct a single operation $\circ$)**.**
Let $n \geq 2$ and $k \geq 1$.
Consider the construction of $B_\circ$, as described in Section 3.2, from the set

$$O := \prod_{i=1}^{n-1} \prod_{j=1}^{n-i} \mathbb{F}^k$$

As described in Algorithm 3.4.1, every $u \in O$ will generate an operation $\circ$.

We choose a single u as $\binom{n}{2}$ elements of $\mathbb{F}_{2^k}$.

To complete the algorithm, add k copies of the null matrix to obtain all the B_{e_i}'s. The set $\{B_{e_i}\}$ can be adjusted to create the set $\{k_{e_i}\}$ from which we can generate the maps τ_{e_i} in the form of the Theorem 2.4.7.

***As a Python-like pseudo-code** :*
*Consider **coeff** as a list of number list.*

```

#
# Extend the coefficients of 'o'
# Input :
# coeff : the coefficients to insert
#
def extendCartesian( coeff ):
  valors = [
    [ coeff[j][i-j] : j in [1..(i-1)] ]
    cat coeff[i] : i in [1..#coeff] ]
  return valors
```

```

#
# Generate B_ei
# Input :
# valors : the coefficients to insert
# i : which B_ei we want to generate
#
def generateBei( valors , i ):

  # Generate empty matrix
  Bei = ZeroMatrix(GF(2), n , k )

  # Insert a list of k zeros in the i-th position
  valors = [valors[j] : j in [1..(i-1)] cat [0 : j in [1..k]] cat [valors[j] : j
    in [i..n]]

  for i in range(n):
    # Insert in Bei the valors vector
    Bei = InsertRow(Bei, valors[i] , i)

  return Bei
```

Here is an example to understand the functionality of the two functions:

extendedCartesian *generate the list that will become* $B_\circ$ *without the null-diagonal. Every element of the output is the list of values that will become the rows of* B_{e_i}*. In the list for a* B_{e_i}*, it is absent the* i*-th element of* B_{e_i}*.*

$$((a_1, a_2, a_3), (a_4, a_5), (a_6), ()) \longrightarrow ((a_1, a_2, a_3), (a_1, a_4, a_5), (a_2, a_4, a_6), (a_3, a_5, a_6))$$

generateBei *does the transformation from a list of value to the matrix* B_{e_i}*. For example in* $n = 4$ *and* $k = 2$*:*

$$(1, \alpha, \alpha + 1) \text{ and } i = 2 \longrightarrow B_{e_2} = \begin{pmatrix} 1 & 0 \\ 0 & 0 \\ 0 & 1 \\ 1 & 1 \end{pmatrix} = \begin{pmatrix} 1 \\ 0 \\ \alpha \\ \alpha + 1 \end{pmatrix}$$

```
1
2 #
3 # Generate the different operation
4 # Input :
5 # n,k : total dimension and dimension for generate the matrixs
6 #   coeff: the coefficient to use for generating 'o'
7 # Output :
8 # B_ei : list of all the B_ei for the canonical base
9 #
10 def generateOperation(n,k,coeff):
11   zero = ZeroMatrix(GF(2),n,k)
12
13   valors = extendCartesian(coeff)
14   B_ei = [ generateBei(valors[i],i) : i in range(n) ]
15
16   # Add the remaining k zero matrix
17   B_ei = B_ei cat [zero]*k
18   return B_ei
```

generateOperation*, for example, with* $n = 4$ *and* $k = 2$*:*

$$((\alpha, \alpha+1, 1), (0,1), (\alpha+1), ()) \to \begin{cases} B_{e_1} = \begin{pmatrix} 0 & 0 \\ 0 & 1 \\ 1 & 1 \\ 1 & 0 \end{pmatrix} & B_{e_2} = \begin{pmatrix} 0 & 1 \\ 0 & 0 \\ 0 & 0 \\ 1 & 0 \end{pmatrix} \\ B_{e_3} = \begin{pmatrix} 1 & 1 \\ 0 & 0 \\ 0 & 0 \\ 1 & 1 \end{pmatrix} & B_{e_4} = \begin{pmatrix} 1 & 0 \\ 1 & 0 \\ 1 & 1 \\ 0 & 0 \end{pmatrix} \\ B_{e_5} = \begin{pmatrix} 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \end{pmatrix} & B_{e_6} = \begin{pmatrix} 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \end{pmatrix} \end{cases}$$

After generating the B_{e_i}'s, we can calculate all B_v's. So we obtain all the τ_v's adding the B_v in an identity matrix of dimension $n+k$.

If all $B_{e_i} = 0$, we have that the operation $\circ$ is exactly the XOR operation $+$. Otherwise $T_\circ \neq T_+$ and $T_+ \subset \mathrm{AGL}_\circ$ by construction.

Note 4.1.1.
The fixed k is just a lower bound of the dimension of $U(T_\circ)$.
From the input coefficients, we can get a bigger $U(T_\circ)$. To test that the dimension is really k, it is just needed to construct $B_\circ$ and check if the rank is n.

With the algorithm, we create an 1-1 correspondence δ between

$$O \overset{\delta}{\longleftrightarrow} \left\{ \{B_{e_i}\}_{i \in 1..(n+k)} \;\middle|\; \begin{array}{c} T_\circ \text{ in Theorem 2.4.7 form, and} \\ \dim(U(T_\circ)) \geq k \end{array} \right\}$$

If we need to save the operation $\circ$, it has a space complexity of $\mathcal{O}\left(k\frac{n(n-1)}{2}\right)$.
Just consider the fact that we need to save $\binom{n}{2}$ numbers that can be represent in k bits.

The algorithm, on a standard computer, create a random single operation $\circ$ with $(n+k, k) = (128, 64)$ in ~ 0.420 seconds and saving the operation occupies 15.75 kB in memory.

In Appendix A.2, a Magma code of the algorithm is presented and explained.

4.2 Mixing Layer

In this section, we will explain the steps made to achieve the construction of an algorithm that search all the operation $\circ$ that linearize a linear map $\lambda \in \mathrm{GL}_+(n)$.

Suppose we have a $\lambda \in \mathrm{GL}_+ \cap \mathrm{GL}_\circ$.
We write λ as a block matrix, $\lambda = \begin{pmatrix} A & D \\ C & B \end{pmatrix}$.

We are interested in the conditions that make λ contained in $\mathrm{GL}_\circ \cap \mathrm{GL}_+$.

Proposition 4.2.1. *Let $n \geq 2$ and $k \geq 1$. Let $V = \mathbb{F}^{n+k}$.*
Let $\lambda \in \mathrm{GL}_+ \cap \mathrm{GL}_\circ$ with $\mathrm{T}_\circ \subseteq \mathrm{AGL}_+$ abelian elementary regular subgroup such that $T_+ \subseteq \mathrm{AGL}_\circ$.
Then, for every $y \in V$, if $\tau_y = M_y\sigma_y$, we have that $M_y\lambda = \lambda M_{y\lambda}$.

Proof.
Let $x, y \in V$. Let $\tau_y = M_y\sigma_y = (M_y + y)$. For the linearity of λ with respect to the operation $\circ$ we have

$$(x \circ y)\lambda = x\lambda \circ y\lambda$$

where

$$\begin{aligned}(x \circ y)\lambda &= (xM_y + y)\lambda \\ &= xM_y\lambda + y\lambda\end{aligned}$$

and

$$x\lambda \circ y\lambda = x\lambda M_{y\lambda} + y\lambda$$

Imposing the equality we get

$$xM_y\lambda + y\lambda = x\lambda M_{y\lambda} + y\lambda \Longrightarrow x(M_y\lambda) = x(\lambda M_{y\lambda}) \Longrightarrow M_y\lambda = \lambda M_{y\lambda}$$

□

Corollary 4.2.1. *With the hypothesis of the Proposition 4.2.1, write λ as a block matrix $\lambda = \begin{pmatrix} A & D \\ C & B \end{pmatrix}$.*
Then $\lambda = \begin{pmatrix} A & D \\ 0 & B \end{pmatrix}$ with

$$\begin{cases} A \in \mathrm{GL}_+(n) \\ B \in \mathrm{GL}_+(k) \\ D \in M_{n,k}(\mathbb{F}_q) \\ B_y B = A B_{y\lambda} & \forall y \in V \end{cases}$$

Proof. Write λ as a block matrix, $\lambda = \begin{pmatrix} A & D \\ C & B \end{pmatrix}$.

From the hypothesis :$\lambda \in \mathrm{GL}_+ \cap \mathrm{GL}_\circ$.
From the Proposition 4.2.1:

$$M_y\lambda = \lambda M_{y\lambda} \iff \begin{pmatrix} I_n & B_y \\ 0 & I_k \end{pmatrix} \begin{pmatrix} A & D \\ C & B \end{pmatrix} = \begin{pmatrix} A & D \\ C & B \end{pmatrix} \begin{pmatrix} I_n & B_{y\lambda} \\ 0 & I_k \end{pmatrix}$$

and obtain

$$\begin{pmatrix} I_n & B_y \\ 0 & I_k \end{pmatrix} \begin{pmatrix} A & D \\ C & B \end{pmatrix} = \begin{pmatrix} A + B_y C & D + B_y B \\ C & B \end{pmatrix}$$

$$\begin{pmatrix} A & D \\ C & B \end{pmatrix} \begin{pmatrix} I_n & B_{y\lambda} \\ 0 & I_k \end{pmatrix} = \begin{pmatrix} A & AB_{y\lambda} + D \\ CB_{y\lambda} & B \end{pmatrix}$$

Imposing the equality, we get

$$\begin{cases} B_y B = A B_{y\lambda} & \\ B_y C = 0 & \forall_{y \in V} \\ C B_{y\lambda} = 0 & \forall_{y \in V} \end{cases}$$

We claim that $C = 0$.
In fact, from the fact that $U(T_\circ)\lambda = U(T_\circ) = \mathrm{Span}\{e_{n+1}, \cdots, e_{n+k}\}$

$$U(T_\circ)\lambda = U(T_\circ)$$

$$\begin{pmatrix} 0 & 0 \\ 0 & I_k \end{pmatrix} \begin{pmatrix} A & D \\ C & B \end{pmatrix} = \begin{pmatrix} 0 & 0 \\ C & B \cdot I_k \end{pmatrix} \begin{pmatrix} 0 & 0 \\ C & B \end{pmatrix}$$

from which we obtain $C = 0$ and $B \in \mathrm{GL}_+$.

We have that λ is of the form

$$\lambda = \begin{pmatrix} A & D \\ 0 & B \end{pmatrix}$$

Note that $\lambda \in GL_+ \implies \det(\lambda) = \det(A)\det(B) \neq 0$.
So $A \in \mathrm{GL}_+(n)$ and $B \in \mathrm{GL}_+(k)$. □

Remark 4.2.1. From the proof of the proposition, we observe that the matrix D can be any matrix $M_{n,k}(\mathbb{F})$ as it is irrelevant in the condition of the definition.

For an operation $\circ$ over $V = \mathbb{F}^{n+k}$ such that $T_\circ \subseteq \mathrm{AGL}_+$ abelian elementary subgroup such that $T_+ \subseteq \mathrm{AGL}_\circ$ abelian subgroup, we have that $\lambda \in \mathrm{GL}_+ \cap \mathrm{GL}_\circ$ if the condition of the Corollary 4.2.1 are met.

Let $x = (x_1, x_2), y = (y_1, y_2) \in V = \mathbb{F}^{n+k}$ and suppose we have a $\lambda \in \mathrm{GL}_+ \cap \mathrm{GL}_\circ$. We will consider the two different equations:

$$(x + y)\lambda = x\lambda + y\lambda \qquad (x \circ y)\lambda = x\lambda \circ y\lambda$$

The first equality:

$$\begin{aligned} x\lambda + y\lambda &= (x_1A + y_1A, x_1D + y_1D + x_2B + y_2B) \\ &= (x + y)\lambda \end{aligned} \tag{4.1}$$

and the second:

$$\begin{aligned} x\lambda \circ y\lambda &= (x_1A, x_1D + x_2B) \circ (y_1A, y_1D + y_2B) \\ &= (x_1A, x_1D + x_2B)\left(\begin{pmatrix} I & B_{y\lambda} \\ 0 & I \end{pmatrix} + (y_1A, y_1D + y_2B)\right) \\ &= (x_1A + y_1A, x_1AB_{y\lambda} + x_1D + x_2B + y_1D + y_2B) \\ &\quad \text{because } \lambda \in \mathrm{GL}_+ \cap \mathrm{GL}_\circ, \text{ we have for all } v \in V\text{: } AB_{v\lambda} = B_vB \\ &= (x_1A + y_1A, x_1B_yB + x_1D + x_2B + y_1D + y_2B) \\ &= (x_1 + y_1, x_1B_y + x_2 + y_2)\lambda \\ &= (x \circ y)\lambda \end{aligned} \tag{4.2}$$

If we compare the equations 4.1 and 4.2, we notice that

$$\underbrace{(x \circ y)\lambda}_{4.1} = \underbrace{(x_1A + y_1A, x_1B_yB + x_1D + x_2B + y_1D + y_2B) = (x + y)\lambda + (0, x_1B_yB)}_{4.2}$$

So we get that λ acts on the operation $\circ$ as it does on the operation $+$ with a *"difference"* that can be collected in the value $(0, x_1B_yB)$.
We can see this value as the $x \cdot y$ in the radical ring description that is present in the preliminaries Subsection 2.4.1.

Now we can recall the condition for an operation $\circ$ and connect them to λ:

1. If for all $x, y \in V$ as $x = (x_1, x_2)$ and $y = (y_1, y_2)$ we have that $x_1B_yB = 0$, then we have that $T_+ \equiv T_\circ$.

2. From the fact that $T_\circ$ is abelian

$$x_1B_yB = y_1B_xB$$

3. From the associativity property of $T_\circ$

$$((x \circ y) \circ z)\lambda = (x \circ (y \circ z)) \implies x_1(B_y + B_z)B = x_1B_{y+z}B$$

4. From the Corollary 4.2.1

$$x_1B_yB = x_1B_{y+v}B = 0$$

   where $v \in U(T_o)$.
   Note that for $v \in U(T_\circ)$ and any $v \in V$, we have $B_y = B_{y+v}$ because $\tau_y\tau_{y+v} = \tau_y\tau_y\tau_v = \tau_v$.

5. From the fact that $T_\circ$ is elementary, we have that $y\tau_y = 0$ and so $y_1B_yB = 0$.

We can now state the main problem of the section:

Problem 4.2.1. Let $n \geq 2$ and $k \geq 1$. Let $V = \mathbb{F}^{n+k}$.
Let $\lambda \in \mathrm{GL}_+(n)$ mixing layer. Does another operation $\circ$ such that $\lambda \in \mathrm{GL}_\circ(n)$ exist?

4.2.1 SearchLinearity Algorithm

At this point we have sufficient condition on the blocks of λ to state an algorithm that retrive the operations $\circ$.
We assume that $U(T_\circ)$ is generated by the last k vector of the canonical basis.

Algorithm 4.2.1 (SearchLinearity)**.**
Let $n \geq 2$ and $k \geq 1$. Let $V = \mathbb{F}^{n+k}$.
Let $\lambda \in \mathrm{GL}_+$ in the block form $\lambda = \begin{pmatrix} A & D \\ 0 & B \end{pmatrix}$ with $A \in \mathrm{GL}_+(n)$ and $B \in \mathrm{GL}_+(k)$.

Consider $\{e_1, \ldots, e_{n+k}\}$ the canonical basis of V.

Note that we have that $\tau_{e_i} = \sigma_{e_i}$ for $e_i \in U(T_\circ)$ and so for $i \in (n+1)..(n+k)$. This means that $B_{e_i} = 0$ for $i \in (n+1)..(n+k)$.

We can now suppose $\lambda \in \mathrm{GL}_\circ$ and obtain the constrain that $AB_{y\lambda} = B_y B$ and so is linear for an operation $\circ$.
This constrain can be seen as a matrix linear system that has as equations:

1. *for all $i \in 1..n$:*
$$B_{e_i} B = AB_{e_i\lambda} = A\left(\sum_{i=1}^{n+k} c_i B_{e_i}\right)$$
where c_i is the i-th component of $e_i\lambda = c_1 e_1 \circ \cdots \circ c_{n+k} e_{n+k}$.
From the fact that $B_{e_i} = 0$ for $i \in (n+1)..(n+k)$, we can rewrite the equation to
$$B_{e_i} B = AB_{e_i\lambda} = A\left(\sum_{i=1}^{n} c_i B_{e_i}\right)$$
We can note that from the Algorithm 2.4.1, we have that the first n c_i's in the combination with respect to the operation $\circ$ are the same with respect to the operation $+$.

2. *for all $i \in 1..(n+k)$:*
$$e_i B_{e_i} = 0$$

3. *for all $i \in 1..(n+k)$ and $j \in 1..(n+k)$:*
$$e_j B_{e_i} = e_i B_{e_j}$$

The solutions $\{B_{e_i}\}_{i \in 1..n} \cup \{B_{e_i} = 0\}_{i \in (n+1)..(n+k)}$ can be used to generate the operation $\circ$.

If exist a solution different from all the matrices be 0*, we have an operation* $\circ$ *that is different form* $+$*.*

Note 4.2.1. Since we are searching for all the possible operations $\circ$ that linearise λ, we can find operations $\circ$ such that $\dim(U(T_\circ)) \geq k$.

So, the algorithm can be summarized as:

1. *Calculate* $\{e_i\lambda\}_{i\in 1..n} = \{v_1, \dots, v_n\}$.
2. *Resolve the matrix system, for all* $i \in 1..n$

$$B_{e_i}B = AB_{v_i} = A\left(\sum_{j\in 1..n} c_j B_{e_j}\right)$$

3. *Check that the solution is such that* $e_iB_{e_i} = 0$ *and* $e_iB_{e_j} = e_jB_{e_i}$ *for all* $i \in 1..n$ *and* $j \in 1..n$.
4. *The solution can be used in the Algorithm 4.1.1 to generate the operation* $\circ$

Proposition 4.2.2. *The SearchLinearity algorithm resolve the Problem 4.2.1.*

Proof. It follows directly from the construction made in the algorithm.
We search the possible B_v's that linearise λ and so impose to the B_{e_i}'s the constrains that the blocks of λ need to have to be in $\mathrm{GL}_+ \cap \mathrm{GL}_\circ$. □

Remark 4.2.2.
To optimize the algorithm, the matrix linear system can be seen as a linear system just by *unrolling* the matrices:

$$\begin{pmatrix} b_{1,1} & \cdots & b_{1,k} \\ \vdots & \ddots & \vdots \\ b_{n,1} & \cdots & b_{n,k} \end{pmatrix} \longleftrightarrow \Big(\overbrace{b_{1,1}, \cdots, b_{1,k}}^{\text{1-st row}}, \overbrace{b_{2,1}, \cdots, b_{2,k}}^{\text{2-nd row}}, \cdots, \overbrace{b_{n,1}, \cdots, b_{n,k}}^{\text{n-th row}}\Big)$$

With this:

- Resolve the matrix system for all $j \in 1..n$, $\sum_{i \in v_j} AB_{e_i} = B_{e_j}B$ is equivalent to $n^2 \cdot k$ linear equation in $n \cdot (n-1) \cdot k$ variables.
- Check that the solution are in the form $e_i B_{e_i} = 0$ and $e_i B_{e_j} = e_j B_{e_i}$ is equivalent to $\binom{n+1}{2} k$ linear equations.

And so we only need to find a solution of a $(n^2k + \binom{n+1}{2}k) \times n^2k$ matrix.

Remark 4.2.3. The solutions $\mathcal{B} = \{\{B_{e_i}\}_{i \in 1..n}\}$ that we obtain from the algorithm are a subspace of $\left(\mathbb{F}^{n \times k}\right)^n$. It will be generated by a basis $\{\mathcal{B}_j\}_{j \in 1..l}$ with some dimension l that depends on λ.

Example 4.2.1. Consider $\lambda \in \mathrm{GL}_+(3)$. We fix $n = 3$ and $k = 1$.

$$\lambda = \begin{pmatrix} 1 & 0 & 1 \\ 1 & 1 & 1 \\ 0 & 0 & 1 \end{pmatrix}$$

as permutation

$$\lambda : ((1,0,0)\ (1,0,1)) \quad ((0,1,0)\ (1,1,1)\ (0,1,1)\ (1,1,0))$$

And so

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

For a better reading experience, we will denote the B_{e_i} as row vector even if they are intended as column vectors and with I_n the identity matrix of dimension n.

We have that $B_{e_3} = (0,0)$ and $e_3\lambda = e_3$.

$$B_{e_1} = B_{e_1}B = A(B_{e_1} + B_{e_3}) = AB_{e_1} \Longrightarrow (A + I_2)B_{e_1} = 0 \to B_{e_1} = \{(0,0),(0,1)\}$$

$$B_{e_2}B = A(B_{e_1} + B_{e_2} + B_{e_3}) \Longrightarrow (A + I_2)B_{e_2} = AB_{e_1}$$

and so we have the cases

$$\begin{cases} B_{e_1} = (0,0) \Longrightarrow B_{e_2} = \{(0,0),(0,1)\} \\ B_{e_1} = (0,1) \Longrightarrow B_{e_1} = \{(1,0)\} \end{cases}$$

Now, if we consider $B_{e_1} = (0,0)$, the only acceptable solution is $B_{e_2} = (0,0)$. In this case $B_{e_1} = B_{e_2} = B_{e_3} = (0,0)$ and so we get that the operation $\circ$ is equivalent to $+$.

On the other hand, if we consider $B_{e_1} = (0,1), B_{e_2} = (1,0), B_{e_3} = (0,0)$, we get a operation $\circ$ different from $+$.

As an example that the operation $\circ$ is not the standard XOR $+$, consider

$$\begin{aligned}
(1,0,0) \circ (0,1,0) &= (1,0,0) \cdot \left(\begin{pmatrix} I_2 & B_{e_2}^T \\ 0 & 1 \end{pmatrix} \right) + (0,1,0) \\
&= (1,0,0) \cdot \left(\begin{pmatrix} 1 & 0 & 1 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \right) + (0,1,0) \\
&= (1,0,1) + (0,1,0) \\
&= (1,1,1) \\
&\neq (1,1,0) = (1,0,0) + (0,1,0)
\end{aligned}$$

and an example that shows that the operation $\circ$ linearize λ

$$\begin{aligned}
(0,1,1) = (1,1,1)\lambda &= ((1,0,0) \circ (0,1,0))\lambda \\
&= (1,0,0)\lambda \circ (0,1,0)\lambda \\
&= (1,0,1) \circ (1,1,1) \\
&= (1,0,1) \cdot \left(\begin{pmatrix} 1 & 0 & 1 \\ 0 & 1 & 1 \\ 0 & 0 & 1 \end{pmatrix} \right) + (1,1,1) \\
&= (1,0,0) + (1,1,1) \\
&= (0,1,1)
\end{aligned}$$

4.2.2 Computational Complexity

As the problem is reduced to solve a linear system, we will compute the computational complexity as a metric to compare and observe the possible real world application.

Proposition 4.2.3. *Let $n+k = \dim V$ where $k = \dim(\mathrm{Rk}(B))$ and $n = \dim(\mathrm{Rk}(A))$. Assume the complexity to be :*

- *applying λ or λ^{-1} has cost* 1[1]
- *checking if a number is zero has cost* 1
- *inserting a single value in a matrix has cost* 1
- *solving a linear system in n variable and m equation, has cost* $\min(m,n) \cdot nm^2$
- *the output is not saved in memory*

Then the time complexity is $\mathcal{O}\left(n^6k^3\right)$ and the space complexity is $\mathcal{O}\left(l \cdot 2^{k-1}n^2\right)$ where l is the dimension of the solution subspace.

Proof.
Following the algorithm's steps:

- Calculating $\{e_i\lambda\}_{i\in n+1..n+k} = \{v_1, \ldots, v_k\}$ has cost k
- Generating the matrix needs $n^2k \cdot \binom{n+1}{2}k$ operation, if we naïvely fill it
- Resolve the matrix system is equivalent to a system $2n^2 \cdot k \times n^2k$ variables. So it has a time complexity of $n^2k \cdot n^2k \cdot n^2k = n^6k^3$. It will generate at least a solution with space complexity $\frac{n(n-1)}{2}2^k$. Let l be the dimension of the solution subspace.

So we have

$$\text{TIME} : \mathcal{O}(k + n^3k^2(n+1) + n^6k^3) \sim \mathcal{O}\left(n^6k^3\right) \qquad \text{SPACE} : \mathcal{O}(l \cdot 2^{k-1}n(n-1))$$

So considering a medium worst case in which $k \sim n$

$$\text{TIME} : \mathcal{O}\left(n^9\right) \qquad \text{SPACE} : \mathcal{O}\left(l \cdot 2^{\frac{n}{2}-1}n^2\right)$$

□

A top performance PC[3] with a computational capability of $4.64 \cdot 10^{12} \sim 2^{38}$ flops/s

[1] We can consider λ as a look-up table in memory or as a hardware implementation.
[2] Using Gauss reduction. It can be lowered using a sparse-matrix.
[3] For a real hardware, the Radeon HD 5000 Series has the computational capability described.

Size of the block cipher	$n \sim k$	in bit	Time in second
64	32	5	2^7 : 2 minutes
128	64	6	2^{16} : 270 days
256	128	7	2^{25} : 32 years

Table 4.1: theoretical computation time for a Radeon HD 5000. Case $n \sim k$.

In the worst case ($k \sim n$), the cost of search an operation $\circ$ is $\mathcal{O}(n^9)$. In the Table 4.1 are reported some theoretical time for the standard block cipher size in modern cryptography.

In the case where the dimension k is smaller than n ($k << n$), the cost of search an operation $\circ$ is $\mathcal{O}(n^6)$.In the Table 4.2 are reported some theoretical time for the standard block cipher size in modern cryptography.

Size of the block cipher	$n \sim$	in bit	Time in second
64	64	6	2^{-2} : 0.25 seconds
128	128	7	2^4 : 16 seconds
256	256	8	2^{10} : 17 minutes

Table 4.2: theoretical computation time for a Radeon HD 5000. Case $n \gg k$.

Example 4.2.2. It is important to observe that the possibility to translate the problem in finding a solution to a linear system, has a huge impact on the performance.

We compared the algorithm with a different and more naïve algorithm that

1. Generate O as described in Section 3.2
2. For every operation $u \in O$:
   Filter the set O and let $O' = \{u \in O | AB_{e_i\lambda} = B_{e_i}B\}$ where the B_{e_i} are obtained by u.
3. Return O'

We obtained, for the naïve algorithm, the computational time reported in the Table 4.3 where

- **Partial time** indicates the seconds needed to construct the operations space O to filter

- **Time** represent the total time of the algorithm
- **Difference** indicates the effective time that was used by the algorithm to search in the operations space O

n	k	Number of operation	Partial time	Time	Difference
2	1	2	0.000	0.000	0.000
2	2	1	0.000	0.000	0.000
2	3	2	0.000	0.000	0.000
2	4	1	0.010	0.010	0.000
2	5	2	0.000	0.000	0.000
2	6	2	0.040	0.040	0.000
2	7	2	0.120	0.120	0.000
2	8	2	0.460	0.460	0.000
2	9	1	1.760	1.760	0.000
2	10	4	7.180	7.180	0.000
2	11	1	30.800	30.820	0.020
3	1	1	0.000	0.000	0.000
3	2	4	0.010	0.010	0.000
3	3	8	0.890	0.890	0.000
3	4	1	53.050	53.090	0.040
4	1	4	0.020	0.020	0.000
4	2	16	63.210	63.260	0.050

Table 4.3: experimental computational time for the naïve algorithm.

We plotted the results of the SearchLinearity algorithm.
As we can observe in Figure 4.1, when we consider $n \sim k$, the time complexity increase really fast.

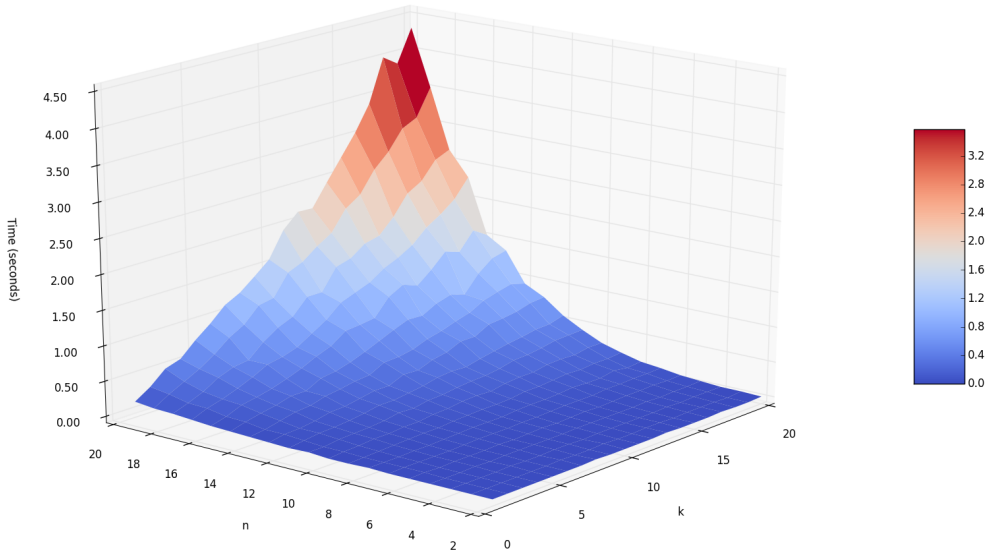

Figure 4.1: Graph of the time to search the operations $\circ$ that linearize λ with fixed n and k.

4.3 PRESENT's mixing layer

PRESENT (see [BKL+07]) is a lightweight block cipher, developed by the Orange Labs, Ruhr University Bochum and the Technical University of Denmark and presented at the Cryptographic Hardware and Embedded Systems of 2007. It is designed to respond to some critical environment where AES was unsuitable (quoting the content of the presented paper):

1. The cipher is to be implemented in hardware in a easy way
2. Application will require only a moderate security levels.
   They considered 80-bit security adequate.
3. Applications are unlikely to require the encryption of large amounts of data.
   Implementations might therefore be optimised for performance or for space without too much practical impact.
4. In some applications it is possible that the key will be fixed at the time of device manufacture.
   In such cases there would be no need to re-key a device. This means that the cipher should consider *key manipulation attacks*.
5. After security, the physical space required for an implementation will be the primary consideration.
   This is closely followed by peak and average power consumption, with the timing requirements being a third important metric.
6. In applications that demand the most efficient use of space, the block cipher will often only be implemented as encryption-only.

Taking these consideration in account, the authors choosed to develop a block cipher with a 64-bit block and a 80-bit key.

The International Organization for Standardization and the International Electrotechnical Commission included PRESENT in the new international standard for lightweight cryptographic methods (see [ISO12]).

4.3.1 Algebraic description and properties

We can define PRESENT as an iterated, translation based, block cipher

$$\text{PRESENT} : \mathcal{M} \times \mathcal{K} \longrightarrow \mathcal{M}$$

with

- Message space : $\mathcal{M} = \mathbb{F}^{64}$
- Key space : $\mathcal{K} = \mathbb{F}^{80}$
- Number of rounds : $r = 31$
- Key scheduler : as a function that given a key $k \in \mathcal{K}$, it returns 32 ordered k_i's round-keys with $i \in 1..32$.
  Let 32-th round-key k_0 is used to pre-whitening the result.
  We denote with σ_{k_i} the translation $x\sigma_{k_i} = x + k_i$.
- Permutation box : PRESENT is constructed on 16 parallel 4-bits permutation box γ.

$$\gamma : \mathbb{F}^4 \longrightarrow \mathbb{F}^4$$

$$(x_1, x_2, x_3, x_4)\gamma = \begin{pmatrix} x_1x_2x_4 + x_1x_3x_4 + x_1 + x_2x_3x_4 + x_2x_3 + x_3 + x_4 + 1 \\ x_1x_2x_4 + x_1x_3x_4 + x_1x_3 + x_1x_4 + x_1 + x_2 + x_3x_4 + 1 \\ x_1x_2x_4 + x_1x_2 + x_1x_3x_4 + x_1x_3 + x_1 + x_2x_3x_4 + x_3 \\ x_1 + x_2x_3 + x_2 + x_4 \end{pmatrix}$$

- Mixing Layer : PRESENT has a mixing layer λ

$$\lambda : \mathbb{F}^{64} \longrightarrow \mathbb{F}^{64}$$

λ can be seen as a bit-permutation that changes the bit-positions.

$$e_i \xrightarrow{\lambda} e_j \Rightarrow \left(\begin{array}{cccc|cccc} 1 \to 1 & 2 \to 17 & 3 \to 33 & 4 \to 49 & 5 \to 2 & 6 \to 18 & 7 \to 34 & 8 \to 50 \\ 9 \to 3 & 10 \to 19 & 11 \to 35 & 12 \to 51 & 13 \to 4 & 14 \to 20 & 15 \to 36 & 16 \to 52 \\ 17 \to 5 & 18 \to 21 & 19 \to 37 & 20 \to 53 & 21 \to 6 & 22 \to 22 & 23 \to 38 & 24 \to 54 \\ 25 \to 7 & 26 \to 23 & 27 \to 39 & 28 \to 55 & 29 \to 8 & 30 \to 24 & 31 \to 40 & 32 \to 56 \\ \hline 33 \to 9 & 34 \to 25 & 35 \to 41 & 36 \to 57 & 37 \to 10 & 38 \to 26 & 39 \to 42 & 40 \to 58 \\ 41 \to 11 & 42 \to 27 & 43 \to 43 & 44 \to 59 & 45 \to 12 & 46 \to 28 & 47 \to 44 & 48 \to 60 \\ 49 \to 13 & 50 \to 29 & 51 \to 45 & 52 \to 61 & 53 \to 14 & 54 \to 30 & 55 \to 46 & 56 \to 62 \\ 57 \to 15 & 58 \to 31 & 59 \to 47 & 60 \to 63 & 61 \to 16 & 62 \to 32 & 63 \to 48 & 64 \to 64 \end{array}\right)$$

The single i-th round of PRESENT can be represented with $\gamma\lambda\sigma_{k_i}$ where γ and λ are round independent and σ_{k_i} is the i-th round key obtained by the key scheduler. All the 31 round have this form except for the 0-th round which is a pre-whitening where the block cipher just sum the k_0 key on the plaintext and it can be represented with σ_{k_0}.

For this reason, PRESENT is a iterated block cipher and it can be represented as

$$\text{PRESENT} = \sigma_{k_0}\left(\prod_{i=1}^{31}\gamma\lambda\sigma_{k_i}\right)$$

We can now state some properties for λ and γ:

1. γ is 4-differential uniform
2. γ is weakly 4-differential uniform
3. γ is not anti-crocked
4. λ is such that $\lambda^3 = 1$
5. λ has 4 fixed points, λ fixes the bits in position $\{1, 22, 43, 64\}$

For this reasons, the hypothesis of the Theorem 2.3.2 are not achieved and so it may be present a hidden sum into the PRESENT block cipher.

4.3.2 Hidden sum on PRESENT's mixing layer

We now consider λ, the PRESENT's Mixing Layer, and search the operations that will linearize it.
To permit the research using the algorithm presented in the thesis, we have to do some preparation.

To use the Algorithm 4.2.1, we need to have a linear function in the block form:

$$\lambda = \begin{pmatrix} A & D \\ 0 & B \end{pmatrix} \qquad \begin{cases} A \in \mathrm{GL}_+ \\ B \in \mathrm{GL}_+ \end{cases}$$

For this reason we consider the permutation

$$\pi = (e_1 e_{61})(e_{22} e_{62})(e_{43} e_{63}) \in \mathrm{Sym}(\{e_i | i \in 1..64\})$$

such that the conjugation of λ with π transform λ in a algorithm's accepted matrix.

$$\pi\lambda\pi^{-1} = \begin{pmatrix} A & 0 \\ 0 & I_4 \end{pmatrix} = \hat{\lambda}$$

In a general case, we need to have that $\pi \in \mathrm{GL}_+(n+k)$.

We can now apply the Algorithm 4.2.1 and obtain all the possible operations that linearize $\hat{\lambda}$. The operation space will be denoted with O.
The time to compute the operations space O is ~ 10.420 seconds and it is generated by 2360 60-tuples of 60×4 matrices.
So the number of operations that linearize $\hat{\lambda}$, in the form described in Theorem 2.4.7, are $\#O = 2^{2360}$.

We have to search the operations $\circ$ that linearize λ of PRESENT.
For this reason, consider a $\circ \in O$ obtained from the algorithm. From the fact that $\circ$ linearizes $\hat{\lambda}$, we can obtain a new operation $\triangle$ that linearizes λ just by conjugation

$$T_\circ = \left\langle \begin{array}{c} \tau_{e_1} = k_{e_1}\sigma_{e_1} \\ \tau_{e_2} = k_{e_2}\sigma_{e_2} \\ \vdots \\ \tau_{e_{n+k}} = k_{e_{n+k}}\sigma_{e_{n+k}} \end{array} \right\rangle \xrightarrow{\pi^{-1}(\cdot)\pi} \left\langle \begin{array}{c} M_{v_1}\sigma_{v_1} = \tau_{v_1} \\ M_{v_2}\sigma_{v_2} = \tau_{v_2} \\ \vdots \\ M_{v_{n+k}}\sigma_{v_{n+k}} = \tau_{v_{n+k}} \end{array} \right\rangle = T_\triangle$$

where $v_i = e_i\pi$ and the $M_{v_i} = \pi^{-1}k_{e_i}\pi$ could not be in the form of Theorem 2.4.7.
We have that $\{v_i | i \in 1..(n+k)\}$ form a basis for $\mathbb{F}^{n+k}$.
We have that

$$T_\triangle = \pi^{-1}T_\circ\pi$$

So for every $\tau_v \in T_\circ$ correspond a single $\tau_w \in T_\triangle$.
We can now construct $\tau_v \in T_\triangle$ considering

$$v = (c_1v_1 \triangle \cdots \triangle c_{n+k}v_{n+k}) = \pi^{-1}(c_1e_1 \circ \cdots \circ c_{n+k}e_{n+k})\pi$$

Consider $I = \{i | c_i \neq 0\}$. We have that $\tau_v = M_v\sigma_v$ with

$$M_v = \prod_{i\in I} M_{v_i} = \prod_{i\in I} \pi^{-1}k_{e_i}\pi = \pi^{-1}\left(\prod_{i\in I} k_{e_i}\right)\pi = \pi^{-1}\begin{pmatrix} I_n & \sum_{i\in I} B_{e_i} \\ 0 & I_k \end{pmatrix}\pi$$

We so can construct the operation $\triangle$ that linearizes the PRESENT's λ using the operation $\circ$ and the permutation π.

Example 4.3.1. We have the operation $\circ$ in the form of the Theorem 2.4.7 and the conjugacy map π such that $\pi^{-1}T_\circ\pi = T_\triangle$.

Let $\{v_i | i \in 1..n+k\}$ basis in $\mathbb{F}^{n+k}$ for the operation $\triangle$ where $v_i = e_i\pi$.
Suppose we want to calculate $x \triangle y$. We know, from construction:

$$y\pi^{-1} = \bigcirc_{i \in I_y} e_i \qquad \tau_y = \pi^{-1}\tau_{y\pi^{-1}}\pi$$

We proceed as follows:

$$\begin{aligned}(x \triangle y) = x\tau_y &= x\pi^{-1}\tau_{y\pi^{-1}}\pi \\ &= \left((x\pi^{-1}) \begin{pmatrix} I_n & \sum_{i \in I_y} B_{e_i} \\ 0 & I_k \end{pmatrix} + (y\pi^{-1})\right)\pi \\ &= (x\pi^{-1}\tau_{y\pi^{-1}})\pi = ((x\pi^{-1}) \circ (y\pi^{-1}))\pi\end{aligned}$$

From this, we can see that we only need the operation $\circ$ and the conjugacy map π to construct $\triangle$.

If we consider a linear map $\lambda = \pi^{-1}\hat{\lambda}\pi$ with $\hat{\lambda} \in \mathrm{GL}_\circ$, we have

$$\begin{aligned}(x \triangle y)\lambda &= ((x\pi^{-1}) \circ (y\pi^{-1}))\pi\lambda \\ &= ((x\pi^{-1}) \circ (y\pi^{-1}))\hat{\lambda}\pi \\ &= (x\pi^{-1}\hat{\lambda} \circ y\pi^{-1}\hat{\lambda})\pi = (x\lambda \triangle y\lambda)\end{aligned}$$

and so $\lambda \in \mathrm{GL}_\triangle$

4.3.3 Computational results

From the construction done in the previous subsection, we will describe some algorithms that optimize the use of the operation $\triangle$ obtained by conjugacy π of an operation $\circ$ in the form of the Theorem 2.4.7.
Successively an operation $\triangle$ that linearize the PRESENT's mixing layer λ will be presented.

We will now describe an algorithm to rapidly execute the operation $\triangle$ that is the operation generated by the conjugacy class $T_\triangle = \pi^{-1}T_\circ\pi$.

Algorithm 4.3.1 (Permuted Operation)**.** *Consider an operation* $\circ$ *in the form of the Theorem 2.4.7 and a permutation* $\pi \in \mathrm{GL}_+$.
Let $\{k_{e_i}\}_{i\in 1..(n+k)}$ *the matrices of the transformation* τ_{e_i} *for the canonical basis.*

Let $x, y \in \mathbb{F}^{n+k}$.
To calculate $x \triangle y$*, do:*

1. *Let* $v = x\pi^{-1}$. *Let* $w = y\pi^{-1} = \bigcirc_{i\in I_w} e_i$
2. *Calculate* z *as*
$$z = v \circ w = v(k_w + w) = v\left(\left(\prod_{i\in I_w} k_{e_i}\right) + w\right)$$
3. *Return* $z\pi$

By construction, $z\pi = x \triangle y$.

Note 4.3.1. Let $w = c_1e_1 \circ \cdots \circ c_{n+k}e_{n+k}$ and $w = c'_1e_1 + \cdots + c'_{n+k}e_{n+k}$ be the combination with respect to the operation $\circ$ and $+$.
From the Algorithm 2.4.1, we have

$$c_i = c'_i \qquad \text{for } i \in 1..n$$

and so we have that

$$I_w = \{i | c'_i \neq 0 \text{ and } i \in 1..n\}$$

and from this we get

$$k_w = \prod_{\substack{i \in 1..n \\ i \in I_w}} k_{e_i}$$

Starting from the PRESENT's mixing layer, we consider the canonical basis permutation

$$\pi = (e_1e_{61})(e_{22}e_{62})(e_{43}e_{63}) \in \mathrm{Sym}(\{e_i | i \in 1..64\})$$

We can now consider

$$\hat{\lambda} = \pi\lambda\pi^{-1} = \begin{pmatrix} A & 0 \\ 0 & I_4 \end{pmatrix}$$

We can now use the Algorithm 4.2.1 and obtain the operation space O that contains all the operations $\circ$ such that $\hat{\lambda} \in \mathrm{GL}_\circ$.

We randomly take an operation $\circ$ that is represented in Table 4.4.
The operation has $\mathrm{Rk}\, B_\circ = 60 = n$ and so the operation has $\dim U(T_\circ) = 4 = k$.
To compact the table, we considered every row of B_{e_i} as a number in $0..(2^4 - 1)$. The representation is the same as the one used in Algorithm 4.1.1.

B_{e_1}	0	8	0	10	8	9	7	3	0	9	4	15	1	15	7	15	8	14	7	0	4	5	9	0	15	1	10	10	2	15	6	7	0	5	7	5	10	8	6	6	10	4	8	7	1	2	8	0	11	14	9	11	14	7	13	1	5	3	8	0
B_{e_2}	8	0	7	10	14	7	13	6	7	8	3	12	15	13	11	6	14	15	9	11	14	15	12	6	9	6	0	3	14	8	15	6	10	0	10	4	11	8	3	9	9	0	8	6	7	2	2	8	8	5	14	2	1	6	0	9	12	11	12	8
B_{e_3}	0	7	0	2	10	4	14	5	15	8	12	15	0	15	15	15	7	7	6	7	3	1	12	6	8	15	12	15	9	11	3	14	15	4	2	9	0	7	6	6	15	2	0	9	14	13	8	0	12	7	6	11	5	9	12	13	15	10	1	0
B_{e_4}	10	10	2	0	8	8	2	15	12	15	12	5	7	3	12	12	15	12	8	8	3	14	15	14	14	8	11	9	7	4	10	6	0	6	5	0	9	10	0	1	1	2	12	0	8	3	13	1	7	8	6	8	8	8	4	8	0	2	2	11
B_{e_5}	8	14	10	8	0	15	0	5	7	9	10	14	10	11	4	2	14	14	11	1	7	6	8	6	13	12	3	0	6	6	9	9	7	9	9	12	8	6	0	11	3	0	6	12	12	3	6	8	15	14	7	8	13	8	2	15	11	15	2	8
B_{e_6}	9	7	4	8	15	0	12	15	7	10	10	6	12	0	0	12	14	0	8	8	0	5	13	9	11	7	5	0	12	14	11	5	3	2	15	1	7	4	10	5	7	11	12	11	9	8	2	4	3	0	0	12	15	1	1	4	11	12	7	14
B_{e_7}	7	13	14	2	0	12	0	4	4	7	13	6	6	5	10	10	9	11	4	13	2	3	8	0	11	7	2	6	10	14	10	10	8	11	11	9	2	12	11	13	13	8	11	1	8	10	7	15	14	11	11	13	14	6	9	2	13	1	13	5
B_{e_8}	3	6	5	15	5	15	4	0	7	1	11	3	8	7	6	0	14	12	10	5	0	0	14	6	11	9	14	2	6	0	14	5	9	10	11	0	9	10	14	7	8	8	13	8	6	6	9	2	7	6	15	14	5	11	4	15	4	4	5	14
B_{e_9}	0	7	15	12	7	7	4	7	0	6	2	6	2	7	9	11	10	3	0	5	4	14	7	9	14	12	6	12	5	6	6	13	15	8	15	15	8	15	2	10	12	12	15	1	15	15	9	0	0	9	14	0	15	11	13	1	15	3	8	0
$B_{e_{10}}$	9	8	8	15	9	10	7	1	6	0	6	3	8	7	9	13	11	7	6	8	8	6	12	7	4	9	1	12	10	2	8	1	0	2	11	1	6	4	0	8	5	15	12	2	5	0	12	10	9	9	4	2	15	8	5	5	10	12	6	7
$B_{e_{11}}$	4	3	12	12	10	10	13	11	2	6	0	12	5	2	12	7	9	7	5	0	15	11	5	10	11	9	14	3	11	12	10	3	15	13	12	13	11	2	11	1	12	13	10	13	3	7	8	10	1	8	5	0	13	9	3	9	9	14	0	7
$B_{e_{12}}$	15	12	15	5	14	6	6	3	6	3	12	0	6	10	15	0	7	9	5	13	0	11	9	15	11	7	8	3	15	0	3	9	14	8	3	12	4	6	7	1	5	0	9	9	7	5	0	1	8	6	7	13	13	1	12	12	1	6	11	9
$B_{e_{13}}$	1	15	0	7	10	12	6	8	2	8	5	6	0	8	0	8	8	3	9	8	8	6	10	8	2	15	0	4	15	14	1	8	12	14	1	0	15	8	2	2	12	11	12	2	5	9	0	11	7	7	8	12	3	4	3	14	12	10	13	10
$B_{e_{14}}$	15	13	15	3	11	0	5	7	7	7	2	10	8	0	2	4	1	15	15	4	8	14	10	3	13	4	0	0	5	5	11	0	5	14	13	11	8	12	11	13	0	8	0	0	13	5	14	14	8	5	13	3	4	1	8	5	5	6	13	0
$B_{e_{15}}$	7	11	15	12	4	0	10	6	9	9	12	15	0	2	0	3	12	11	10	5	1	3	3	14	9	0	6	10	0	15	1	0	15	13	9	9	1	12	14	2	13	0	5	4	12	0	1	5	0	4	1	3	11	2	8	8	9	10	4	5
$B_{e_{16}}$	15	6	15	12	2	12	10	0	11	13	7	0	8	4	3	0	8	12	12	0	12	4	4	4	13	4	13	0	14	5	11	11	0	11	10	5	2	0	5	10	0	2	1	4	13	8	8	8	12	13	9	1	3	8	15	6	3	7	12	11
$B_{e_{17}}$	8	14	7	15	14	14	9	14	10	11	9	7	8	1	12	8	0	7	8	13	15	9	6	8	0	8	0	2	5	6	11	15	7	13	3	11	9	12	0	15	10	3	2	2	14	0	12	8	10	6	12	6	11	6	3	6	4	9	6	8
$B_{e_{18}}$	14	15	7	12	14	0	11	12	3	7	7	9	3	15	11	12	7	0	10	0	0	4	7	14	2	4	11	8	0	1	12	5	4	12	10	0	8	13	5	11	15	10	12	2	0	1	7	9	8	15	6	12	8	9	0	5	1	5	11	4
$B_{e_{19}}$	7	9	6	8	11	8	4	10	0	6	5	5	9	15	10	12	8	10	0	7	7	5	9	2	2	4	15	0	9	8	12	6	8	7	6	9	6	12	1	8	11	0	2	12	4	5	6	9	15	1	3	13	8	7	12	1	1	8	2	10
$B_{e_{20}}$	0	11	7	8	1	8	13	5	5	8	0	13	8	4	5	0	13	0	7	0	15	5	4	5	14	12	8	5	5	1	6	14	15	5	2	2	15	10	0	11	13	11	3	14	13	8	13	15	3	7	10	4	4	3	0	0	11	13	0	14
$B_{e_{21}}$	4	14	3	3	7	0	2	0	4	8	15	0	8	8	1	12	15	0	7	15	0	5	4	1	12	13	10	1	15	9	5	4	7	11	7	11	10	7	11	12	10	5	12	7	6	0	11	14	12	12	9	12	0	14	8	5	0	11	2	9
$B_{e_{22}}$	5	15	1	14	6	5	3	0	14	6	11	11	6	14	3	4	9	4	5	5	5	0	10	9	10	13	3	10	5	8	2	1	13	2	9	8	1	1	1	1	3	12	11	7	9	8	11	1	8	15	12	6	0	0	13	1	0	0	0	7
$B_{e_{23}}$	9	12	12	15	8	13	8	14	7	12	5	9	10	10	3	4	6	7	9	4	4	10	0	4	12	12	1	9	10	11	1	13	15	7	9	0	4	12	6	1	2	4	0	2	6	9	7	1	8	9	7	4	12	13	12	1	12	15	0	8
$B_{e_{24}}$	0	6	6	14	6	9	0	6	9	7	10	15	8	3	14	4	8	14	2	5	1	9	4	0	6	13	3	13	11	12	9	8	11	14	12	15	8	11	9	12	9	4	8	1	1	9	9	15	4	0	0	5	1	12	9	0	2	7	2	7
$B_{e_{25}}$	15	9	8	14	13	11	11	11	14	4	11	11	2	13	9	13	0	2	2	14	12	10	12	6	0	8	11	9	4	0	13	2	4	11	13	13	7	7	8	1	13	2	10	13	6	6	1	5	6	10	8	11	5	14	10	3	10	10	7	7
$B_{e_{26}}$	1	6	15	8	12	7	7	9	12	9	9	7	15	4	0	4	8	4	4	12	13	13	12	13	8	0	6	12	14	4	1	1	7	12	2	12	12	12	4	15	5	1	4	0	9	9	2	8	10	10	6	0	10	11	9	10	3	1	7	9
$B_{e_{27}}$	10	0	12	11	3	5	2	14	6	1	14	8	0	0	6	13	0	11	15	8	10	3	1	3	11	6	0	5	14	9	4	12	2	8	13	0	0	4	9	7	11	9	5	0	7	13	13	4	2	8	0	2	11	4	10	1	14	2	5	6
$B_{e_{28}}$	10	3	15	9	0	0	6	2	12	12	3	3	4	0	10	0	2	8	0	5	1	10	9	13	9	12	5	0	4	9	1	8	13	10	7	0	5	9	13	5	3	10	15	4	12	5	3	2	3	6	5	8	8	9	5	13	8	7	8	13
$B_{e_{29}}$	2	14	9	7	6	12	10	6	5	10	11	15	15	5	0	14	5	0	9	5	15	5	10	11	4	14	14	4	0	6	7	15	7	11	8	4	1	9	8	4	11	14	0	5	3	2	8	14	8	6	6	13	7	0	6	0	6	14	9	3
$B_{e_{30}}$	15	8	11	4	6	14	14	0	6	2	12	0	14	5	15	5	6	1	8	1	9	8	11	12	0	4	9	9	6	0	12	0	9	6	9	2	7	13	4	7	10	3	4	2	15	13	1	7	8	11	1	8	3	12	9	9	14	9	9	0
$B_{e_{31}}$	6	15	3	10	9	11	10	14	6	8	10	3	1	11	1	11	11	12	12	6	5	2	1	9	13	1	4	1	7	12	0	0	10	1	14	10	8	15	2	4	1	7	10	4	1	5	4	3	2	4	6	7	13	7	7	1	2	4	0	6
$B_{e_{32}}$	7	6	14	6	9	5	10	5	13	1	3	9	8	0	0	11	15	5	6	14	4	1	13	8	2	1	12	8	15	0	0	0	1	3	11	3	5	10	3	2	9	1	6	11	12	13	0	5	14	0	11	4	5	9	10	1	8	1	7	1
$B_{e_{33}}$	0	10	15	0	7	3	8	9	15	0	15	14	12	5	15	0	7	4	8	15	7	13	15	11	4	7	2	13	7	9	10	1	0	14	12	15	6	12	12	3	2	6	11	8	6	12	1	0	2	5	15	15	7	6	15	14	9	6	9	0
$B_{e_{34}}$	5	0	4	6	9	2	11	10	8	2	13	8	14	14	13	11	13	12	7	5	11	2	7	14	11	12	8	10	11	6	1	3	14	0	13	10	4	8	2	10	11	11	13	7	11	9	13	7	2	4	6	10	13	0	6	10	9	13	1	15
$B_{e_{35}}$	7	10	2	5	9	15	11	11	15	11	12	3	1	13	9	10	3	10	6	2	7	9	9	12	13	2	13	7	8	9	14	11	12	13	0	12	5	5	14	10	12	11	0	8	5	3	0	4	12	11	12	7	0	10	3	3	13	1	13	10
$B_{e_{36}}$	5	4	9	0	12	1	9	0	15	1	13	12	0	11	9	5	11	0	9	2	11	8	0	15	13	12	0	0	4	2	10	3	15	10	12	0	10	3	6	1	9	14	3	1	1	8	4	7	12	6	15	3	5	14	10	0	9	2	4	5
$B_{e_{37}}$	10	11	0	9	8	7	2	9	8	6	11	4	15	8	1	2	9	8	6	15	10	1	4	8	7	12	0	5	1	7	8	5	6	4	5	10	0	9	15	12	6	1	13	6	3	12	2	7	8	10	5	12	7	2	0	6	9	8	12	9
$B_{e_{38}}$	8	8	7	10	6	4	12	10	15	4	2	6	8	12	12	0	12	13	12	10	7	1	12	11	7	12	4	9	9	13	15	10	12	8	5	3	9	0	1	1	9	6	4	7	7	12	0	9	15	14	9	4	4	4	9	13	0	1	2	1
$B_{e_{39}}$	6	3	6	0	0	10	11	14	2	0	11	7	2	11	14	5	0	5	1	0	11	1	6	9	8	4	9	13	8	4	2	3	12	2	14	6	15	1	0	4	13	9	2	13	0	10	5	10	11	14	8	13	8	3	5	12	0	7	0	4
$B_{e_{40}}$	6	9	6	1	11	5	13	7	10	8	1	1	2	13	2	10	15	11	8	11	12	1	1	12	1	15	7	5	4	7	4	2	3	10	10	1	12	1	4	0	14	2	7	4	6	7	0	6	10	14	3	11	6	9	1	0	10	4	4	3
$B_{e_{41}}$	10	9	15	1	3	7	13	8	12	5	12	5	12	0	13	0	10	15	11	13	10	3	2	9	13	5	11	3	11	10	1	9	2	11	12	9	6	9	13	14	0	14	7	0	12	3	13	7	5	11	3	10	2	12	7	11	12	10	8	4
$B_{e_{42}}$	4	0	2	2	0	11	8	8	12	15	13	0	11	8	0	2	3	10	0	11	5	12	4	4	2	1	9	10	14	3	7	1	6	11	11	14	1	6	9	2	14	0	13	5	8	5	0	6	0	14	7	5	0	9	13	3	6	4	13	10
$B_{e_{43}}$	8	8	0	12	6	12	11	13	15	12	10	9	12	0	5	1	2	12	2	3	12	11	0	8	10	4	5	15	0	4	10	6	11	13	0	3	13	4	2	7	7	13	0	12	0	0	4	11	8	14	13	1	4	5	8	4	3	11	8	15
$B_{e_{44}}$	7	6	9	0	12	11	1	8	1	2	13	9	2	0	4	4	2	2	12	14	7	7	2	1	13	0	0	4	5	2	4	11	8	7	8	1	6	7	13	4	0	5	12	0	11	8	8	8	13	9	0	8	13	9	3	0	4	0	8	8
$B_{e_{45}}$	1	7	14	8	12	9	8	6	15	5	3	7	5	13	12	13	14	0	4	13	6	9	6	1	6	9	7	12	3	15	1	12	6	11	5	1	3	7	0	6	12	8	0	11	0	3	9	9	6	15	7	9	10	0	5	11	15	3	0	15
$B_{e_{46}}$	2	2	13	3	3	8	10	6	15	0	7	5	9	5	0	8	0	1	5	8	0	8	9	9	6	9	13	5	2	13	5	13	12	9	3	8	12	12	10	7	3	5	0	8	3	0	4	13	4	4	12	15	0	9	5	10	10	1	3	10
$B_{e_{47}}$	8	2	8	13	6	2	7	9	9	12	8	0	0	14	1	8	12	7	6	13	11	11	7	9	1	2	13	3	8	1	4	0	1	13	0	4	2	0	5	0	13	0	4	8	9	4	0	8	2	5	11	12	0	2	8	7	4	4	8	7
$B_{e_{48}}$	0	8	0	1	8	4	15	2	0	10	10	1	11	14	5	8	8	9	9	15	14	1	1	15	5	8	4	2	14	7	3	5	0	7	4	7	7	9	10	6	7	6	11	8	9	13	8	0	10	3	15	15	0	0	10	7	5	6	7	0
$B_{e_{49}}$	11	8	12	7	15	3	14	7	0	9	1	8	7	8	0	12	10	8	15	3	12	8	8	4	6	10	2	3	8	8	2	14	2	2	12	12	8	15	11	10	5	0	8	13	6	4	2	10	0	15	5	12	8	14	9	6	0	1	0	1
$B_{e_{50}}$	14	5	7	8	14	0	11	6	9	9	8	6	7	5	4	13	6	15	1	7	12	15	9	0	10	10	8	6	6	11	4	0	5	4	11	6	10	14	14	14	11	14	14	9	15	4	5	3	15	0	3	0	5	6	2	5	0	7	8	2
$B_{e_{51}}$	9	14	6	6	7	0	11	15	14	4	5	7	8	13	1	9	12	6	3	10	9	12	7	0	8	6	0	5	6	1	6	11	15	6	12	15	5	9	8	3	3	7	13	0	7	12	11	15	5	3	0	0	13	15	3	9	12	1	9	1
$B_{e_{52}}$	11	2	11	8	8	12	13	14	0	2	0	13	12	3	3	1	6	12	13	4	12	6	4	5	11	0	2	8	13	8	7	4	15	10	7	3	12	4	13	11	10	5	1	8	9	15	12	15	12	0	0	0	0	4	0	11	5	10	4	8
$B_{e_{53}}$	14	1	5	8	13	15	14	5	15	15	13	13	3	4	11	3	11	8	8	4	0	0	12	1	5	10	11	8	7	3	13	5	7	13	0	5	7	4	8	6	2	0	4	13	10	0	0	0	8	5	13	0	0	5	5	14	2	11	14	15
$B_{e_{54}}$	7	6	9	8	8	1	6	11	11	8	9	1	4	1	2	8	6	9	7	3	14	0	13	12	14	11	4	9	0	12	7	9	6	0	10	14	2	4	3	9	12	9	5	9	0	9	2	0	14	6	15	4	5	0	13	8	15	12	1	15
$B_{e_{55}}$	13	0	12	4	2	1	9	4	13	5	3	12	3	8	8	15	3	0	12	0	8	13	12	9	10	9	10	5	6	9	7	10	15	6	3	10	0	9	5	1	7	13	8	3	5	5	8	10	9	2	3	0	5	13	0	8	0	5	4	2
$B_{e_{56}}$	1	9	13	8	15	4	2	15	1	5	9	12	14	5	8	6	6	5	1	0	5	1	1	0	3	10	1	13	0	9	1	1	14	10	3	0	6	13	12	0	11	3	4	0	11	10	7	7	6	5	9	11	14	8	8	0	3	2	11	5
$B_{e_{57}}$	5	12	15	0	11	11	13	4	15	10	9	1	12	5	9	3	4	1	1	11	0	0	12	2	10	3	14	8	6	14	2	8	9	9	13	9	9	0	0	10	12	6	3	4	15	10	4	5	0	0	12	5	2	15	0	3	0	1	1	7
$B_{e_{58}}$	3	11	10	2	15	12	1	4	3	12	14	6	10	6	10	7	9	5	8	13	11	0	15	7	10	1	2	7	14	9	4	1	6	13	1	2	8	1	7	4	10	4	11	0	3	1	4	6	1	7	1	10	11	12	5	2	1	0	4	6
$B_{e_{59}}$	8	12	1	2	2	7	13	5	8	6	0	11	13	13	4	12	6	11	2	0	2	0	0	2	7	7	5	8	9	9	0	7	9	1	13	4	12	2	0	4	8	13	8	8	0	3	8	7	0	8	9	4	14	1	4	11	1	4	0	8
$B_{e_{60}}$	0	8	0	11	8	14	5	14	0	7	7	9	10	0	5	11	8	4	10	14	9	7	8	7	7	9	6	13	3	0	6	1	0	15	10	5	9	1	4	3	4	10	15	8	15	10	7	0	1	2	1	8	15	15	2	5	7	6	8	0

Table 4.4: B_{e_i}'s of a random operation $\circ$ that linearize PRESENT's mixing layer λ

Let $x, y \in \mathbb{F}^{64}$ and consider π such that

$$T_{\triangle} = \pi^{-1} T_{\circ} \pi$$

We check that the operation $\circ$ linearize λ by constructing different Magma functions.
To do it, we calculate the two values

$$x\lambda \triangle y\lambda \qquad (x \triangle y)\lambda$$

and check their equality. To execute the operation $\triangle$, we do

$$x \triangle y = ((x\pi^{-1}) \circ (y\pi^{-1}))\pi$$

We so can write the function that will execute the operation $\triangle$ by giving in input the operation $\circ$ and the permutation π.

```
1
2  // Code to execute the operations:
3  //
4  // Create the matrix M_ei
5
6  function getB(x,op)
7    n := #Eltseq(x);
8    B := IdentityMatrix(GF(2),n);
9    for i in [1..#Eltseq(x)] do
10     if x[i] eq 1 then
11       B := B * op[i];
12     end if;
13   end for;
14   return B;
15 end function;
16
17
18 // Execute
19 // x*M_y + y
20
21 function oS(x,y,op)
22   return x * getB(y,op) + y;
23 end function;
24
25
26 // Execute
27 // (x*pi^-1 o y*pi^-1)*pi = x triangle y
28 // If pi=Id --> it execute (x o y)
29
30 function Op(x,y,op,pi)
31   return oS(x*pi^-1 , y*pi^-1,op)*pi;
32 end function;
```

We choose a random operation (the one represented in Table 4.4) and execute 2^{15} random selection for x, y and check the equality.

```
1
2
3  // Taking a Random operation and create the matrices M_ei
4  // Let a the operation space
5
6  op := solToBei(Random(a),n,k);
7  opp := BeiToMei(op,n,k) cat [IdentityMatrix(GF(2),n+k) : i in [1..k]];
8
9  // Check if correct
10 // Create the vectorspace GF(2) of dimension 64
11
12 V := VectorSpace(GF(2),n+k);
13
14 // Test 2^15 random x,y
15
16 t := Cputime();
17 for i in [1..2^15] do
18   x := Random(V); y := Random(V);
19
20   // Let M as the permutation matrix from o to triangle
21
22   // Check
23   // x*lambda triangle y*lambda == (x triangle y)*lambda
24
25   if not(Op(x*presentML , y*presentML , opp, M) eq Op(x,y,opp,M)*presentML) then
26     print x,y, " - Error";
27   end if;
28 end for;
29 Cputime(t);
```

We executed 50 tests and there were no errors.
The mean test time to check 2^{15} random x, y was ~ 13.150 seconds.

Example 4.3.2. We will report a single numerical example to check that the operation $\triangle$ linearize PRESENT's λ.

Let $x = e_2$ and $y = e_3$.

We have that

$$x\pi^{-1} = e_2 \qquad y\pi^{-1} = e_3$$

Now, from Table 4.4, we have

$$e_2 \circ e_3 = (e_2 + e_3) + (e_{61} + e_{62} + e_{63})$$

We now apply π to the result

$$((e_2 + e_3) + (e_{61} + e_{62} + e_{63}))\pi = ((e_2 + e_3) + (e_1 + e_{22} + e_{43}))$$

We now apply λ and get

$$((e_2 + e_3) + (e_1 + e_{22} + e_{43}))\lambda = ((e_{17} + e_{33}) + (e_1 + e_{22} + e_{43}))$$

We now calculate

$$e_2\lambda = e_{17} \qquad e_3\lambda = e_{33}$$

and then

$$e_{17} \triangle e_{33} = (e_{17}\pi^{-1} \circ e_{33}\pi^{-1})\pi = (e_{17} \circ e_{33})\pi$$

$$(e_{17} + e_{33} + (e_{61} + e_{62} + e_{63}))\pi = ((e_{17} + e_{33}) + (e_1 + e_{22} + e_{43}))$$

We have the equality and so for $x = e_2$ and $y = e_3$, we have that

$$(x \triangle y)\lambda = x\lambda \triangle y\lambda$$

We analysed the operation space O and tried to check the rank of the different operation that can be founded.
To do it, we build a Magma script that randomly chooses an operation $\circ$, check $\operatorname{Rk} B_\circ$ and save the dimension obtained in a list.

```
1
2
3  // Create B_o and return the rank
4
5  function rankOpK(lisBei,n,k)
6    return Rank(Matrix(GF(2),n*k,n,&cat[Eltseq(i) : i in lisBei]));
7  end function;
8
9
10 // Initialize the ranks list and execute the test
11
12 ranks := [0: i in [1..61]];
13
14 t := Cputime();
15
16 for i in [1..2^16] do
17   j := rankOpK(solToBei(Random(a),n,k),n,k);
18   ranks[j+1] := ranks[j+1] + 1;
19 end for;
20
21 Cputime(t);
```

The time that we needed to check the rank of 2^{16} random operation was ~ 2034.790 seconds (almost 34 minutes).
The ranks obtained are all such that $\operatorname{Rk} B_\circ = n$ and so $\dim(U(T_\circ)) = 4$.

Bibliography

[AC09] Martin Albrecht and Carlos Cid, *Algebraic techniques in differential cryptanalysis*, pp. 193–208, Springer Berlin Heidelberg, Berlin, Heidelberg, 2009.

[BAK98] Eli Biham, Ross Anderson, and Lars Knudsen, *Serpent: A new block cipher proposal*, pp. 222–238, Springer Berlin Heidelberg, Berlin, Heidelberg, 1998.

[Bi14] Brian Bi, *Odd-dimensional skew-symmetric matrix is singular, even in a field of characteristic 2*, 2014.

[BKL+07] A. Bogdanov, L. R. Knudsen, G. Leander, C. Paar, A. Poschmann, M. J. Robshaw, Y. Seurin, and C. Vikkelsoe, *Present: An ultra-lightweight block cipher*, Proceedings of the 9th International Workshop on Cryptographic Hardware and Embedded Systems, CHES '07, Springer-Verlag, 2007, pp. 450–466.

[Cal15] Marco Calderini, *On boolean functions, symmetric cryptography and algebraic coding theory*, Ph.D. thesis, University of Trento - Department of Mathematics, April 2015.

[Car07] Claude Carlet, *Boolean functions for cryptography and error correcting codes*, 2007.

[Car11] ———, *Boolean functions*, Encyclopedia of Cryptography and Security, 2nd Ed., 2011, pp. 162–165.

[CDS05] A. Caranti, F. Dalla Volta, and M. Sala, *Abelian regular subgroups of the affine group and radical rings*, ArXiv Mathematics e-prints (2005).

[CDS08] ———, *On some block ciphers and imprimitive groups*, ArXiv e-prints (2008).

[CDVS09] A. Caranti, Francesca Dalla Volta, and M. Sala, *An application of the o'nan-scott theorem to the group generated by the round functions of*

an aes-like cipher, Designs, Codes and Cryptography **52** (2009), no. 3, 293–301.

[CW09] Carlos Cid and Ralf-Philipp Weinmann, *Block cipher: Algebraic cryptoanalysis and grobner bases*, pp. 307–327, Springer Berlin Heidelberg, Berlin, Heidelberg, 2009.

[Des77] Des, *Data encryption standard*, In FIPS PUB 46, Federal Information Processing Standards Publication, 1977, pp. 46–2.

[DR99] Joan Daemen and Vincent Rijmen, *Aes proposal: Rijndael*, 1999.

[DR02] Joan Daemen and Vincent Rijmen, *The design of rijndael*, Springer-Verlag New York, Inc., 2002.

[EG83] Shimon Even and Oded Goldreich, *Des-like functions can generate the alternating group.*, IEEE Trans. Information Theory **29** (1983), no. 6, 863–865.

[Fei73] H. Feistel, *Cryptography and computer privacy*, Scientific American, 1973.

[ISO12] *Information technology – Security techniques – Lightweight cryptography – Part 2: Block ciphers*, Standard, International Organization for Standardization, Geneva, CH, January 2012.

[Lan93] Serge Lang, *Algebra*, Addison-Wesley, Menlo Park Cal, 1993.

[Li03] Cai Heng Li, *The finite primitive permutation groups containing an abelian regular subgroup*, Proceedings of the London Mathematical Society **87** (2003), 725–747.

[LLM+10] Joel Brewster Lewis, Ricky Ini Liu, Alejandro H. Morales, Greta Panova, Steven V Sam, and Yan X Zhang, *Matrices with restricted entries and q-analogues of permutations*.

[LN97] R. Lidl and H. Niederreiter, *Finite fields*, EBL-Schweitzer, Cambridge University Press, 1997.

[Mac69] Jessie MacWilliams, *Orthogonal matrices over finite fields*, The American Mathematical Monthly **76** (1969), no. 2, 152–164.

[Mat94] Mitsuru Matsui, *Linear cryptanalysis method for des cipher*, pp. 386–397, Springer Berlin Heidelberg, Berlin, Heidelberg, 1994.

[MMMM13] D. Steven Mackey, Niloufer Mackey, Christian Mehl, and Volker Mehrmann, *Skew-symmetric matrix polynomials and their smith forms*, Linear Algebra and its Applications **438** (2013), no. 12, 4625 – 4653.

[MPW94] Sean Murphy, Kenneth Paterson, and Peter Wild, *A weak cipher that generates the symmetric group*, Journal of Cryptology **7** (1994), no. 1, 61–65.

[Pat99] Kenneth G. Paterson, *Imprimitive permutation groups and trapdoors in iterated block ciphers*, ch. Fast Software Encryption: 6th International Workshop, FSE'99 Rome, Italy, March 24–26, 1999 Proceedings, pp. 201–214, Springer Berlin Heidelberg, Berlin, Heidelberg, 1999.

[RD82] Dorothy Elizabeth Robling Denning, *Cryptography and data security*, Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1982.

[RP97] Vincent Rijmen and Bart Preneel, *A family of trapdoor ciphers*, pp. 139–148, Springer Berlin Heidelberg, Berlin, Heidelberg, 1997.

[Sha45] Claude Shannon, *A mathematical theory of cryptography*, Memorandum, Bell Laboratories, September 1945.

[Sha49] C. Shannon, *Communication theory of secrecy systems*, Bell System Technical Journal, Vol 28, pp. 656–715 (1949).

[Ste11] J. Stewart, *Calculus*, Cengage Learning, 2011.

[Sti02] Douglas Stinson, *Cryptography: Theory and practice,second edition*, 2nd ed., CRC/C&H, 2002.

[Str09] Gilbert Strang, *Introduction to linear algebra*, fourth ed., Wellesley-Cambridge Press, 2009.

[SW08] Rüdiger Sparr and Ralph Wernsdorf, *Group theoretic properties of rijndael-like ciphers*, Discrete Appl. Math. **156** (2008), no. 16, 3139–3149.

[Wer00] Ralph Wernsdorf, *The round functions of serpent generate the alternating group.*

Magma Code

In this section Magma scripts of the algorithms described in the thesis and some additional preparatory Magma scripts will be presented.

A.1 Auxiliary functions

As a general definition, we use this style to generate T_+ and $\mathrm{AGL}_+, \mathrm{GL}_+$ where we

1. Create the vector space V, once fixed a dimension n
2. We create the set of the elemnt of V
3. We generate $\mathrm{Sym}(V)$
4. We create the list of traslations $\tau_v = k_v\sigma_v$ for $v \in V$. In the code, we considered $k_v = I_n$.
5. We generate the subgroup of the translation $T_\circ$
6. $\mathrm{AGL}_\circ$ is generated as the normalizer of $T_\circ$ with respect to $\mathrm{Sym}(V)$
7. $\mathrm{GL}_\circ$ is the stabilizer of $\mathrm{AGL}_\circ$ with respect to 0

```
V := VectorSpace(GF(2),n);
Vs := {v:v in V};
S := Sym(Vs);
T := [map<V -> V| x :-> x+v> : v in V];
t := sub<S|[[t(v): v in Vs]:t in T]>;

AGLp := Normalizer(S,t);
GLp := Stabilizer(AGLp,V!0);
```

We then have some auxiliary function to navigate between permutation as group elements or as matrices.
For this reason we sometimes need to convert between this two representation.

```
// Auxiliary function to convert a decimal number in a n bit vector.
function decToBin(n,k)
```

```
  tmp :=(Intseq(n,2));
  zero :=[];
  if #tmp eq k then
  else
    zero := [0 : i in [1..(k-#tmp)]];
  end if;
  return  [GF(2)! i : i in tmp cat zero ];
end function;



// From permutation group to matrix + vector (from permGroup to AGL_+)

function PermToMatrix(permGroup,n)

  V := VectorSpace(GF(2),n);
  perm := Generators(permGroup);
  matrixs := [];
  coeffs := [];
  for i in perm do
    coeff := (V!0)^i;
    matrix := Matrix(GF(2),n,n,[ V!decToBin(2^j,n)^i + coeff : j in [0..(n-1)] ]);

    matrixs := Append(matrixs,matrix);
    coeffs := Append(coeffs,coeff);
  end for;

  return matrixs,coeffs;
end function;




// From single permutation to matrix + vector (from perm to AGL_+)

function PermSToMatrix(perm,n)
  V := VectorSpace(GF(2),n);
  matrixs := [];
  coeffs := [];
  perm := [perm];

  for i in perm do
    coeff := (V!0)^i;
    matrix := Matrix(GF(2),n,n,[ V!decToBin(2^j,n)^i + coeff : j in [0..(n-1)] ]);

    matrixs := Append(matrixs,matrix);
    coeffs := Append(coeffs,coeff);
  end for;

  return matrixs,coeffs;
end function;
```

A.2 constructOperation

This code generate the different M_{e_i}'s and their corrispective translation τ_{e_i}'s.

The coefficients are passed in input as list of list of integers, concerning the form described in Section 3.2.

```
function genMatrix(n,k,i, num)
  zero := [GF(2)|0 : i in [1..k]];
  matrix := [decToBin(num[i],k) : i in [1..#num]];
  I := IdentityMatrix(GF(2),(n+k));
  return InsertBlock( I , Matrix(GF(2),n,k,Insert(matrix,i,zero)) , 1 , (n+1) );
end function;

function genOperation(n,k,coeff)
  V := VectorSpace(GF(2),(n+k));
  e_i := [ V! decToBin(2^i,(n+k)) : i in [0..(n+k-1)]];


  valors := [  [coeff[j][(i-j)] : j in [1..(i-1)]] cat coeff[i] : i in [1..#coeff] ];

  B_ei := [genMatrix(n,k,i,valors[i]) : i in [1..n]];
  B_ei := B_ei cat [ IdentityMatrix(GF(2),(n+k)) : i in [1..k]];


  tau_ei := [ map< V -> V | x :-> x*B_ei[i] + e_i[i]> : i in [1..(n+k)]];

  return tau_ei , B_ei;
end function;

```

The algorithm's work-flow:

1. Generates the vectorial space $n + k$ and creates the canonical basis for it.
2. From the list *coeff* with $\binom{n}{2}$ elements, enlarges the list to a n^2 maintaining the valors$[i][i] == 0$ and valors$[i][j] ==$ valors$[j][i]$ that corresponds to the property of the matrices B_{e_i}'s.
3. Generates the matrix B_{e_i} with every single list and insert it in a identity matrix
4. Concatenates with k identity matrices
5. Creates the maps and return

A.3 searchLinearity

This function search for the operations $\circ$ that linearize λ.
It returns *aa* as the space of all the operations, *bb* as the generators of the space.

```
function searchOperation2(lambda,n,k,t)
```

```
2
3   V := VectorSpace(GF(2),(n+k));
4   Vn := VectorSpace(GF(2), (2*n*n*k) );
5   e_i := [ V! decToBin(2^i,(n+k)) : i in [0..(n+k-1)]];
6   e_il := [ElementToSequence(e_i[j] * lambda) : j in [1..#e_i]];
7
8   A := Submatrix(lambda,1,1,n,n);
9   B := Submatrix(lambda,(n+1),(n+1),k,k);
10
11  mat := SparseMatrix(GF(2),(2*n*n*k),n*n*k);
12
13  for t in [1..n] do
14    for i in [1..n] do
15      for j in [1..k] do
16
17        // Matrix A
18        for m in [1..k] do
19          mat[(t-1)*n*k + (i-1)*k + j][(t-1)*n*k + (i-1)*k + m] := mat[(t-1)*n*k + (i
    -1)*k + j][(t-1)*n*k + (i-1)*k + m] + B[m][j];
20        end for;
21
22        // Matrix B
23        for u in [1..n] do
24          if e_il[t][u] eq 1 then
25            for o in [1..n] do
26              mat[(t-1)*n*k + (i-1)*k + j][(u-1)*n*k + (o-1)*k + j] := mat[(t-1)*n*k
    + (i-1)*k + j][(u-1)*n*k + (o-1)*k + j] + A[i][o];
27            end for;
28          end if;
29        end for;
30      end for;
31    end for;
32  end for;
33
34  // eiB_ei = 0
35  // eiB_ej + ejB_ej = 0
36
37  pos := 1;
38  for j in [1..n] do
39    for i in [1..n] do
40      for l in [1..k] do
41        mat[n*n*k + pos][(i-1)*n*k + (j-1)*k + l] := 1;
42        mat[n*n*k + pos][(j-1)*n*k + (i-1)*k + l] := 1;
43        pos := pos + 1;
44      end for;
45    end for;
46  end for;
47
48  aa,bb := Solution(Transpose(Matrix(mat)), Vn!0);
49  return bb, [ solToBei(Eltseq(i),n,k) : i in Generators(bb)];
50
51end function;
52
```

The algorithm's work-flow:

1. Generates the vectorial space $n + k$ and creates the canonical basis for it.

2. Generates the vectorial space $2 * n^2k$ that will be the vector space where the operation space will be contained.

3. Calculates all the $e_i\lambda$'s and store the results.

4. Generates the sparse matrix that contains all the constrains to linearize λ

5. Finds the solution space and return it. Additionally it returns the generator of the space.

A.4 exactNumberOperation

This script, taking in input n and k, count all the operation $\circ$ defined over $\mathbb{F}^{n+k}$ such that $\dim U(T_\circ) = k$.

```
function countOperation(n,k)

  nn := n - k;

  function decToBin(n,k)
    tmp :=(Intseq(n,2));
    zero :=[];
    if #tmp eq k then
    else
      zero := [0 : i in [1..(k-#tmp)]];
    end if;
    return  [GF(2)! i : i in tmp cat zero ];
  end function;

  function genOperation(n,k,coeff)

    valors := [  [coeff[j][(i-j)] : j in [1..(i-1)]] cat [coeff[i][j] : j in [1..#
    coeff[i]]] : i in [1..#coeff] ] cat [[coeff[j][(n-j)] : j in [1..(n-1)]]];
    zero := [GF(2)|0 : i in [1..k]];
    matrix := [[decToBin(num[i],k) : i in [1..#num] ]: num in valors];
    B_ei := [Eltseq(Matrix(GF(2),n,k,Insert(matrix[i],i,zero))) : i in [1..#matrix]];
    return (B_ei);
  end function;



  t := Cputime();

  l := [0..2^k-1];
  ll := CartesianProduct([CartesianProduct([l : k in [1..nn-r]]) : r in [1..nn-1]]);

  cc := [0 : i in [1..(nn+2)]];

  for coeff in ll do
    op := genOperation(nn,k,coeff);
    mop := Matrix(GF(2),nn,nn*k, &cat op);
    m := Rank(mop);
    cc[m+1] := cc[m+1] + 1;
  end for;
```

```
39
40   print "$" ,cc[nn+1] ,"$ & $" ,  Cputime(t) , "$ \\";
41   return [];
42 end function;
```

The algorithm's work-flow is:

1. Generates the space of all the operation with B_{e_i}'s as $n \times k$ matrices.
2. Checks if $\operatorname{Rk} B_\circ = k$ and counts the positive matches.
3. Prints the number of operations and the time spent in computation.