Bachelor thesis
Computing Science

Radboud University

Influence of Design on Differential and Linear Propagation Properties of Block Cipher Family Skinny

Author:
Denise Verbakel
s1018597

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

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

Second assessor:
Dr. ir. Shahram Rasoolzadeh
shahram.rasoolzadeh@ru.nl

June 2, 2021

Abstract

Designing a good performing block cipher is not as obvious as it seems. Almost every designer is looking into the basic aspects of a block cipher, such as the round function, the cost regarding hardware and software implementations and security aspects. However, in addition to this, it is also important to study the interaction of the nonlinear and linear layers inside the round function. This should be done in order to better understand how certain design choices of a cipher affect differential and linear propagation properties. A better understanding in this will make us able to design more efficient round functions and, as a logical consequence, more efficient ciphers. In this thesis, we systematically analyze the design approach of SKINNY with respect to the interactions between the layers in the round function and their impact on the differential and linear propagation properties. We show that alignment is a very important property that greatly influences these propagation properties. We also elaborate upon one of its most profound consequences called clustering and state that also this phenomenon affects the differential and linear propagation properties.

Keywords: cryptanalysis, permutations, propagation properties, round functions, SKINNY, symmetric cryptography, (tweakable) block ciphers

Contents

1 Introduction **3**
- 1.1 Related Work . . . 4
- 1.2 Our Contribution . . . 5
- 1.3 Outline . . . 5

2 Preliminaries **6**
- 2.1 Cryptographic Primitives . . . 6
  - 2.1.1 Permutation . . . 6
  - 2.1.2 (Tweakable) Block Cipher . . . 7
- 2.2 Notation and Conventions . . . 8
  - 2.2.1 General Notation and Conventions . . . 8
  - 2.2.2 Notation Regarding Sets . . . 8
  - 2.2.3 Notation Regarding Finite Fields . . . 8
  - 2.2.4 Notation Regarding Linear Transformations . . . 9
  - 2.2.5 Terminology and Associated Notation . . . 9
- 2.3 Cryptanalysis . . . 11
  - 2.3.1 Differential Cryptanalysis . . . 11
  - 2.3.2 Linear Cryptanalysis . . . 16
- 2.4 Box Partitioning and Alignment . . . 19
  - 2.4.1 Nonlinear Layer N . . . 19
  - 2.4.2 Box Partitions . . . 19
  - 2.4.3 Alignment . . . 20
  - 2.4.4 Superbox Structure . . . 21
- 2.5 Bit and Box Weight Histograms . . . 22
  - 2.5.1 Activity and Weights . . . 22
  - 2.5.2 Weight Histograms . . . 23
  - 2.5.3 Extensions of Branch Numbers . . . 24
  - 2.5.4 Ordered Weight Pairs . . . 25
- 2.6 Two-round Trail Weight Histograms . . . 25
- 2.7 Cluster Histograms . . . 27
  - 2.7.1 Equivalence Classes . . . 27
  - 2.7.2 Cluster Histogram . . . 28
- 2.8 Two-round Trail Clustering . . . 30

3 Block Cipher Family Skinny **32**
3.1 Description of Skinny . . . 32
3.1.1 Instances of Skinny . . . 32
3.1.2 Skinny's Round Function . . . 33
3.1.3 Skinny's Alignment Properties . . . 36
3.2 Round Cost . . . 37
3.2.1 S-Box Layer . . . 38
3.2.2 Mixing Layer . . . 40
3.2.3 Shuffle Layer . . . 42

4 Huddling **43**
4.1 Making Skinny Instances Larger . . . 43
4.2 Bit Weight Histograms . . . 44
4.2.1 Distribution of Ordered Bit Weight Pairs Skinny64 . 47
4.2.2 Distribution of Ordered Bit Weight Pairs Skinny128 . 49
4.3 Box Weight Histograms . . . 50
4.3.1 Distribution of Ordered Box Weight Pairs Skinny64 . 53
4.3.2 Distribution of Ordered Box Weight Pairs Skinny128 54
4.4 Huddling . . . 55
4.4.1 Bit Huddling . . . 55
4.4.2 Superbox Huddling Effect . . . 56
4.5 Two-round Trail Weight Histograms . . . 57

5 Clustering **61**
5.1 Cluster Histograms . . . 61
5.2 Two-round Trail Clustering . . . 62
5.2.1 Linear Dependencies . . . 66

6 Conclusions **70**

A Espresso Algorithm: Minimal Sum-Of-Product (SOP) Form **75**
A.1 SOP Expressions . . . 75
A.2 S-boxes of Skinny64 and Skinny128 . . . 77

B Two-dimensional Histograms **78**
B.1 Distribution of Ordered Bit Weight Pairs of Skinny64 . . . 78
B.2 Distribution of Ordered Box Weight Pairs of Skinny64 . . . 81
B.3 Distribution of Ordered Bit Weight Pairs of Skinny128 . . . 84
B.4 Distribution of Ordered Box Weight Pairs of Skinny128 . . . 87

Chapter 1

Introduction

In the past, the National Institute of Standards and Technology (NIST) has often held competitions in which block ciphers compete against each other. A recent example of such a competition is the NIST lightweight cryptography competition that was made to "solicit, evaluate, and standardize lightweight cryptographic algorithms that are suitable for use in constrained environments where the performance of current NIST cryptographic standards is not acceptable" [26]. Many of these block ciphers are made by using a round function, which in its turn most of the time consists out of four layers: a nonlinear S-box layer, a mixing layer, a shuffle layer and a constant or round key addition. In the literature, there has been research about how costly their implementations are. However, not a lot of in-depth thought has been put into the interaction between the nonlinear and linear layers inside such a round function. A few investigations were already done in this area, but not as intensive as recently done by [8].

The importance of investigating the interaction between the nonlinear and linear layers can be found in attacks that are based on cryptanalysis against block ciphers. The rationale of such an attack is: the more you know, the more we can attack or break. A concrete example of such an attack is the so-called truncated differential attack [21]. These attacks "exploit sets of differentials that share the same box activity pattern in their input difference and the same box activity pattern in their output difference" [8]. These activity patterns are results of how the layers of a certain block cipher are constructed and thus we want to avoid these problems in the ciphers design.

By looking at the ciphers that did enter the NIST lightweight cryptography competition, we notice that there are several ciphers that use the Skinny primitive, namely ForkAE, Remus, Romulus, Skinny-AEAD and Skinny-Hash. Yet, this primitive has never been investigated on how

the different layers of its round function influence its performance. In this thesis, we want to give insight into the block cipher family SKINNY and analyze, just as was done in [8], the impact of the interaction of the nonlinear and linear layer on the differential and linear propagation properties of this particular block cipher family. Therefore, we address the following research question in this thesis:

"How do the linear and nonlinear layers in the round function of SKINNY *interact when it comes to the differential and linear propagation properties?"*

In other words, in this thesis we want to analyze how the design choices of the round function - in particular the choice on how the nonlinear S-box layer and the linear mixing layer are used together - have an impact on the propagation properties. Here, we mean with propagation properties the (truncated) differential and linear trails and their probability distribution.

In order to answer our research question, we created some programs based on the tools used in [8]. These programs can be found at

`https://github.com/DVerbakel/BachelorThesis`

In the description of this GitHub repository, we also provide the link to the original tools written for [8].

1.1 Related Work

As a logical consequence, this thesis functions as an extension to [8] to gain knowledge about SKINNY. In that research the notion of alignment is formalized and the four primitives RIJNDAEL, SATURNIN, SPONGENT and XOODOO are investigated. They "propose a way to analyze the interactions between the linear and the nonlinear layers w.r.t. the differential and linear propagation" [8] and compare the four ciphers using experiments performed by a computer. They have defined what alignment means; have shown that RIJNDAEL, SATURNIN and SPONGENT are aligned, whereas XOODOO is unaligned; and have shown that exactly this alignment leads to different forms of clustering.

We use the same definitions, notations and conventions as used in [8]: see chapter 2. However, this work also uses standard terminology originating from the differential and linear cryptanalysis. This means that in theory we will use, for instance, [4] and [24] as references as well. This also holds for [1], since they define the block cipher family SKINNY and discusses the round cost of the round function on which we will elaborate in chapter 3.

1.2 Our Contribution

In this thesis, we expose block cipher family SKINNY to the same studies as done in [8]. We study alignment and one of its consequences, the so-called clustering, as a very important property that influences the differential and linear propagation properties through the interplay of the linear and nonlinear layers. We do this by going into detail about, for instance, bit, box and two-round trail weight histograms, the huddling effect, cluster histograms and two-round trail clustering. Most results we obtain in these areas are also compared with the results of found in [8].

1.3 Outline

This thesis consists of 6 chapters of which the introduction is the first one. After this, chapter 2 discusses all the preliminaries needed in order to understand the investigation into block cipher family SKINNY and the presented results. Next, chapter 3 introduces this block cipher family and discusses the round cost of three different layers. In chapter 4 we investigate the bit and box weights and the related phenomenon called huddling. In chapter 5 we look into another effect, namely clustering. Finally, chapter 6 concludes our thesis.

We also provide two appendices along with this work. Appendix A provides information about the Espresso algorithm we used in chapter 3 and Appendix B includes more information about the two-dimensional histograms from chapter 4.

Chapter 2

Preliminaries

Before investigating the block cipher family SKINNY and answering our research question, we provide some background information. As previously stated, this thesis functions as an extension to [8]. Since we will use the same formulations and definitions as given in that research, we need some prior knowledge about notation, conventions, cryptanalysis and terminology introduced in [8] (section 2.2 to section 2.8). Next to this, we also briefly present basic cryptographic primitives relevant for this study (section 2.1).

2.1 Cryptographic Primitives

2.1.1 Permutation

In this thesis we encounter *permutations*, which are denoted by $f : \mathbb{F}_2^b \to \mathbb{F}_2^b$ [8]. We elaborate on this notation in subsection 2.2.3, but here we first present the underlying idea of a permutation.

The definition of a permutation is "an ordered arrangement of the elements of a set" [29]. In other words, it is a bijective function τ that uniquely maps each of the 2^b possible inputs to one of the 2^b possible outputs. Simply said, what a permutation thus does, is changing the b elements of a certain state. A schematic representation of a permutation is shown in Figure 1.

Figure 1: Schematic representation of a permutation.

2.1.2 (Tweakable) Block Cipher

Another cryptographic term that is often used is *block cipher*. We can denote it by $E : \mathbb{F}_2^b \times \mathbb{F}_2^k \to \mathbb{F}_2^b$ [18]. Again, this notation is elaborated upon in subsection 2.2.3.

Block ciphers are "conventional cryptosystems that typically handle a fixed number of symbols at a time (under a given key) and do this encryption/decryption independent of past input blocks. For the encryption process, the data (plaintext) enters the block cipher from the left and leaves it on the right as ciphertext" [37]. In other words, we can actually describe a block cipher as a keyed permutation. This means that we can see the key in a block cipher as an index: the key is linked to a certain permutation from a large array of different permutations. A schematic representation of a block cipher is shown in Figure 2.

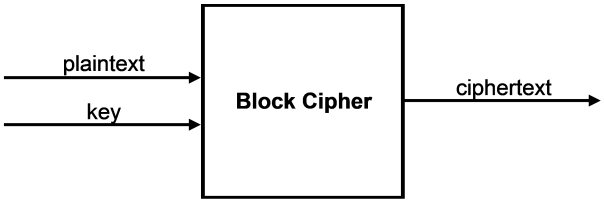


Figure 2: Schematic representation of a block cipher with a b-bit plaintext input and a k-bit key input [18].

Next to normal block ciphers, there are also *tweakable block ciphers*, which we can denote by $\tilde{E} = \mathbb{F}_2^b \times \mathbb{F}_2^k \times \mathbb{F}_2^t \to \mathbb{F}_2^b$ [18] (see subsection 2.2.3). The difference between these two variants is that a block cipher only takes a key next to the plaintext, whereas a tweakable block cipher takes both a key and a tweak next to the plaintext [18]. The purpose of using this extra input is to avoid a high cost for variability when the key is fixed. When using a tweak, we can thus still easily introduce such variability. A schematic representation of a tweakable block cipher is shown in Figure 3.

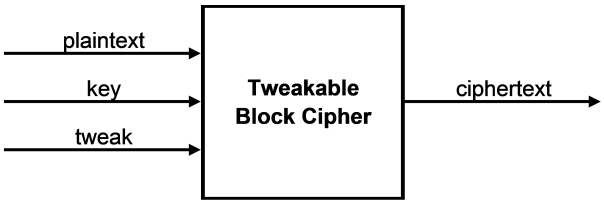


Figure 3: Schematic representation of a tweakable block cipher with a b-bit plaintext input, a k-bit key input and a t-bit tweak input [18].

In this thesis, the tweakable block cipher SKINNY is transformed into a permutation by using the *fixed-key perspective* as done to the other ciphers discussed in [8]. This means that all the bits of the key and the tweak are set to 0 such that we have a key and tweak that consists of only zeros. In this way, the tweak-addition of SKINNY (see section 3.1.2) will not have any impact on the other bits in the round function.

2.2 Notation and Conventions

To understand the mathematical aspects of this thesis, we will first introduce notation, terminology and conventions. These are the same as described in [8], but then slightly elaborated upon.

2.2.1 General Notation and Conventions

Non-negative integers are denoted by $\mathbb{Z}_{\geq 0}$ and positive integers by $\mathbb{Z}_{>0}$. We use $k \in \mathbb{Z}_{\geq 0}$ as a placeholder for a non-negative integer value. This thus means that k can represent any non-negative integer.

Indices will always begin at 0 instead of 1. Next to this, we define

$$[0, k-1] = \{i \in \mathbb{Z}_{\geq 0} : 0 \leq i \leq k-1\}$$

2.2.2 Notation Regarding Sets

Let S be a set and let $\sim$ be an equivalence relation on S. Recall that the relation $\sim$ is an equivalence relation if it is reflexive ($\forall x \in S : x \sim x$), symmetric (if $x \sim y$ then $y \sim x$), and transitive (if $x \sim y$ and $y \sim z$ then $x \sim z$) [29]. We then write $[a]_\sim$ for the equivalence class of $a \in S$, which is a subset of S defined by:

$$[a]_\sim = \{b \in S : b \sim a\}$$

Next to this, we denote the cardinality of S as $\#S$.

2.2.3 Notation Regarding Finite Fields

Let $\mathbb{F}_2$ be a finite field (made from the set $\{0, 1\}$). A finite field is defined as a set with a finite number of elements "in which four operations (called addition, multiplication, subtraction, and division) can be defined so that, with the exception of division by zero, the sum, product, difference, and quotient of any two elements in the set is an element of the set" [19]. In this work, we use the symbol + for vector addition in $\mathbb{F}_2$ (which is equal to

bitwise addition). For a more mathematical definition of (finite) fields, we refer to [19].

Next to this, we define $\mathbb{F}_2^k$ to be a vector space of dimension k over the finite field $\mathbb{F}_2$. We know that there exists an isomorphism f between $(\mathbb{F}_2)^k$ and $\mathbb{F}_{2^k}$ given by

$$v \star w = f^{-1}\left(f(v) \bigstar f(w)\right)$$

Here, $v, w \in (\mathbb{F}_2)^k$ are vectors, $\star$ is defined as the multiplication in $(\mathbb{F}_2)^k$ and $\bigstar$ is defined as the multiplication in $\mathbb{F}_{2^k}$. Because of this, we know that the vector space $\mathbb{F}_2^k$ has the structure of a finite field of cardinality 2^k.

The ith standard basis vector in $\mathbb{F}_2^k$ is notated as e_i^k. Recall that a set of these standard basis vectors is a linearly independent subset of $\mathbb{F}_2^k$ that generates $\mathbb{F}_2^k$ [19]. For $j \in [0, k-1]$ we have

$$e_{ij}^k = \begin{cases} 1 & \text{if } i = j \\ 0 & \text{otherwise} \end{cases}$$

2.2.4 Notation Regarding Linear Transformations

Let $\mathrm{L} : \mathbb{F}_2^b \to \mathbb{F}_2^b$ be a linear transformation. Recall that a function is a linear transformation when for all $x, y \in \mathbb{F}_2^b$ and $c \in \mathbb{F}_2$ the following holds [19]:

$$\mathrm{L}(x + y) = \mathrm{L}(x) + \mathrm{L}(y)$$
$$\mathrm{L}(cx) = c\mathrm{L}(x)$$

Note that the second equation always holds when $x \in \mathbb{F}_2^b$ and $c \in \mathbb{F}_2$: if we have $c = 0$, then this equation states that $\mathrm{L}(0) = 0$ and if we have $c = 1$, it states that $\mathrm{L}(x) = \mathrm{L}(x)$. If also the first equation is satisfied, there exists a matrix $\mathrm{M} \in \mathbb{F}_2^{b \times b}$ such that

$$\mathrm{L}(a) = \mathrm{M}a$$

The transpose $\mathrm{L}^\top : \mathbb{F}_2^b \to \mathbb{F}_2^b$ of this linear transformation is defined as

$$\mathrm{L}^\top(a) = \mathrm{M}^\top a$$

In case the inverse of $\mathrm{L}^\top$ is well-defined, this inverse is denoted by $\mathrm{L}^{-\top}$.

2.2.5 Terminology and Associated Notation

A *state* is a vector of b bits, denoted by $a \in \mathbb{F}_2^b$. To a state we can, for example, apply a permutation as explained in subsection 2.1.1. We can address the ith component of a state a with a_i.

In this work, the index sets $\mathrm{B}_i \subseteq [0, b-1]$ form a so called *ordered partition*. An index set is a set of one or more elements that partitions the full b-bit state in smaller pieces. An ordered partition of such an index set B_i can be described as a collection of disjoint nonempty subsets that have the number of bits in B_i as their union [29] in which the order of the elements in the subsets is important.

A *projection* is described as "a function that produces relations of smaller degree from an n-ary relation by deleting fields" [29]. In other words, a projection maps a set to a subset. We write $P_i(a) : \mathbb{F}_2^b \to \mathbb{F}_2^{\#B_i}$ for the projection onto the bits of a indexed by B_i.

A permutation can be built by composing a number of lightweight *round functions*. For some $r \in \mathbb{Z}_{>0}$ we can then write

$$f\,[r] = \mathrm{R}_{r-1} \circ \mathrm{R}_{r-2} \circ \cdots \circ \mathrm{R}_1 \circ \mathrm{R}_0$$

with $f[0] = \mathrm{id}$, which is the identity function [29]. A round function f is, in its turn, composed of *step functions*. For $i \in [0, r-1]$ we write

$$\mathrm{R}_i = \iota_i \circ \mathrm{L}_i \circ \mathrm{N}_i$$

where ι_i is addition of a round constant (a constant that depends on a particular round), L_i is a linear map and N_i is a nonlinear map. Another way of notating this would be

$$\mathrm{R}_i = \iota_i \circ \mathrm{L} \circ \mathrm{N}$$

where L is the linear layer of f and N is the nonlinear layer of f. We can do this in case the round function is identical with the exception of the round constant addition ι.

The nonlinear layer N consists of multiple *S-boxes*. In N there are n S-boxes of size m. An S-box is thus a nonlinear transformation which operates on a projection of the state, e.g., $\mathbb{F}_2^m$ such that $n \cdot m = b$ [17]. Here, we suppose in combination with index sets and $j \in [0, b-1]$ that

$$\mathrm{B}_j = \{jm, \ldots, (j+1)m - 1\}$$

A *shuffle (layer)* is a linear transformation $\pi : \mathbb{F}_2^b \to \mathbb{F}_2^b$ given by

$$\pi(a) = P_\tau a$$

where τ is a permutation of the index space written as $\tau : [0, b-1] \to [0, b-1]$ and P_τ is the permutation matrix associated with some τ. This

permutation matrix is obtained by permuting the columns of the $(b \times b)$ identity matrix according to τ, where this identity matrix is defined as

$$\underbrace{\begin{pmatrix} 1 & 0 & \cdots & 0 \\ 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 1 \end{pmatrix}}_{b \text{ columns}} \left.\vphantom{\begin{pmatrix} 1 \\ 0 \\ \vdots \\ 0 \end{pmatrix}}\right\} b \text{ rows}$$

2.3 Cryptanalysis

In the proposed research question, it is mentioned that we want to investigate the impact on differential and linear propagation properties. In order to do so, we need to know what differential and linear cryptanalysis is and what this entails. We start by describing the concepts of differential and linear cryptanalysis and follow up with a list of definitions used in [8].

2.3.1 Differential Cryptanalysis

Differential cryptanalysis is "a method which analyzes the effect of particular differences in plaintext pairs on the differences of the resultant ciphertext pairs" [4]. This statistical cryptanalysis exploits the (high) probabilities of these differences [4, 6]. We will now introduce and expand the terminology and definitions about differential cryptanalysis given in [8].

First of all, we give the differences as stated above a name: an ordered pair of input and output difference is called a *differential*. The notation we use is $(\Delta_{\text{in}}, \Delta_{\text{out}}) \in \mathbb{F}_2^b \times \mathbb{F}_2^b$, where thus Δ_{in} represents the input difference and Δ_{out} the output difference.

If we take a permutation $f : \mathbb{F}_2^b \to \mathbb{F}_2^b$, a fixed $x \in \mathbb{F}_2^b$ and a fixed $\Delta_{\text{in}} \in \mathbb{F}_2^b$, we can look at the difference between $f(x)$ and $f(x + \Delta_{\text{in}})$. This difference is then called Δ_{out}. We can define the *solution set of the differential* $(\Delta_{\text{in}}, \Delta_{\text{out}})$ as the set of all $x \in \mathbb{F}_2^b$ for which the difference between $f(x)$ and $f(x + \Delta_{\text{in}})$ equals the given Δ_{out}. A mathematical notation for this is given in Definition 1.

Definition 1

Let $f : \mathbb{F}_2^b \to \mathbb{F}_2^b$ be a permutation and define

$$U_f\left(\Delta_{\text{in}}, \Delta_{\text{out}}\right) = \left\{x \in \mathbb{F}_2^b : f(x) + f\left(x + \Delta_{\text{in}}\right) = \Delta_{\text{out}}\right\}$$

We call $U_f\left(\Delta_{\text{in}}, \Delta_{\text{out}}\right)$ the *solution set of the differential* $\left(\Delta_{\text{in}}, \Delta_{\text{out}}\right)$.

We can count the number of states x that are in the solution set of a certain differential. Next to this, since $x \in \mathbb{F}_2^b$ and since $\mathbb{F}_2^b$ is a vector space having dimension b, we know that there are 2^b possible states x. We introduce the term *differential probability (DP)* as the probability of having a particular number of states x in a solution set. This is shown in Definition 2.

Definition 2

The *differential probability (DP) of a differential* $\left(\Delta_{\text{in}}, \Delta_{\text{out}}\right)$ over the permutation $f : \mathbb{F}_2^b \to \mathbb{F}_2^b$ is defined as

$$\text{DP}_f\left(\Delta_{\text{in}}, \Delta_{\text{out}}\right) = \frac{\#U_f\left(\Delta_{\text{in}}, \Delta_{\text{out}}\right)}{2^b}$$

When $\#U_f\left(\Delta_{\text{in}}, \Delta_{\text{out}}\right) \geq 1$, we know that we have $\#U_f\left(\Delta_{\text{in}}, \Delta_{\text{out}}\right) \geq 2$ since if $x \in U_f\left(\Delta_{\text{in}}, \Delta_{\text{out}}\right)$ then it follows that $x + \Delta_{\text{in}} \in U_f\left(\Delta_{\text{in}}, \Delta_{\text{out}}\right)$. In case we have such x and $x + \Delta_{\text{in}}$ that are in the solution set of the differential, the ordered pair $(x, x + \Delta_{\text{in}})$ is said to *follow* the differential $\left(\Delta_{\text{in}}, \Delta_{\text{out}}\right)$. In this case, Δ_{in} (the input difference) is *compatible* with Δ_{out} (the output difference) through f. We then call $\left(\Delta_{\text{in}}, \Delta_{\text{out}}\right)$ a *valid* differential.

We can also look at the differential probability for each step function inside the round function. If we have that $\#U_{\text{R}_i}\left(q^{(i)}, q^{(i+1)}\right) \geq 1$ holds for a sequence of input and/or output differences $\left(q^{(0)}, q^{(1)}, \ldots, q^{(k)}\right) \in \left(\mathbb{F}_2^b\right)^{k+1}$, we can say that this sequence is a *k-round differential trail*. In other words: if for all $0 \leq i \leq k-1$, there exists at least one $x \in \mathbb{F}_2^b$ such that

$$\text{R}_i(x) + \text{R}_i\left(x + q^{(i)}\right) = q^{(i+1)}$$

then we call the sequence of input and/or output differences a k-round differential trail. Definition 3 formulates this concept mathematically.

Definition 3

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

This is not the only possible notation of a trail. Another way to specify a k-round differential trail is $Q = (b_{-1}, a_0, b_0, \ldots, a_k, b_k)$. Here, the trail is written by giving the intermediate differences between N_i and L_i as well. This means that, in relation to the notation in Definition 3, we have $q^{(i+1)} = b_i = \mathrm{L}_i(a_i)$.

The *set of all the differential trails in a differential* $\left(q^{(0)}, q^{(k)}\right)$ can be referred to by $\mathrm{DT}\left(\Delta_{\mathrm{in}}, \Delta_{\mathrm{out}}\right)$ if we take $q^{(0)} = \Delta_{\mathrm{in}}$ and $q^{(k)} = \Delta_{\mathrm{out}}$. The differential $(\Delta_{\mathrm{in}}, \Delta_{\mathrm{out}})$ is then addressed as the *enveloping differential* of the trails in $\mathrm{DT}\left(\Delta_{\mathrm{in}}, \Delta_{\mathrm{out}}\right)$. In case $\#\mathrm{DT}\left(\Delta_{\mathrm{in}}, \Delta_{\mathrm{out}}\right) > 1$, we state that these trails *cluster* together in the differential $(\Delta_{\mathrm{in}}, \Delta_{\mathrm{out}})$.

We define the *differential trail core* as $\left(q^{(1)}, \ldots, q^{(k-1)}\right)$, which is a set of differential trails with the same inner differences. We obtain this core by leaving out the initial and final difference of a differential trail given by $\left(\Delta_{\mathrm{in}}, q^{(1)}, \ldots, q^{(k-1)}, \Delta_{\mathrm{out}}\right)$. The differential trail core is said to be *in* the differential $(\Delta_{\mathrm{in}}, \Delta_{\mathrm{out}})$.

As we can infer from the definitions above, each round differential $\left(q^{(i)}, q^{(i+1)}\right)$ has a solution set $U_{\mathrm{R}_i}\left(q^{(i)}, q^{(i+1)}\right)$. We consider the transformed set of points $U_i = f[i]^{-1}\left(U_{\mathrm{R}_i}\left(q^{(i)}, q^{(i+1)}\right)\right)$ at the input of f. We first transform these points in order to be able to take an intersection: otherwise, we are looking at pairs between $q^{(i)}$ and $q^{(i+1)}$ instead of at Δ_{in}. In other words, we should translate the pairs to the input such that they are defined at the same state instead of at different intermediate states. If an ordered pair $(x, x + \Delta_{\mathrm{in}})$ *follows* the differential trail, it should hold that x is in each solution set of each round differential. In another notation:

$$x \in U_f(Q) = \bigcap_{i=0}^{k-1} U_i \tag{1}$$

Analogously to Definition 2, we can now define the *DP of a differential trail* as the fraction of states x that satisfy the relation in (1): see Definition 4.

Definition 4

The *DP of a differential trail* is defined as

$$\mathrm{DP}_f(Q) = \frac{\#U_f(Q)}{2^b}$$

After having defined $\mathrm{DP}_f(Q)$, we can now determine whether or not the round differentials are (statistically) independent [34]. This is written down in Definition 5.

Definition 5

The round differentials are said to be *independent* if

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

We know that every ordered pair $(x, x+\Delta_{\mathrm{in}})$ follows exactly *one* differential trail: with the same x and the same initial difference, we cannot suddenly get different intermediate and final differences. This thus means that every $x \in U_f(\Delta_{\mathrm{in}}, \Delta_{\mathrm{out}})$ is just in a single differential trail Q and hence in one element of $U_f(Q)$. Since the set of differential trails makes up a differential, we see that the following equation holds:

$$\mathrm{DP}_f(\Delta_{\mathrm{in}}, \Delta_{\mathrm{out}}) = \sum_{Q \in \mathrm{DT}(\Delta_{\mathrm{in}}, \Delta_{\mathrm{out}})} \mathrm{DP}_f(Q) \tag{2}$$

In words, this relation states that the DP of a differential $(\Delta_{\mathrm{in}}, \Delta_{\mathrm{out}})$ must be equal to the sum of the DPs of all the differential trails with Δ_{in} as input difference and Δ_{out} as output difference.

Next to this, we can also formulate a relation between the DP of a round differential and the DP values of its S-box differentials with a slightly different reasoning as above. First, we note that we can easily compute any differential $(\Delta_{\mathrm{in}}, \Delta_{\mathrm{out}})$ over a round function R. If we specify the intermediate differences b and c, we can describe the differential trail as $(\Delta_{\mathrm{in}}, b, c, \Delta_{\mathrm{out}})$. However, due to the linearity of L, we can write $c = \mathrm{L}(b)$ and thus $b = \mathrm{L}^{-1}(c)$. We know that a difference is invariant under addition of a constant as well: adding the same number to both sides has no impact when subtracting the numbers from each other. From these two remarks, it follows that we can also represent the differential trail as $\left(\Delta_{\mathrm{in}}, \mathrm{L}^{-1}(\Delta_{\mathrm{out}}), \Delta_{\mathrm{out}}, \Delta_{\mathrm{out}}\right)$. Secondly, we observe that the differential only contains a single trail and that its DP is thus the DP of

the differential $\left(\Delta_{\text{in}}, \mathrm{L}^{-1}(\Delta_{\text{out}})\right)$ over the S-box layer consisting of multiple S-boxes. This holds because of our reformulation of the differential trail. Mathematically we formulate this relation as follows:

$$\mathrm{DP}_{\mathrm{R}}\left(\Delta_{\text{in}}, \Delta_{\text{out}}\right) = \prod_{0 \leq j < n} \mathrm{DP}_{\mathrm{S}_j}\left(P_j(\Delta_{\text{in}}), P_j\left(\mathrm{L}^{-1}(\Delta_{\text{out}})\right)\right) \tag{3}$$

Here, S_j represents the j^{th} S-box in the S-box layer and P_j represents a projection on the j^{th} box. In words, this equation denotes that the DP of a round differential is the product of the DP values of all its S-box differentials.

The last terminology regarding differentials and differential trails that we will discuss, is the so called *restriction weight*. This concept is derived from the term 'information content' from Shannon's entropy [33]. In case the solution set U_f is linear, this weight expresses approximately the same as the rank of a matrix [19] does. It thus represents the number of equations to solve: it shows the number of independent equations. The restriction weight can be defined as is shown in Definition 6.

Definition 6

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

$$\mathrm{w_r}\left(\Delta_{\text{in}}, \Delta_{\text{out}}\right) = -\log_2 \mathrm{DP}_f\left(\Delta_{\text{in}}, \Delta_{\text{out}}\right)$$

Analogously, we can define the *restriction weight of a differential trail*. For this, we apply the same method as done in previous definitions: we look at the round differentials. To obtain the restriction weight of a differential trail, we will thus sum the weights of the different round differentials. The mathematical notation for this is given in Definition 7.

Definition 7

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

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

If the round differentials are independent (see Definition 5), we have an extra relation between the DP of a differential trail and the restriction weight of that differential trail. We have, in fact, that

$$\mathrm{DP}_f(Q) = 2^{-\mathrm{w_r}(Q)}$$

2.3.2 Linear Cryptanalysis

Linear cryptanalysis is "a known plaintext attack in which the attacker studies probabilistic linear relations (called linear approximations) between parity bits of the plaintext, the ciphertext, and the secret key" [5]. In other words, this method exploits high correlations (in absolute sense) between linear combinations of input bits and linear combinations of output bits of a permutation [8, 24]. Analogously to differential cryptanalysis, after knowing what linear cryptanalysis is about, we will state and elaborate upon the terminology and definitions given in [8].

We start off by making the correlation (pattern) between the linear combinations of input bits and linear combinations of output bits concrete. We call this the *(signed) correlation between the linear mask at the input and the linear mask at the output* of a permutation in $\mathbb{F}_2^b$. This concept is described in Definition 8.

Definition 8

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

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

To see if $\mathrm{C}_f(u, v) \neq 0$ we look at $u^\top x$ and $v^\top f(x)$ for all possible x. If these two values are equal, we increment the number; if these values are not equal, we decrement the number. So, when $\mathrm{C}_f(u, v) \neq 0$, it means that there is some imbalance in the number of (dis)agreements for all possible x. If this is the case, we say that u is *compatible* with v. The ordered pair of linear masks (u, v) is called a *linear approximation*. Next to a linear approximation, we also have *linear trails*. These linear trails are somewhat similar to the differential trails as described in Definition 3. Definition 9 discusses the linear cryptanalysis counterpart of a trail.

Definition 9

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

Next to this similar definition from differential cryptanalysis, we have a lot of analogue interpretations in linear cryptanalysis. We start off with the comparable definitions for the set of all differential trails in a differential, the

enveloping differential, clustering, the differential trail core and being in a differential.

The *set of all the linear trails in a linear approximation* (u, v) can be addressed as $\mathrm{LT}(u, v)$ if we take $q^{(0)} = u$ and $q^{(k)} = v$. The linear approximation (u, v) is then called the *enveloping linear approximation* of the trails in $\mathrm{LT}(u, v)$. In case $\#\mathrm{LT}(u, v) > 1$, we say that these trails *cluster* together in the linear approximation (u, v).

We define the *linear trail core* as $\left(q^{(1)}, \ldots, q^{(k-1)}\right)$, which is a set of linear trails with the same inner linear masks. We obtain this core by leaving out the initial and final linear mask of a linear trail given by $\left(u, q^{(1)}, \ldots, q^{(k-1)}, v\right)$. The linear trail core is said to be *in* the linear approximation (u, v).

Next, we will describe the *correlation contribution* of a linear trail Q over a function $f : \mathbb{F}_2^b \to \mathbb{F}_2^b$. This contribution of a trail can be described as the product of each (signed) correlation between $q^{(i)}$ and $q^{(i+1)}$ of each step function R_i. A formal format is given in Definition 10.

Definition 10

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

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

We see that this definition looks like Definition 5, even though the context of both these definitions differs a lot.

If we use correlation matrices [36] and the theory that is known about these matrices [14] we can conclude that, in combination with Definition 10, it follows that

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

Note that this equation bears some resemblance with equation (2).

Similarly to what is shown in equation (3), we can work towards a relation between the correlation contribution of a round function and the (signed) correlation of the S-boxes. We denote that we can also easily compute any linear approximation (u, v) over a round function R. If we specify the intermediate linear masks b and c, we can describe the linear trail as (u, b, c, v). However, due to the linearity of L, we can write $c = \mathrm{L}(b)$ and thus $b = L^{\top}(c)$. Combining this with the invariance of constant addition, it follows that we

can represent the linear trail as $\left(u, \mathrm{L}^{\top}(v), v, v\right)$. We now observe that the linear approximation only contains a single trail and that its correlation contribution is the (signed) correlation of the linear approximation $\left(u, L^{\top}(v)\right)$ over the S-box layer, consisting of multiple S-boxes. This holds because of our reformulation of the linear trail. In comparison to equation (3), in this linear case the round constant addition does affect the sign of correlation. We thus get the following:

$$\mathrm{C_R}(u, v) = (-1)^{v^{\top}\iota(0)} \prod_{0 \le j < n} \mathrm{C_{S_j}} \left(P_j(u), P_j\left(\mathrm{L}^{\top}(v)\right)\right)$$

Again, S_j represents the j^{th} S-box in the S-box layer and P_j represents a projection on the j^{th} box.

After this, we introduce the *linear potential* of a linear approximation, which is a data complexity metric. The higher this number gets, the worse it is and the more likely a linear attack will be successful. This metric is defined as given in Definition 11.

Definition 11

The *linear potential (LP)* of a linear approximation (u, v) is defined as

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

Finally, we will also define a weight metric as done for differential cryptanalysis in Definition 6 and Definition 7. We call the weight metric used in linear cryptanalysis the *correlation weight*. Definition 12 describes how this is defined for a linear approximation.

Definition 12

The *correlation weight of a linear approximation* (u, v) with $\mathrm{LP}_f(u, v) \neq 0$ is given by

$$\mathrm{w_c}(u, v) = -\log_2 \mathrm{LP}_f(u, v)$$

Analogously to Definition 7, Definition 13 defines the weight of a trail.

Definition 13

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

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

2.4 Box Partitioning and Alignment

In this section, we will look at the partition of index spaces defined by N (the nonlinear layer). We need this information to properly answer our research question since "the *alignment* properties of the other step functions with respect to this partition have an important impact on the propagation properties of the round function" [8]. Below, we define what is meant with partitions, give the belonging definitions and talk about what it means for a primitive to be aligned. Again, this is theory from [8] upon which we elaborate.

2.4.1 Nonlinear Layer N

As described in subsection 2.2.5, the nonlinear layer consists of n S-boxes of size m. We can say that N consists of the parallel application of these S-boxes to disjoint parts of the state, which are indexed by B_i. Since there are n S-boxes in one layer, we can write the following:

$$N = S_0 \times \cdots \times S_{n-1}$$

The nonlinear layer N is characterized by

$$P_i \circ (S_0 \times \cdots \times S_{n-1}) = S_i \circ P_i \text{ for } 0 \leq i \leq n-1$$

2.4.2 Box Partitions

We can say that N defines a unique *ordered* partition of the index space $[0, b-1]$ notated as $\Pi_{\mathrm{N}} = (\mathrm{B}_0, \ldots, \mathrm{B}_{n-1})$. Here, the B_i are called N-boxes and Π_{N} is called the *box partition* defined by N. Note that N thus divides the index space into smaller parts that are disjoint and have equal length. If there is no ambiguity regarding N, we write Π_{N} as $\Pi = (\mathrm{B}_0, \ldots, \mathrm{B}_{n-1})$.

The nonlinear layer N just defines one of the possible partitions of the index space $[0, b-1]$. For partitions, see also [20] with the added restriction

that our partitions should be equally-sized. In general, we call a partition *non-trivial* if it has at least two members B_0 and B_1.

Since there is more than one partition, we can define a possible relation between two different partitions, which we will call *refinement*. Basically, some partition Π is a refinement of another partition Π' if every element of Π is a subset of some element of Π'. The mathematical notation for this concept is given in Definition 14.

Definition 14

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

Next to this, we can define what it means for a shuffle layer to be a *Π-shuffle*. Let Π be a partition of the index space $[0, b-1]$ consisting of k members, each of size l. The shuffle layer is a Π-shuffle if the associated permutation matrix can be partitioned into k identity matrices of dimension $(l \times l)$. Simpler said, the shuffle layer is a Π-shuffle if the bits that are grouped together in input box i are all shuffled in the same output box j. In this case, we can specify the *bit index permutation* as a *box index permutation*.

2.4.3 Alignment

We can use the knowledge in the previous section to introduce what it means for a step function ϕ to be *Π-aligned*: a step function is Π-aligned if it respects the boundaries of the boxes defined by the partition Π. Definition 15 describes this notion formally by again considering k members of size l.

Definition 15

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

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

In this case, we call the ϕ_i box functions.

As a result of this, we can define what it means for a whole round function to be aligned. For this, next to Definition 15, we need to combine the knowledge of a non-trivial partition, a Π-shuffle and Definition 14. When a round function is aligned, is explained in Definition 16. Note that this

definition states that the addition ι of a round constant does not influence the alignment properties.

Definition 16

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

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

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

The final type of alignment that we discuss is alignment concerning a primitive (in our case: a permutation). If all of the round functions of a primitive are aligned, we call the primitive *aligned*. Consequently, if this is not the case, we call the primitive *unaligned*.

2.4.4 Superbox Structure

Every aligned primitive has a *superbox* structure [16, 17, 28], that we can use in order to investigate distributions and bounds on the DP of two-round differentials and the LP of two-round trails. Below, we will explain what a superbox structure means.

First, consider a two-round structure defined as $\pi \circ \mathrm{M} \circ \mathrm{N} \circ \pi \circ \mathrm{M} \circ \mathrm{N}$. We notice that the last two steps do not have an effect on the distributions due to the linearity of π and M. Because of this, we simplify the expression to $\mathrm{N} \circ \pi \circ \mathrm{M} \circ \mathrm{N}$ by leaving out those two steps. Another important thing we notice is that $\mathrm{N} \circ \pi$ equals $\pi \circ N'$ with $N' := \pi^{-1} \circ \mathrm{N} \circ \pi$ since we know that $\pi \circ \pi^{-1} = \mathrm{id}$. We can thus rewrite our expression and obtain $\pi \circ N' \circ \mathrm{M} \circ \mathrm{N}$. Again, due to the linearity of π, this step has no effect on the distributions: the expression now equals $N' \circ \mathrm{M} \circ \mathrm{N}$.

Because $\Pi_{\mathrm{N}'} = \Pi_{\mathrm{N}} \leq \Pi_{\mathrm{M}}$ by Definition 16, we can thus see this as a parallel application of a certain number of superboxes: a *superbox layer*. In a sequence of two rounds, we can see $N' \circ \mathrm{M} \circ \mathrm{N}$ as a (composite) nonlinear layer and $\pi \circ \mathrm{M} \circ \pi$ as a (composite) linear layer. If the linear layer is aligned to a non-trivial partition Π such that $\Pi_{\mathrm{M}} \leq \Pi$, then this two-round structure is *aligned* to Π_{M}. In other words: the two-round structure is then Π_M*-aligned*.

2.5 Bit and Box Weight Histograms

In this section, we define what the bit weight and the box weight is and provide their histograms: the bit weight histogram and the box weight histogram. In order to do so, we will need to introduce some other terms like what it means for a bit or box to be active or passive. The notion of these two histograms will be used in chapter 4 to analyze huddling properties. These notations are presented in [8] and are explained in more detail below.

2.5.1 Activity and Weights

Box Activity and Box Weight

We start off by defining what it means for a box to be active. We need an *indicator function* to do so. We define this indicator function as $1_i : \mathbb{F}_2^b \to \mathbb{F}_2$ with respect to a box partition Π by

$$1_i(a) = \begin{cases} 0 & \text{if } P_i(a) = 0 \\ 1 & \text{otherwise} \end{cases}$$

The box B_i is called *active* in the difference or linear mask $a \in \mathbb{F}_2^b$ if we are in the second case of the indicator function, so if $1_i(a) = 1$. Otherwise, if we end up in the first case of the indicator function, we call the box B_i *passive*. So, in other words we can state that a box is active in a difference or linear mask if there are one or more bits in that particular box non-zero.

We can now link the metric *box weight* to this described box activity. We define the box weight, denoted by w_Π, as follows:

$$w_\Pi(a) = \#\{i \in [0, n-1] : 1_i(a) \neq 0\}$$

The box weight is thus equal to the number of active boxes B_i.

Bit Activity and Bit Weight

We can also define what it means for a bit to be active in approximately the same way as the box activity (without the use of an indicator function). The bit i is namely *active* in the difference or linear mask $a \in \mathbb{F}_2^b$ if $a_i = 1$ and *passive* otherwise.

As done before, we can link this kind of activity to some metric: this time to the *bit weight*. We define the bit weight, denoted by w_2, as follows:

$$w_2(a) = \#\{i \in [0, b-1] : a_i \neq 0\}$$

The bit weight is thus equal to the number of active bits i.

Activity Pattern

A last concept that handles activity is the *activity pattern* of $a \in \mathbb{F}_2^b$. It is defined as

$$r_{\Pi}(a) = \sum_{i=0}^{n-1} 1_{\mathrm{B}_i}(a) e_i^n$$

In words this can be explained as a vector whose ith element is 1 if box B_i is active and 0 if it is passive. To clarify this, we take as an example a state that is partitioned in four boxes $\mathrm{B}_0, \mathrm{B}_1, \mathrm{B}_2$ and B_3. Assume that the first two boxes are active and the last two are passive. The activity pattern can then be written down as $(1, 1, 0, 0)$.

2.5.2 Weight Histograms

As stated in [8], we can use the sum of the number of active boxes at the input and output of the linear layer L to say something about the weight of a two-round trail $(q_{\text{in}}, a, b, q_{\text{out}})$ over $\mathrm{N} \circ \mathrm{L} \circ \mathrm{N}$. This weight can namely be bounded from below by that sum of this number. The number of active boxes only depends on a because we have that $b = L(a)$ in differential trails and $a = L^{\top}(b)$ in linear trails.

The distribution of states a according to this number of active boxes determines the *mixing power* of L with respect to Π_{N}. In order to quantify this mixing power, we look at the weight distribution of $(a, L(a))$ over all possible differences or linear masks $a \in \mathbb{F}_2^b$. This can be presented inside a *weight histogram.* Definition 17 shows how we define the *bit weight histogram.*

Definition 17

The (bit) *weight histogram* of a linear transformation $\mathrm{L} : \mathbb{F}_2^b \to \mathbb{F}_2^b$ is a function $\mathcal{N}_{2,\mathrm{L}} : \mathbb{Z}_{\geq 0} \to \mathbb{Z}_{\geq 0}$ given by

$$\mathcal{N}_{2,\mathrm{L}}(k) = \#\left\{a \in \mathbb{F}_2^b : w_2(a) + w_2(\mathrm{L}(a)) = k\right\}$$

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

$$\mathcal{C}_{2,\mathrm{L}}(k) = \sum_{l \leq k} \mathcal{N}_{\mathrm{L}}(l)$$

In a similar way, Definition 18 defines what a *box weight histogram* is.

Definition 18

The (box) *weight histogram* of a linear transformation $\mathrm{L} : \mathbb{F}_2^b \to \mathbb{F}_2^b$ is a function $\mathcal{N}_{\Pi,\mathrm{L}} : \mathbb{Z}_{\geq 0} \to \mathbb{Z}_{\geq 0}$ given by

$$\mathcal{N}_{\Pi,\mathrm{L}}(k) = \#\left\{a \in \mathbb{F}_2^b : w_\Pi(a) + w_\Pi(\mathrm{L}(a)) = k\right\}$$

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

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

The left-most values inside the bit or box weight histogram are called the *tail* of the histogram. This tail corresponds thus to the values representing low weight.

If we have an aligned primitive, these weight histograms can be computed by *convolving* the weight histograms of its box functions. This is the case since the superbox structure of an aligned primitive provides us the possibility to use a divide-and-conquer approach [12]. Concretely, let $S(w) = \left\{v \in \mathbb{Z}_{\geq 0}^s : \sum_{i=0}^{s-1} v_i = w\right\}$ where s is defined to be the number of superboxes. We can then compute the bit weight histograms by using

$$\mathcal{N}_{2,\mathrm{M}}(\mathrm{w}) = \sum_{v \in S(\mathrm{w})} \prod_{i=0}^{s-1} \mathcal{N}_{2,\mathrm{M}_i}(v_i)$$

and the box weight histogram by using

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

For more details, we refer to [8].

2.5.3 Extensions of Branch Numbers

But what are these bit and box weight histograms exactly? As explained in [8], the bit and box weight histograms are, respectively, natural extensions of the bit and box branch numbers. We can read off the *differential branch number* [14, 31] by looking at the smallest non-zero entry of the histogram of L. Analogously, we can read off the *linear branch number* [14, 31] by looking at the smallest non-zero entry of the histogram of $\mathrm{L}^\top$.

The differential and linear branch number do not have to be the same. This is the case for our SKINNY cipher. As a result, we will thus make a distinction between the *differential bit weight*, the *linear bit weight*, the *differential box weight* and the *linear box weight*.

When we have extracted both branch numbers from the weight histograms, we can state something about the mixing power. This is the case since a higher branch number typically implies a higher mixing power and analogously a lower branch number implies a lower mixing power. Next to this, the histograms also show a more nuanced comparison of the mixing layers than just comparing the branch numbers. An important application of this can be seen in the box weight histogram. Take a linear layer L and another linear layer L' with the same nonlinear layers. If L has systematically lower values in the tail of the box weight histogram than L', we can expect that L has fewer two-round trails with low weight than L'. Note that we can thus still say something useful about a comparison of L and L' even though the branch numbers might be the same.

2.5.4 Ordered Weight Pairs

After having seen the bit and box weight (histograms) and their branch numbers, we define another term related to these concepts. The *ordered bit and box weight pairs* are (ordered) tuples of which the x-coordinate represents the input bit or box weight and the y-coordinate represents the output bit or box weight. Here, we talk about the input and output of the mixing layer `MixColumns`. In Definition 19 we define these weight pairs formally.

Definition 19

We call $(\mathrm{w}_{2,\mathrm{in}}, \mathrm{w}_{2,\mathrm{out}})$ the *ordered bit weight pair* where $\mathrm{w}_{2,\mathrm{in}}$ represents the input bit weight and $\mathrm{w}_{2,\mathrm{out}}$ represents the output bit weight.

Analogously, we call $(\mathrm{w}_{\Pi,\mathrm{in}}, \mathrm{w}_{\Pi,\mathrm{out}})$ the *ordered box weight pair* with $\mathrm{w}_{\Pi,\mathrm{in}}$ the input box weight and $\mathrm{w}_{\Pi,\mathrm{out}}$ the output box weight.

In chapter 4 we will look at the distributions of these ordered weight pairs for both SKINNY64 and SKINNY128.

2.6 Two-round Trail Weight Histograms

In this section, we define two other weight histograms: the two-round differential trail weight histogram and the linear trail weight histogram using respectively the restriction weight (differential trail weight) and the correlation weight (linear trail weight) as defined in subsection 2.3.1 and subsec-

tion 2.3.2. These histograms are used in chapter 4 to relate the phenomenon huddling to the distribution of trail weights. Again, these definitions come from [8] and are elaborated upon below.

To define the differential trail weight histogram, we look again at Definition 17 and Definition 18. The *two-round differential trail weight histogram* is defined similarly as in previous definitions with the exception that we define $\mathcal{N}$ and $\mathcal{C}$ differently. Instead of looking at the bit or box weight, we look at the restriction weight as defined in Definition 7. In this definition we also look at a trail Q instead of a linear transformation L, see Definition 20. In this definition we use the symbol X for the set of all trails Q.

Definition 20

The (two-round differential trail) *weight histogram* of the trail Q is a function $\mathcal{N}_{\mathrm{r}} : \mathbb{Z}_{\geq 0} \to \mathbb{Z}_{\geq 0}$ given by

$$\mathcal{N}_{\mathrm{r}}(k) = \#\left\{Q \in X : \mathrm{w_r}(Q) = k\right\}$$

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

$$\mathcal{C}_{\mathrm{r}}(k) = \sum_{l \leq k} \mathcal{N}_r(l)$$

After having defined the two-round differential trail weight histogram, we can also define its linear version. The *two-round linear trail weight histogram* is the same as Definition 20, but instead of using the restriction weight, we use the correlation weight as described in Definition 13. The definition of this weight histogram can be seen in Definition 21.

Definition 21

The (two-round linear trail) *weight histogram* of the trail Q is a function $\mathcal{N}_{\mathrm{c}} : \mathbb{Z}_{\geq 0} \to \mathbb{Z}_{\geq 0}$ given by

$$\mathcal{N}_{\mathrm{c}}(k) = \#\left\{Q \in X : \mathrm{w_c}(Q) = k\right\}$$

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

$$\mathcal{C}_{\mathrm{c}}(k) = \sum_{l \leq k} \mathcal{N}_c(l)$$

Note that we again used the symbol X for the set of all trails Q. Here, the same reasoning as in subsection 2.5.3 still holds: the lower the tail, the better, since this means that there are less states with small weights.

The way to compute the two-round differential trail weight histogram and the two-round linear trail weight histogram for aligned primitives also did not change compared to what we explained in subsection 2.5.2. We can namely still convolve the trail weight histograms of the superbox structure. In mathematical notation, we then thus get that we can compute the differential trail weight histogram by using

$$\mathcal{N}_{\mathrm{r,M}}(\mathrm{w}) = \sum_{v \in S(\mathrm{w})} \prod_{i=0}^{s-1} \mathcal{N}_{\mathrm{r,M}_i}(v_i) \tag{4}$$

and the linear trail weight histogram by using

$$\mathcal{N}_{\mathrm{c,M}}(\mathrm{w}) = \sum_{v \in S(\mathrm{w})} \prod_{i=0}^{s-1} \mathcal{N}_{\mathrm{c,M}_i}(v_i) \tag{5}$$

2.7 Cluster Histograms

In this section, we define what a cluster histogram is and provide the theory needed for the establishment of this definition. The notion of a cluster histogram will be used in chapter 5 to analyze clustering. This histogram is introduced in [8] upon which we will shortly elaborate.

2.7.1 Equivalence Classes

In order to build up the definition of a cluster histogram, we first look into equivalence classes as described in subsection 2.2.2. We define two different kinds of equivalences, namely *box activity equivalence* and *cluster equivalence*. Definition 22 starts off with defining what it means to be box activity equivalent. Hereby, the definition of an activity pattern is used as introduced in subsection 2.5.1.

Definition 22

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

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

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

When two states are box activity equivalent, we can say something about trail cores. This is the case, since box activity equivalence has a purpose in the relation between trail cores and differentials. It has also a use in the relation between trail cores and linear approximations. This relation is formulated in Lemma 1.

Lemma 1

Two trail cores $(a_0, b_0, \ldots, a_{r-2}, b_{r-2})$ and $\left(a_0^*, b_0^*, \ldots, a_{r-2}^*, b_{r-2}^*\right)$ over a function $f = \mathrm{N}_{r-1} \circ \mathrm{L}_{r-2} \circ \mathrm{N}_{r-2} \circ \cdots \circ \mathrm{L}_0 \circ \mathrm{N}_0$ that are in the same differential (or linear approximation) satisfy $a_0 \sim a_0^*$ and $b_{r-2} \sim b_{r-2}^*$.

In other words, this states that if two trail cores are in the same differential or linear approximation, that then the first and the last intermediate differences are box activity equivalent. This proof is based on the symmetry and transitivity properties of equivalence relations. For the exact proof of this lemma, we refer to [8].

Let us focus on the case $r = 2$ in Lemma 1. We can now refine the definition of box activity equivalence to a definition of cluster equivalence, which was the second kind of equivalence to discuss. Definition 23 gives the exact notion of when two states are cluster equivalent.

Definition 23

Two states are *cluster equivalent* with respect to a linear mapping $\mathrm{L} : \mathbb{F}_2^b \to \mathbb{F}_2^b$ and a box partition Π if they are box activity equivalent before L and after it (see Figure 4):

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

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

Note that the size of the cluster classes is important for the notion of clustering, since the size of these classes upper bounds the amount of possible trail clustering.

2.7.2 Cluster Histogram

Now, if we apply Lemma 1 to the case $r = 2$, we see that both $a \sim a^*$ and $L(a) \sim L(a^*)$ should hold. But after applying Definition 23, we also know that it follows that $a \approx a^*$. This result is drawn up in Corollary 1.

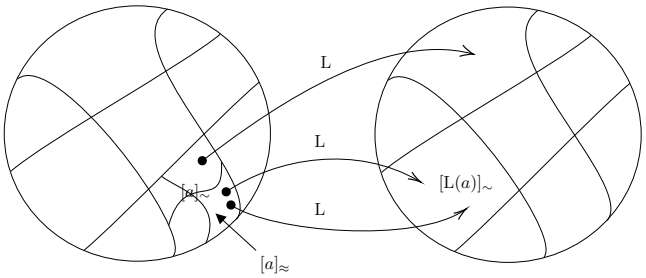


Figure 4: Partitions of $\mathbb{F}_2^b$ defined by $\sim$ and $\approx$ [8].

Corollary 1

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

What this corollary shows is that the defining differences of any two-round trail cores that cluster together end up in the same cluster class. If these cluster classes are small, we say that there is little clustering. Analogously, if the cluster classes are bigger, there is more clustering.

As we defined the bit and box weight in subsection 2.5.1, we can define yet another weight. This weight is defined for an element in the cluster class. We say that for all $a' \in [a]_\approx$, the box weight $\mathrm{w}_{\Pi}(a') + \mathrm{w}_{\Pi}(\mathrm{L}(a'))$ is the same and denote this weight by $\tilde{\mathrm{w}}([a]_\approx)$.

If we combine the knowledge of this weight with all the other information above, we can finally define what a *cluster histogram* is. This is described in Definition 24.

Definition 24

Let $\mathrm{L} : \mathbb{F}_2^b \to \mathbb{F}_2^b$ be a linear transformation. Let $\approx$ be the equivalence relation given in Definition 23. The *cluster histogram* $N_{\Pi,\mathrm{L}} : \mathbb{Z}_{\geq 0} \times \mathbb{Z}_{\geq 0} \to \mathbb{Z}_{\geq 0}$ of L with respect to the box partition Π is given by

$$N_{\Pi,\mathrm{L}}(k, c) = \# \left\{ [a]_\approx \in \mathbb{F}_2^b / \approx \colon \tilde{\mathrm{w}}\left([a]_\approx\right) = k \wedge \#[a]_\approx = c \right\}$$

Here, / means 'with respect to', the parameter k functions as a placeholder for a certain box weight and the parameter c represents the number of states. For a fixed box weight, so for a fixed k, this histogram shows the distribution of sizes of the cluster classes with that certain value of k.

2.8 Two-round Trail Clustering

In this section, we introduce two final types of weight histograms, namely the restriction weight histogram and the correlation weight histogram. These two histograms are also introduced in [8], look like the four other histograms previously defined and will be elaborated upon below. We use these two histograms in chapter 5 to discuss the two-round trail clustering.

In order to define the *restriction weight histogram*, we take a closer look at Definition 20. The only difference compared to the two-round differential trail weight histogram is that instead of looking at the trail Q, we look at the differential $(\Delta_{\text{in}}, \Delta_{\text{out}})$. Therefore, we can define the restriction weight histogram as given in Definition 25.

Definition 25

The (restriction) *weight histogram* of the differential $(\Delta_{\text{in}}, \Delta_{\text{out}})$ is a function $\mathcal{N}_{\text{r}} : \mathbb{Z}_{\geq 0} \to \mathbb{Z}_{\geq 0}$ given by

$$\mathcal{N}_{\text{r}}(k) = \#\left\{(\Delta_{\text{in}}, \Delta_{\text{out}}) \in \mathbb{F}_2^b \times \mathbb{F}_2^b : \text{w}_{\text{r}}((\Delta_{\text{in}}, \Delta_{\text{out}})) = k\right\}$$

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

$$\mathcal{C}_{\text{r}}(k) = \sum_{l \leq k} \mathcal{N}_r(l)$$

The same reasoning holds for the *correlation weight histogram*: it is the same as Definition 21, but then we change the trail Q to the linear approximation (u, v). The mathematical notation can be seen in Definition 26.

Definition 26

The (correlation) *weight histogram* of the linear approximation (u, v) is a function $\mathcal{N}_{\text{c}} : \mathbb{Z}_{\geq 0} \to \mathbb{Z}_{\geq 0}$ given by

$$\mathcal{N}_{\text{c}}(k) = \#\left\{(u, v) \in \mathbb{F}_2^b \times \mathbb{F}_2^b : \text{w}_{\text{c}}((u, v)) = k\right\}$$

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

$$\mathcal{C}_{\text{c}}(k) = \sum_{l \leq k} \mathcal{N}_c(l)$$

Again, we can compute these histograms for aligned primitives by convolving the restriction and correlation weight histograms of the superbox structure. The mathematical notation belonging to the computation of the restriction and correlation weight histograms is similar as given in section 2.6: the computation of the restriction weight histogram is similar as in equation (4) and the computation of the correlation weight histogram is the same as in equation (5).

Chapter 3

Block Cipher Family SKINNY

3.1 Description of SKINNY

As the title of this thesis and name of this chapter already reveal, SKINNY [1] is a certain block cipher family. According to the designers of SKINNY, its goal is "to compete with NSA's recent design SIMON in terms of hardware/software performances, while proving in addition much stronger security guarantees with regards to differential/linear attacks" [1]. In this section, we will present the different instances of this block cipher family, describe SKINNY's round function and look at its alignment properties.

3.1.1 Instances of SKINNY

The lightweight block cipher family SKINNY consists of six versions: three versions with a block size of 64 bits and three versions with a block size of 128 bits [1]. Table 1 gives an overview of these six different versions.

Table 1: The six different instances of SKINNY [1, 2].

	Block size ($= n$)	Tweakey size	# rounds
SKINNY-64-64	64	64 (n)	32
SKINNY-64-128	64	128 ($2n$)	36
SKINNY-64-192	64	192 ($3n$)	40
SKINNY-128-128	128	128 (n)	40
SKINNY-128-256	128	256 ($2n$)	48
SKINNY-128-384	128	384 ($3n$)	56

Since we assume a fixed-key perspective in this paper, we note that the tweakey size is not important for our research. We thus consider two different versions of SKINNY, which we from now on call SKINNY64 (the 64-bit version) and SKINNY128 (the 128-bit version).

SKINNY64 and SKINNY128 have a similar so called *internal state*. "In both $n = 64$ and $n = 128$ versions, the internal state is viewed as a 4×4 square array of cells, where each cell is a nibble (in the $n = 64$ case) or a byte (in the $n = 128$ case)" [1]. Note that a byte consists of 8 bits, whereas a nibble - also called a half-byte - only consists of 4 bits. A visual representation of the internal state is shown in Figure 5.

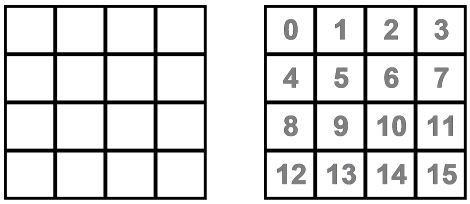


Figure 5: (a) The internal state of SKINNY64 and SKINNY128 represented as a 4×4-matrix build up from 16 nibbles respectively 16 bytes. (b) The same internal state as shown in Figure 5(a) but then with cell numbering.

3.1.2 SKINNY's Round Function

Encryption

In order to encrypt some plaintext using SKINNY64 or SKINNY128, we apply the same round function a certain number of times as given in Table 1. This round function, as shown in Figure 6, consists of five operations: `SubCells`, `AddConstants`, `AddRoundTweakey`, `ShiftRows` and `MixColumns` [1]. In the next sections we will describe what each operation is doing to the internal state.

`SubCells`

The nonlinear layer inside the round function is called `SubCells`. $\Pi_{\texttt{SubCells}}$ partitions the state into the sixteen cells as shown in Figure 5. Depending on whether we consider SKINNY64 or SKINNY128, we use respectively a 4-bit S-box ($\mathcal{S}_4$) or an 8-bit S-box ($\mathcal{S}_8$) in this layer [1]. A schematic overview of how this layer uses these S-boxes can be seen in Figure 7.

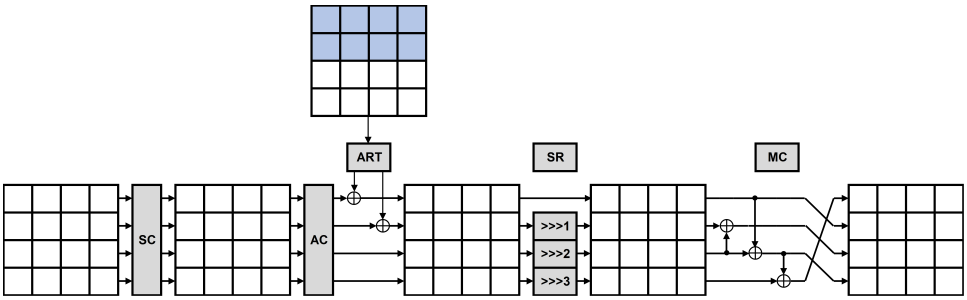


Figure 6: The round function of Skinny64 and Skinny128. Here, we used the following abbreviations: `SC` for `SubCells`, `AC` for `AddConstants`, `ART` for `AddRoundTweakey`, `SR` for `ShiftRows` and `MC` for `MixColumns`. The blue rows represent the XORed rows of the round tweakey in each round [1, 2, 23].

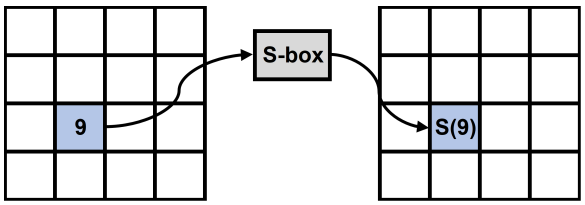


Figure 7: Schematic overview of the `SubCells` operation. Here, S is used to show the S-box as a function on the 9^{th} cell.

We can represent both $\mathcal{S}_4$ and $\mathcal{S}_8$ with NOR and XOR operations as can be seen in Figure 8. Next to this, we can also show these S-boxes in hexadecimal notation. These notations are included in Appendix A (section A.2).

`AddConstants`

Skinny's round constant addition inside the round function is called `AddConstants`. To generate the round constants, a 6-bit affine Linear-Feedback Shift Register (LFSR) is used [1]. This is a shift register of which the input bit is a linear function of its preceding state. After having a freshly generated round constant, this step involves the XORing of "three round constants to the first three cells of the first column of an internal state" [23].

`AddRoundTweakey`

The round tweakey addition of both Skinny64 and Skinny128 is called `AddRoundTweakey`. The first and second row of all tweakey arrays, so half a block from the tweakey state, are extracted and XORed to the ciphers internal state, respecting the array positioning [1, 2]. After addition, the arrays are updated as depicted in Figure 9.

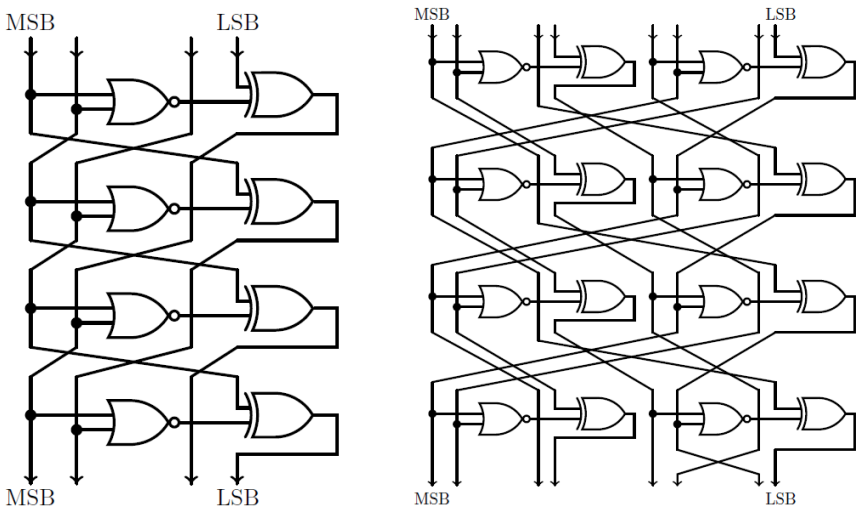


Figure 8: (a) $\mathcal{S}_4$: the 4-bit S-box of SKINNY64 [2]. Here, MSB means the most significant bit and LSB the least significant bit. (b) $\mathcal{S}_8$: the 8-bit S-box of SKINNY128 [2]. MSB and LSB have the same definitions as in Figure 8(a).

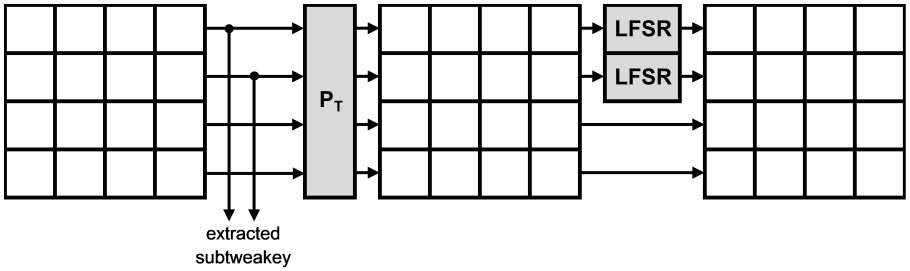


Figure 9: The tweakey schedule to update the tweakey state. Here, P_T is the permutation $(0, \ldots, 15) \mapsto (9, 15, 8, 13, 10, 14, 12, 11, 0, 1, 2, 3, 4, 5, 6, 7)$ and the LFSRs are only used on the second and/or third tweakey array(s) when we have a tweakey size of $2n$ or $3n$ (as shown in Table 1) [1].

`ShiftRows`

The shuffle layer inside the round function is named `ShiftRows`. It is a $\Pi_{\texttt{SubCells}}$-shuffle as described in subsection 2.4.2. If we look at this sub-operation for SKINNY64, we rotate the nibbles in a row of the internal state to the right. In particular, the first, second, third and fourth cell rows are rotated by 0, 1, 2 and 3 positions to the right [1]. For SKINNY128 this operation is the same. However, instead of rotating nibbles we rotate bytes. A schematic overview of this layer can be seen in Figure 10.

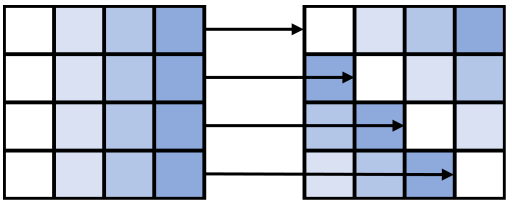


Figure 10: Schematic overview of the `ShiftRows` operation.

`MixColumns`

The mixing layer of both SKINNY64 and SKINNY128 is called `MixColumns`. This mixing layer is schematically illustrated in Figure 11. In this step, every column of the ciphers internal state, consisting of either nibbles respectively bytes, is multiplied by the binary matrix shown on the left side [1]. When discussing the linear propagation properties, we need to multiply with the transposed version of this matrix. This matrix is shown on the right side. When we use the transposed version of this binary matrix, we notate the transposed mixing layer as $\texttt{MixColumns}^\top$.

$$\begin{pmatrix} 1 & 0 & 1 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 0 \end{pmatrix} \qquad \begin{pmatrix} 1 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 1 \\ 1 & 0 & 0 & 0 \end{pmatrix}$$

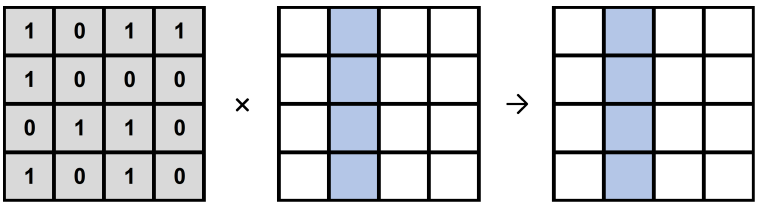


Figure 11: Schematic overview of the `MixColumns` operation. Here, the gray (4×4)-grid represents the binary matrix.

Decryption

Next to encrypting the plaintext, we naturally also want to decrypt the ciphertext that has been encrypted before. For this, we use the same round function. However, instead of using the operations described above, the inverses of all operations are used [1].

3.1.3 SKINNY's Alignment Properties

By Definition 15, the mixing layer of both SKINNY64 and SKINNY128 is aligned to a non-trivial partition $\Pi_{\texttt{MixColumns}}$. This partition corresponds to

the 4 columns, each containing either 4 nibbles or 4 bytes. Next to this, we can use Definition 14 to conclude that $\Pi_{\texttt{SubCells}} \leq \Pi_{\texttt{MixColumns}}$. It follows from Definition 16 and the definition of alignment of a primitive that both Skinny64 and Skinny128 are aligned. Figure 12 shows the steps and their alignment properties.

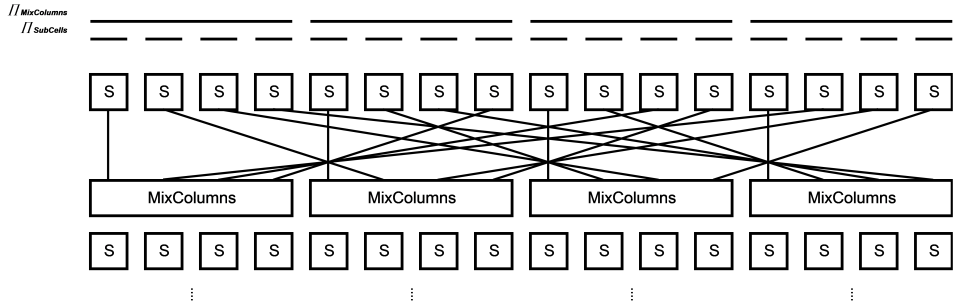


Figure 12: Alignment properties of Skinny64 and Skinny128. The lines in this figure represent the right shifts from the `ShiftRows` layer.

Comparison to Rijndael, Saturnin, Spongent and Xoodoo

The alignment properties of the ciphers Rijndael, Saturnin, Spongent and Xoodoo were investigated in [8]. Table 2 compares the alignment of those ciphers with the two instances of Skinny.

Table 2: Alignment comparison.

Cipher	Aligned?
Rijndael	Yes
Saturnin	Yes
Skinny64	Yes
Skinny128	Yes
Spongent	Yes
Xoodoo	No

3.2 Round Cost

After getting familiar with Skinny64 and Skinny128, we are going to look at the implementation cost of their round function and compare these to the implementation cost of the ciphers discussed in [8]. We look at three layers of the round function in order to determine this cost, namely the S-box layer, the mixing layer and the shuffle layer.

3.2.1 S-Box Layer

The S-box layer of the two instances of SKINNY is `SubCells`, as described in section 3.1.2. We know that this operation uses S-boxes that are invertible. [1] states that the S-boxes of both SKINNY64 and SKINNY128 have a maximum differential probability of 2^{-2} and also a maximum absolute linear bias of 2^{-2}. In this case, the absolute linear bias corresponds with the linear potential as defined in Definition 11 [8, 30]. To see this, take the agreements A for the number of x for which $u^\top x = v^\top f(x)$ and the disagreements D for the number of x for which $u^\top x \neq v^\top f(x)$. We rewrite Definition 11 as follows:

$$\begin{aligned}
\mathrm{LP}_f(u,v) &= \mathrm{C}_f(u,v)^2 \\
&= \left(\frac{1}{2^b} \sum_{x \in \mathbb{F}_2^b} (-1)^{u^\top x + v^\top f(x)} \right)^2 \\
&= \left(\frac{1}{2^b} \left(A - D \right) \right)^2 \\
&= \left(\frac{1}{2^b} \left(A - \left(2^b - A \right) \right) \right)^2 \\
&= \left(\frac{1}{2^b} \left(2A - 2^b \right) \right)^2 \\
&= \left(\frac{1}{2^b} \cdot 2 \left(A - 2^{b-1} \right) \right)^2 \\
&= \left(2 \left(\frac{A}{2^b} - \frac{1}{2} \right) \right)^2
\end{aligned}$$

Next to this, we know that the absolute linear bias [30] is defined as

$$\left| \frac{A}{2^b} - \frac{1}{2} \right|$$

Knowing that $\left| \frac{A}{2^b} - \frac{1}{2} \right| = 2^{-2}$ gives us that $\frac{A}{2^b} - \frac{1}{2} = 2^{-2}$ or $\frac{A}{2^b} - \frac{1}{2} = -2^{-2}$. Filling this back into our rewritten version of $\mathrm{LP}_f(u,v)$ gives us

$$\begin{aligned}
\mathrm{LP}_f(u,v) &= \left(2 \left(\pm 2^{-2} \right) \right)^2 \\
&= \left(\pm 2^{-1} \right)^2 \\
&= 2^{-2}
\end{aligned}$$

and thus, we see that indeed the absolute linear bias corresponds with the linear potential as stated.

Note thus that Skinny64 has the lowest known maximum DP and LP values regarding its width [8]. In comparison to the smaller version, Skinny128 does not achieve this. We know that 2^{-2} is not the lowest possible value. This would be 2^{-6}, which leads to not being sure if the implementation cost will increase by width [8].

As done in [8], we report on the implementations of both versions of Skinny with the minimum number of binary XOR, binary AND/OR and unary NOT operations as found in [1].

Next to this, the minimal sum-of-products (SOP) form in Boolean algebra of the S-boxes is determined. The Espresso algorithm, a heuristic logic minimizer [13, 25, 27], was used in order to do this. In Appendix A we provide the SOP expressions for both Skinny64 and Skinny128. After having found the SOP forms, we can use De Morgan's laws [29] to turn these expressions into a two-layer NAND gate circuit as explained in [8]. Note that this is a theoretical two-layer NAND gate circuit since in practice a single NAND gate cannot have, for instance, 19 inputs.

Comparison to Rijndael, Saturnin, Spongent and Xoodoo

In [8] the minimal number of binary XOR, the minimal number of binary AND/OR, the minimal number of unary NOT, the SOP form and the number of NAND gates per bit for each of the S-boxes are determined of the ciphers Rijndael, Saturnin, Spongent and Xoodoo. As subsection 3.2.1 states, we did the same for both instances of Skinny. A comparison of the first three concepts can be seen in Table 3 and a comparison of the number of NAND gates per bit is illustrated in Table 4.

Table 3: S-box computational cost comparison using the number of operations in $\mathbb{F}_2$ [8].

		Operations in $\mathbb{F}_2$			
	Max	# operations			
Cipher	DP/LP	Reference(s)	XOR	AND/OR	NOT
Rijndael	2^{-6}	[9, 32]	81	32	4
Saturnin	2^{-2}	[11]	6	6	-
Skinny64	2^{-2}	[1]	4	4	4
Skinny128	2^{-2}	[1]	8	8	8
Spongent	2^{-2}		?		
Xoodoo	2^{-2}	[15]	3	3	3

Table 4: S-box computational cost comparison using a 2-layer NAND circuit [8].

	2-layer NAND circuit																								
	# NAND gates per # inputs (2 through 23 inclusive)																							Totals	
Cipher	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	Gates	Inputs
Rijndael	?																						?		
Saturnin	4	5	6	1	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	16	52
Skinny64	5	1	8	1	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	15	50
Skinny128	7	17	8	18	20	9	2	-	-	-	-	-	-	1	-	-	-	1	-	-	1	1	85	465
Spongent	-	6	8	-	3	1	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	18	75
Xoodoo	3	6	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	9	24

In Table 3 we note that Skinny64 looks the most like Xoodoo and that Skinny128 has the most similarities with Saturnin in terms of the number of operations in $\mathbb{F}_2$. Since "the number of AND/OR operations is related to the cost of masking countermeasures" [8] we can state that Skinny64 takes half the cost of masking countermeasures for Skinny128. We can also conclude that this cost for Skinny64 is 8 times cheaper than the cost of Rijndael and that the cost for Skinny128 is 4 times cheaper than the cost of Rijndael.

We see in Table 4 that the cost of the Skinny64 S-box is comparable with the cost of the S-boxes of the ciphers Saturnin and Spongent. The cost of Skinny128 is roughly about 5 times bigger than that. However, we should note that Skinny128 is 8 bits wide instead of 4 bits like Skinny64. Since these numbers give an indication for the size of a hardware circuit and the number of cycles in bit-sliced software implementations [8] we actually pay a quite high price for Skinny128; especially when we recall that this cipher does not have the lowest known maximum DP and LP values.

3.2.2 Mixing Layer

The mixing layer of Skinny is `MixColumns`, as elaborated upon in section 3.1.2. For Skinny64 "the `MixColumns` layer requires 3 XOR gates to update 4 bits, thus amounting to 0.75 XOR per bit of internal state" [1]. This also holds for Skinny128 since we can also say that we require 3 XOR gates to update 8 bits but we need twice as many matrices, which results in having 0.75 XOR per bit of the internal state. Below we will give the explanation on how we end up at 0.75 XOR per bit.

We give a reasoning by looking at the binary (4×4)-matrix inside this sub-operation. The second row of this matrix has only one 1, which indicates that this does not cost anything. This means that the second row of the new state will simple be the first row of the old state. We see that the third and fourth row both contain two times a 1, which indicates that this costs 1 XOR for 4 bits. The third row of the new state will thus be the result of XORing the second and third row of the old state. Similarly, the fourth row of the new state will be the result of XORing the first and third row of the old state. Lastly, we see that even though the first row in the binary matrix has three 1's in it, we only have to use 1 XOR for 4 bits. This is the case since we can XOR the fourth row of the *new* state with the fourth row of the *old* state. In total this thus gives an average of 3 XOR operations for 4 bits, resulting in 0.75 XOR per bit. Note that we thus have a circuit depth of 2 XOR gates.

Another way to see this at a glance is to look at Figure 6. Under the gray block containing `MC`, we see three different XOR symbols drawn. These thus also indicate that we have 0.75 XOR per bit.

To emphasize that both Skinny64 and Skinny128 result in 0.75 XORs per bit, we can give explicit calculations. We know that Skinny64 has a total state of 64 bits, that it needs 3 XORs and has 16 input bits for the `MixColumns` operation. A calculation then shows that we have $\frac{16 \times 3}{64} = 0.75$ XOR per bit. Similarly, we know that Skinny128 has a total state of 128 bits, that it needs 3 XORs and has 32 input bits for the `MixColumns` operation. We then get that there are $\frac{32 \times 3}{128} = 0.75$ XOR per bit needed.

Comparison to Rijndael, Saturnin, Spongent and Xoodoo

The cost of the mixing layer of Rijndael, Saturnin, Spongent (non-existent) and Xoodoo is investigated in [8]. Table 5 shows a comparison of the cost of the mixing layer of both instances of Skinny to those other ciphers.

Table 5: Mixing layer computational cost comparison [8].

Cipher	Reference	# operations per bit	Type of operation
Rijndael	[22]	$97/32 \approx 3$	addition
Saturnin	[11]	2.25	XOR
Skinny64	[1]	0.75	XOR
Skinny128	[1]	0.75	XOR
Spongent	[7]	0	-
Xoodoo	[15]	2	XOR

We see that the cost for Skinny64 and Skinny128 are comparable to each other (as stated above) but not comparable to any of the others. We see that this cost is roughly 3 times lower than the implementation cost regarding Saturnin and Xoodoo and a lot lower than the cost of Rijndael.

3.2.3 Shuffle Layer

The sub-operation `ShiftRows` is Skinny's shuffle layer as described in section 3.1.2. In hardware, this operation is just a simple wiring between gates, which is considered to be a negligible cost. Therefore, we measure this cost through a software implementation. In [8], the assembly code was executed on an ARM Cortex-M4 processor. Since the two instances of Skinny are mainly hardware-oriented [1] and since there is no known optimized assembly code for this particular processor, we cannot determine this cost. Writing optimized assembly code for the ARM Cortex-M4 is considered to be out-of-scope for this thesis.

Comparison to Rijndael, Saturnin, Spongent and Xoodoo

As stated in subsection 3.2.3, the implementation cost of the shuffle layer is measured by using an ARM Cortex-M4 processor. Table 6 shows these results with the two entries for Skinny64 and Skinny128 added.

Table 6: The cost of a round in cycles per byte on the ARM Cortex-M4 [8].

Cipher	Reference	# cycles per byte
Rijndael	[35]	10.0
Saturnin	[10]	2.7
Skinny64		?
Skinny128		?
Spongent		?
Xoodoo	[3]	1.1

Chapter 4

Huddling

In this chapter, we will discuss the phenomenon *huddling* [8]. In order to do this, we present the bit and box weight histograms, compare them to the bit and box weight histograms of the four ciphers discussed in [8] and show the distribution of the ordered weight pairs. After this, we also look into the relation between huddling and the distribution of trail weights. This is again followed up by a comparison of our SKINNY instances to the other four ciphers.

4.1 Making SKINNY Instances Larger

In this section, we will look at so-called two-round structures: the application of the round function twice. For an aligned cipher this simply means that a number of superboxes is applied in parallel. For SPONGENT this is 24 times a 16-bit superbox; for SATURNIN this is 16 times a 16-bit superbox and for RIJNDAEL this is 8 times a 32-bit superbox. However, for SKINNY64 this means that we apply 4 times a 16-bit superbox and for SKINNY128 this is 4 times a 32-bit superbox. Since all the histograms as defined in chapter 2 take on a shape that is determined by the total width (the wider the width, the higher the histograms), it is important to increase the number of superboxes in our SKINNY instances before we start comparing our instances to these other ciphers.

So, for the sake of comparing SKINNY64 and SKINNY128 correctly with RIJNDAEL, SATURNIN, SPONGENT and XOODOO investigated in [8], we decided to define a version of SKINNY operating on 384 bits. This version thus has a state size of 384 bits instead of 64 or 128 bits: the same state size as SPONGENT and XOODOO. We decided to choose 384 bits instead of 256 bits (the state size of RIJNDAEL and SATURNIN) since in the comparison of bit weights SKINNY64 was closest to XOODOO and in the comparison of

box weights it was closest to SPONGENT. To make both comparisons more accurate it thus made more sense for us to use a state size of 384 bits.

From now on, we call the version of SKINNY64 with a 384-bit state SKINNY64-384 and the version of SKINNY128 operating on 384 bits is called SKINNY128-384.

Recall that we also have a difference in the linear and differential bit and box weights for these instances, as mentioned in subsection 2.5.3, for which we need a naming convention. If we address both the differential and linear bit or box weights of any instance of SKINNY, we simply use the names defined above. If we are talking about the linear bit or box weight of any instance of SKINNY, we show this with LIN in the name. Analogously, if we want to state something about the differential bit or box weight of any instance of SKINNY, we denote this by using DIF in the naming.

4.2 Bit Weight Histograms

We start off by discussing the cumulative bit weight histograms for the linear layers of all SKINNY ciphers and ciphers investigated in [8], including the identity permutation that consists of 4-bit S-boxes. We provide the differential and linear bit weight histograms of only SKINNY instances in Figure 13 and a comparison of SKINNY64-384 and SKINNY128-384 with the other four ciphers in Figure 14. Next to this, the differential and linear bit weight histograms of the superboxes of only SKINNY instances are shown in Figure 15 and the histograms of the superboxes of all other ciphers except XOODOO and SPONGENT are depicted in Figure 16.

The first thing we note is that both the linear bit weight histograms of SKINNY64-384 and SKINNY128-384 and also differential bit weight histograms of these instances are the same. This is because both instances have the same state size and have the same linear layer $\mathrm{L} = \pi \circ \mathrm{M} = \texttt{ShiftRows} \circ \texttt{MixColumns}$ and thus also the same $\mathrm{L}^\top = \pi \circ \mathrm{M}^\top = \texttt{ShiftRows} \circ \texttt{MixColumns}^\top$. For both instances, L can be seen as a parallel application of a 4×4-matrix on 4-bit sub-strings. We will see a difference between SKINNY64-384 and SKINNY128-384 when we also apply the S-box. This does play a role in computing the box weight and will be done in section 4.3.

However, we do see a difference when looking at the histograms of the superboxes of both SKINNY64 and SKINNY128. This makes sense if we relate this to our previous observation: the state of SKINNY64 is smaller than that of SKINNY128 with the same L and $\mathrm{L}^\top$.

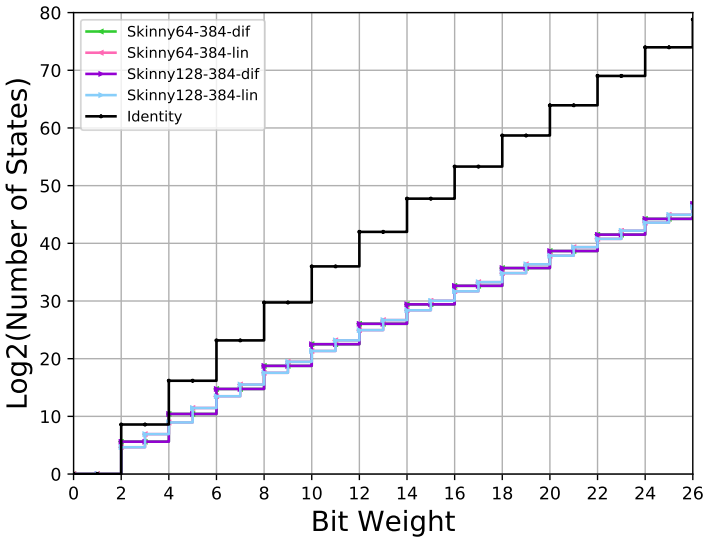


Figure 13: Cumulative (linear and differential) bit weight histograms of Skinny64-384 and Skinny128-384.

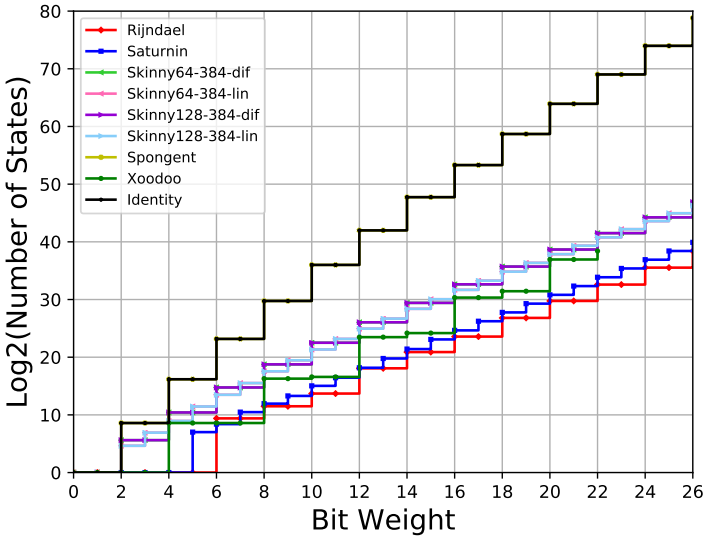


Figure 14: Cumulative (linear and differential) bit weight histograms of all the ciphers investigated in this thesis and in [8].

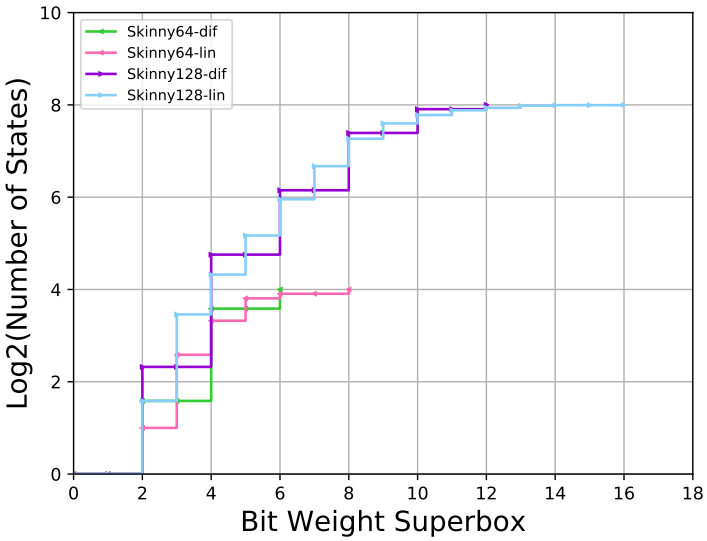


Figure 15: Cumulative (linear and differential) bit weight histograms of the superbox of Skinny64 and Skinny128 (without the identity permutation).

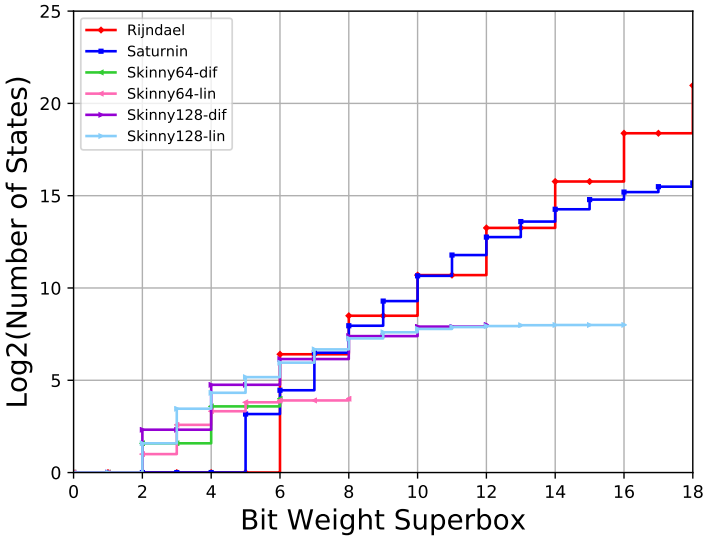


Figure 16: Cumulative (linear and differential) bit weight histograms of the superbox of all the ciphers investigated in this thesis and in [8] (without the identity permutation and except for Xoodoo and Spongent).

The bit branch number of the linear layers of Skinny64-384 and Skinny128-384 are the same, namely 2. This is the case because we consider a linear permutation: a non-zero difference always goes to a non-zero difference and zero always goes to zero. This means that one active S-box

will always go to at least one active S-box, resulting in the lowest possible bit branch number 2. Next to this, the bit branch number of the linear layers of Skinny64-384 and Skinny128-384 is equal to the bit branch number of `SpongentMixLayer` of Spongent and also the identity permutation. This means that also the mixing power of all different instances of Skinny is the lowest possible.

What we see in Figure 14 and Figure 16 is that, just like the bit weight histograms of Spongent, Rijndael and Xoodoo, the bit weight histograms of Skinny64-384-dif and Skinny128-384-dif have only non-zero entries at even bit weights. This means that the linear layers can be modeled as $a \mapsto (\mathrm{I} + \mathrm{M})a$ for some matrix $\mathrm{M} \in \mathbb{F}_2^{b \times b}$ with the property that the bit weight of $\mathrm{M}a$ is even for all $a \in \mathbb{F}_2^b$ [8]. We see that this is not the case for Skinny64-384-lin and Skinny128-384-lin as it was not the case for Saturnin. We can deduce this from the transpose of the binary matrix used in $\texttt{MixColumns}^\top$, since we only have an odd number of ones in every row.

Another observation is that, similar to Xoodoo, Skinny64-384 and Skinny128-384 have higher bit weight histograms than Rijndael and Saturnin. The histograms of the two Skinny versions are in their turn somewhat higher than that of Xoodoo.

We note that the overall comparison of the bit weight histograms of the different ciphers links back to the computational resources invested in the mixing layer [8]. The higher the bit branch number of the mixing layer of a particular cipher, the more invested computational resources in this layer. Although the bit branch numbers of the linear layer of Spongent, Skinny64-384 and Skinny128-384 are the same, we still see this relationship occurring. Spongent namely has a higher tail than the two versions of Skinny from which we can see that Spongent has invested less computational resources. To illustrate this relation, we provide a scatter plot as shown in Figure 17. Here, the red line represents the average estimated number of operations per bit belonging to a certain bit branch number based on our known data points.

4.2.1 Distribution of Ordered Bit Weight Pairs Skinny64

After having compared the different bit weight histograms in general, we are going to look at the distribution of the ordered bit weight pairs (as defined in Definition 19) of Skinny64 in more detail.

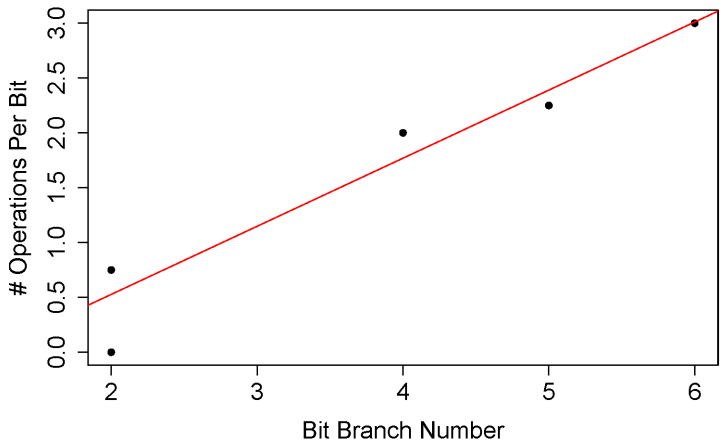
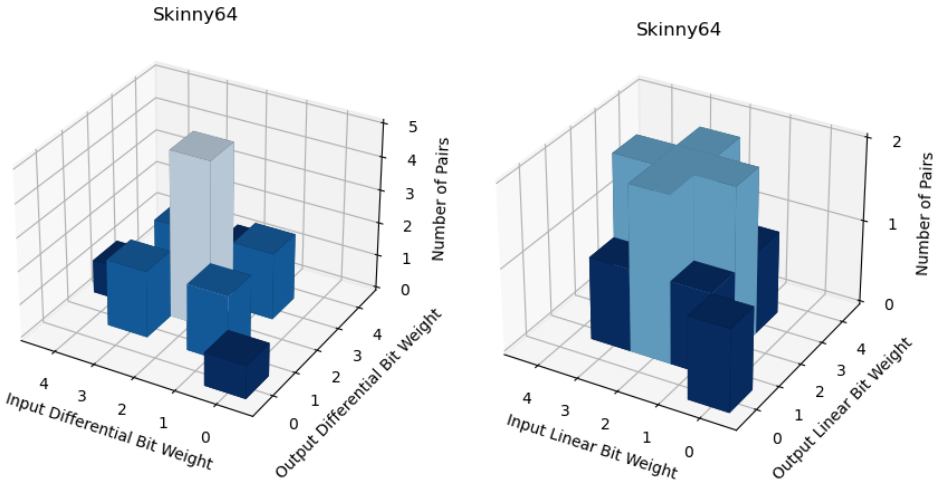


Figure 17: Relation between the bit branch number and the computational cost of the mixing layer.

We provide these distributions by means of two-dimensional histograms in Figure 18. In this figure, the center of the bar represents the position of the weights in the grid. The lighter the blue color, the more pairs of this particular bit weight pairs there are in comparison to the other bit weight pairs. Analogously, the darker the blue color, the lesser pairs of a certain $(w_{2,in}, w_{2,out})$-pair.



(a) **(b)**

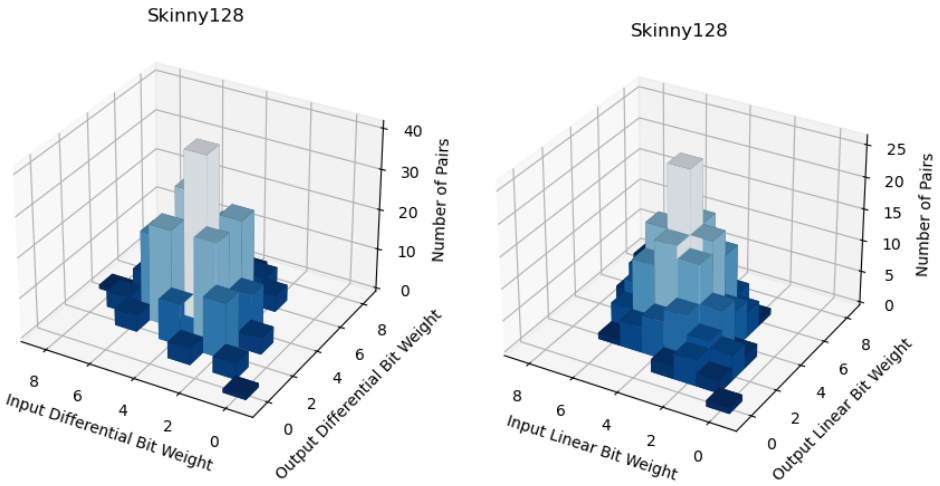
Figure 18: (a) Two-dimensional histogram that shows the distribution of the ordered $(w_{2,in}, w_{2,out})$-pairs when the binary matrix of `MixColumns` is applied to 4-bit vectors. (b) Two-dimensional histogram that shows the distribution of the ordered $(w_{2,in}, w_{2,out})$-pairs when the transposed binary matrix of $\texttt{MixColumns}^\top$ is applied to 4-bit vectors.

We see that both the differential and linear bit weight pairs for SKINNY64 are centered around (2,2). We also see that both two-dimensional histograms are symmetric. In order to see this better and read of the exact values displayed, top and side views along with tables of these histograms are provided in Appendix B (section B.1).

Another thing we can see in Figure 18(a) is already mentioned and explained in section 4.2. We can namely see from this two-dimensional histogram that SKINNY64-DIF has only non-zero entries at even bits in the bit weight histogram since the x and y-coordinates added are always an even number. Likewise, in Figure 18(b) we can also easily see that this does not hold for SKINNY64-LIN since we have, for instance, a bar at (2,1).

4.2.2 Distribution of Ordered Bit Weight Pairs SKINNY128

We will now discuss the distribution of ordered bit weight pairs of SKINNY128 in a similar way as described in subsection 4.2.1. The two-dimensional histogram of SKINNY128-DIF is shown in Figure 19(a) and the one of SKINNY128-LIN is depicted in Figure 19(b).



(a) **(b)**

Figure 19: (a) Two-dimensional histogram that shows the distribution of the ordered $(w_{2,in}, w_{2,out})$-pairs when the binary matrix of `MixColumns` is applied to 8-bit vectors. (b) Two-dimensional histogram that shows the distribution of the ordered $(w_{2,in}, w_{2,out})$-pairs when the transposed binary matrix of $\texttt{MixColumns}^\top$ is applied to 8-bit vectors.

From these two-dimensional histograms we read off that the differential bit weight pairs for SKINNY128 are concentrated around (4,4). This also holds for the linear bit weight pairs. Again, we see two symmetric histograms

of which the top and side views along with tables are given in Appendix B (section B.3).

Just as stated in subsection 4.2.1, we can see that the two-dimensional histogram of Skinny128-dif has only non-zero entries at even bits in the bit weight histogram and that the bit weight histogram of Skinny128-lin has not. This observation is justified, since these two-dimensional histograms are actually the same histograms as the ones of Skinny64 but then convolved with themselves. This is because of the reason discussed in section 4.2. If this would not be the case, the cumulative bit weight histograms of Skinny64-384 and Skinny128-384 would not overlap in Figure 13 and Figure 14.

4.3 Box Weight Histograms

Next, we talk about the cumulative box weight histograms for the linear layers of all instances of Skinny and compare these to the four ciphers of [8]. Again, we include the identity permutation that assumes 4-bit S-boxes. In Figure 20 we illustrate the differential and linear box weight histograms of Skinny. In Figure 21 we compare Skinny64-384 and Skinny128-384 with the other ciphers. In Figure 22 we show the differential and linear box weight histograms of the superboxes of the Skinny instances and in Figure 23 we compare these with the box weight histograms of the superboxes of Rijndael, Saturnin and Spongent.

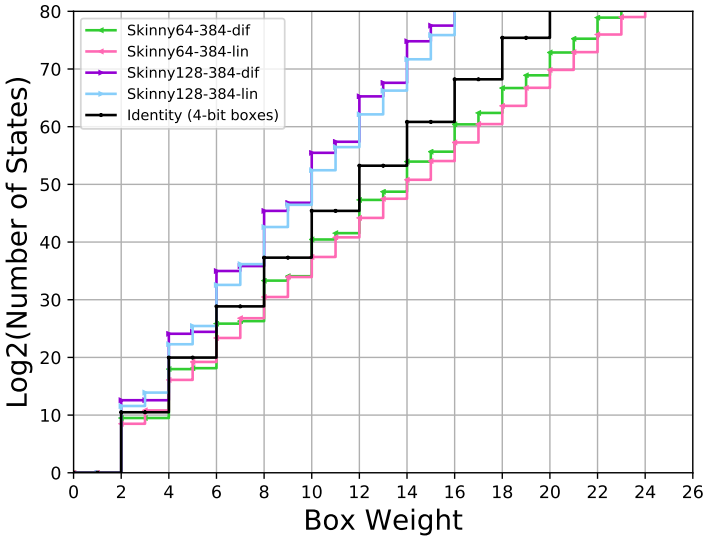


Figure 20: Cumulative (linear and differential) box weight histograms of Skinny64-384 and Skinny128-384.

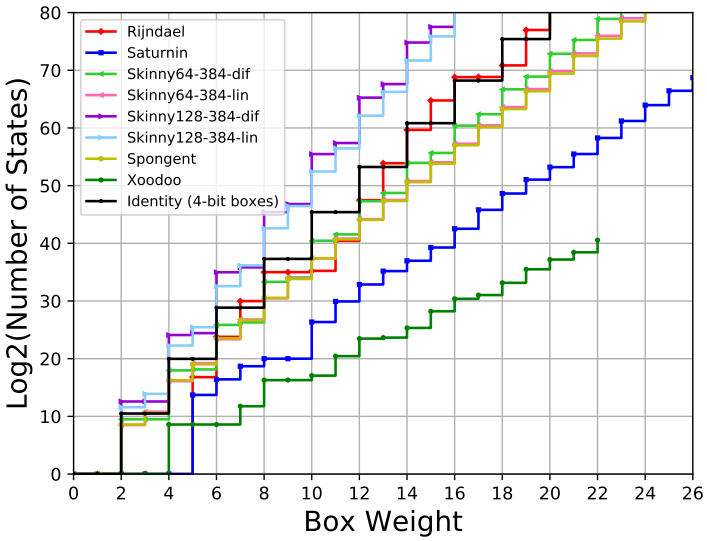


Figure 21: Cumulative (linear and differential) box weight histograms of all the ciphers investigated in this thesis and in [8].

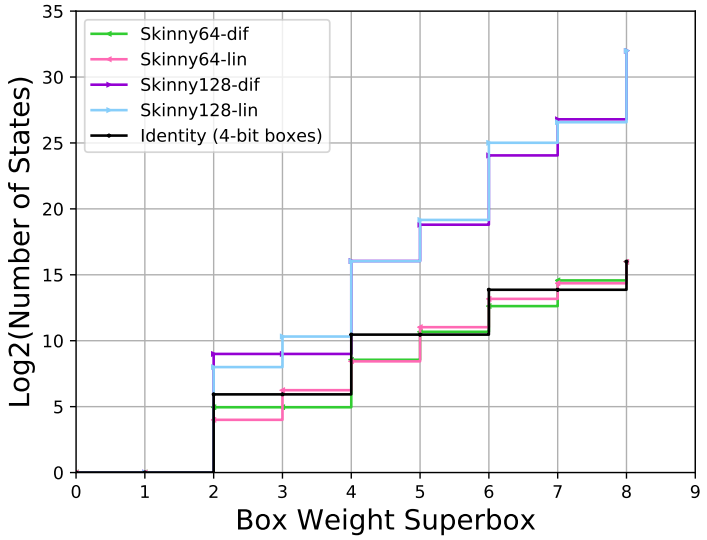


Figure 22: Cumulative (linear and differential) box weight histograms of the superbox of Skinny64 and Skinny128.

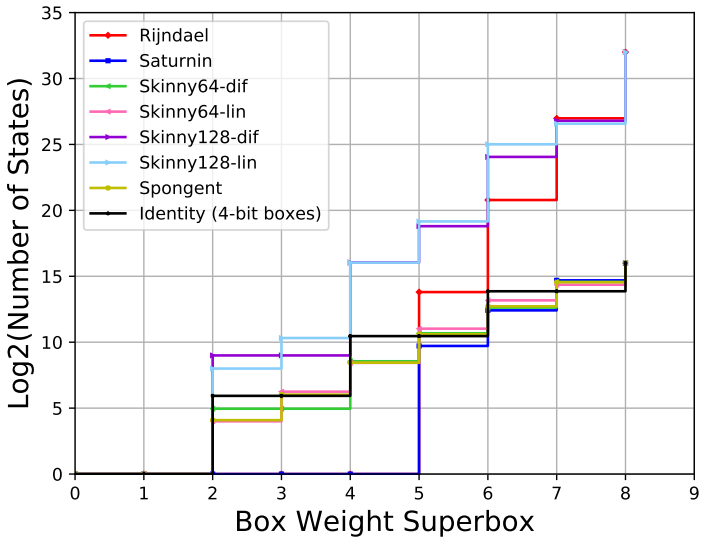


Figure 23: Cumulative (linear and differential) box weight histograms of the superbox of all the ciphers investigated in this thesis and in [8] (except for Xoodoo).

If we look at Skinny64-384 and Skinny128-384 in Figure 20, we see that the box weight histogram of the smaller instance has a lower tail than the bigger instance. If we compare this to the bit weight histograms, we now thus see a difference between our two versions with a 384-bit state.

Another thing we can see is that the box branch numbers of all instances of Skinny are equal to 2: the same as the bit branch numbers. We see the same thing happening for Saturnin, Spongent and Xoodoo but not for Rijndael (here we go from the bit branch number 6 to the box branch number 5). Even though, for example, Spongent has a lower tail in the box weight histogram compared to the bit weight histogram, Skinny64-384 and Skinny128-384 have a higher tail than its bit weight variant.

Overall, we can see that Skinny128-384 is located higher than any other cipher investigated. For the linear layer of Skinny64-384 we can say that the box weight histograms are quite similar to the box weight histogram of the linear layer of Spongent, especially the linear box weight histogram. The differential box weight histogram is located slightly higher than these two histograms.

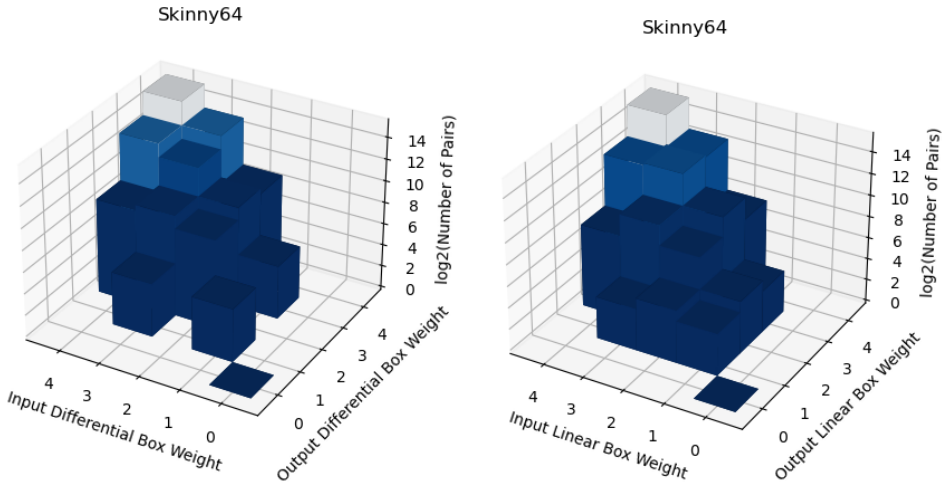
If we look at Figure 22 and Figure 23, we see that this in some way looks like the tail of the cumulative linear and differential box weight histograms. Note that the histograms of Skinny64 and Skinny128 are the same as the histograms of Skinny64-384 and Skinny128-384, since we do not increase the size of the S-boxes. In these figures, we see that the box weight histogram

of the superbox of Skinny128 is really close to the histogram of Rijndael. Likewise, we see that the histogram of the superbox of Skinny64 is very similar to the ones of Saturnin and Spongent.

All previous observations point towards a discrepancy between the bit and box weight histograms, which leads us to two notions concerning huddling: *bit huddling* and *superbox huddling* [8]. We explain what this means and provide more details in section 4.4, but now we are first going to look into the distribution of the ordered box weight pairs of both Skinny64 and Skinny128.

4.3.1 Distribution of Ordered Box Weight Pairs Skinny64

As we know from Definition 19, the ordered box weight pairs are defined analogously to the ordered bit weight pairs. In Figure 24, we show the distributions of the ordered differential and linear box weight pairs by again using two-dimensional histograms. The color scheme is used in the same way as explained in subsection 4.2.1. As also described in this subsection, we use the center of the bar for the positioning of the weights in the grid.



(a) **(b)**

Figure 24: (a) Two-dimensional histogram that shows the distribution of the ordered $(w_{\Pi,\text{in}}, w_{\Pi,\text{out}})$-pairs of the superbox using the differential box weight of Skinny64. (b) Two-dimensional histogram that shows the distribution of the ordered $(w_{\Pi,\text{in}}, w_{\Pi,\text{out}})$-pairs of the superbox using the linear box weight of Skinny64.

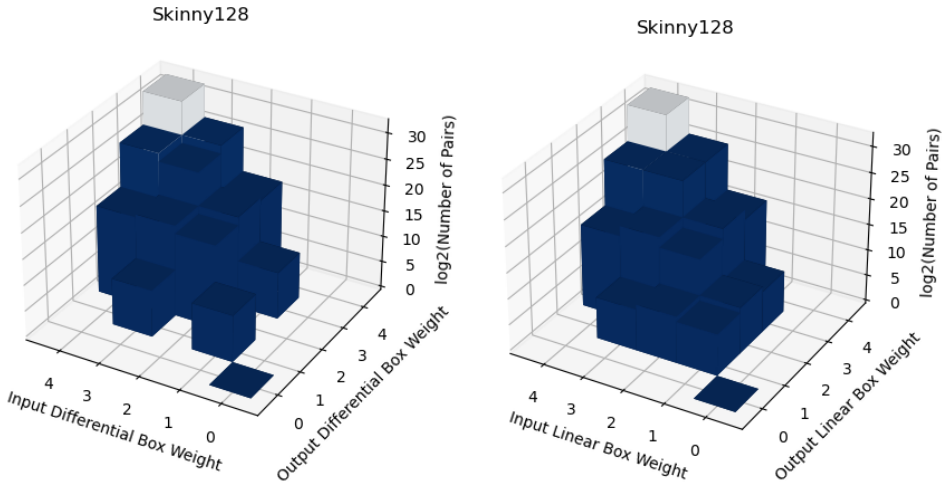
The histogram of Skinny64-dif looks more similar to Skinny64-lin this time compared to the two-dimensional histogram that showed the ordered bit weight pairs. To be concrete, the positions of the bars in these

figures only differ at the coordinates (2,1) and (1,2). To see this better, top and side views along with tables of these two-dimensional histograms are provided in Appendix B (section B.2).

We also observe that both the differential and linear box weight pairs for Skinny64 are concentrated around (4,4) and take on a symmetric distribution. The centering around (4,4) is stronger for the linear $(w_{\Pi,\text{in}}, w_{\Pi,\text{out}})$-pairs than for the differential box weight pairs.

4.3.2 Distribution of Ordered Box Weight Pairs Skinny128

The last distribution we will discuss is the distribution of ordered box weight pairs of Skinny128. We do this in the similar way as done in subsection 4.3.1. We show the two-dimensional histogram of Skinny128-dif in Figure 25(a) and the one of Skinny128-lin in Figure 25(b).



(a) **(b)**

Figure 25: (a) Two-dimensional histogram that shows the distribution of the ordered $(w_{\Pi,\text{in}}, w_{\Pi,\text{out}})$-pairs of the superbox using the differential box weight of Skinny128. (b) Two-dimensional histogram that shows the distribution of the ordered $(w_{\Pi,\text{in}}, w_{\Pi,\text{out}})$-pairs of the superbox using the linear box weight of Skinny128.

For Skinny128 we see approximately the same happening as for Skinny64. The two different histograms again look very similar with the exception of the $(w_{\Pi,\text{in}}, w_{\Pi,\text{out}})$-pairs on (2,1) and (1,2), see Appendix B (section B.4).

The symmetrical two-dimensional histograms for Skinny128-dif and Skinny128-lin are centered around (4,4), just like the ones discussed in subsection 4.3.1. We see that, in comparison to those two histograms, there

are relatively a lot more pairs with an input and output box weight equal to 4. This can for instance be seen by the colors: we go from white at (4,4) immediately to dark blue if we look at any other coordinate in the grid.

4.4 Huddling

In this section, we go into more detail on the discrepancy between the bit and box weight histograms as already mentioned in section 4.3. This first leads us to *bit huddling* (see subsection 4.4.1). After this, in subsection 4.4.2, we look into a consequence of this for aligned primitives called the *superbox huddling effect*.

4.4.1 Bit Huddling

Bit huddling is defined as the phenomenon in which "many active bits huddle together in a few active boxes" [8]. The bit huddling is *high* if there are a lot of active bits huddled together in a few active boxes. It is *low* if we do not have such a dense concentration of active bits.

This kind of huddling has an effect on the contribution of states a to the bit and box weight histograms: by definition we have that $\mathrm{w}_{\Pi}(a) + \mathrm{w}_{\Pi}(\mathrm{L}(a)) \leq \mathrm{w}_2(a) + \mathrm{w}_2(\mathrm{L}(a))$ [8]. Concretely, this means that bit huddling causes the states a to go to the left in the box weights histograms. In other words, the linear layers of ciphers in these histograms have a higher tail than in their corresponding bit weight histograms. Because the tail of the box weight histogram goes up, we find a reduction of mixing power at box level in comparison to bit level.

In order to not encounter this decrease in mixing power at box level, we would want to avoid huddling. In this ideal situation we would thus see the exact same bit and box weight histogram for a particular cipher. However, we cannot fully eliminate huddling because of how boxes are structured. That is, there are states that have more than one active bit inside a box, which will automatically cause bit huddling.

As we will discuss in the next section, the bit huddling of Skinny128 is high, whereas that of Skinny64 is relatively low. This is because Skinny128 has wider S-boxes than Skinny64: our bigger instance has an 8-bit S-box instead of a 4-bit S-box. In order to state this, we will compare the bit weight histograms of the superboxes in Figure 16 with the box weight histograms of the superboxes in Figure 23. For clarity, we will depict them after each other below.

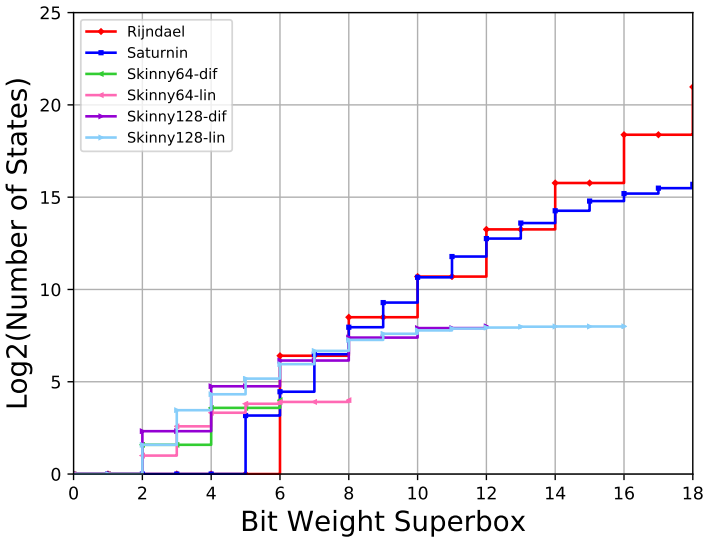


Figure 26: Cumulative (linear and differential) bit weight histograms of the superbox as depicted in Figure 16.

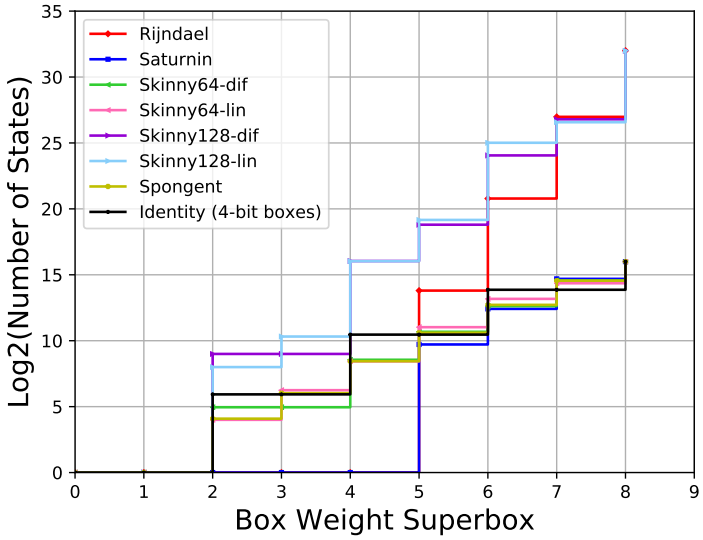


Figure 27: Cumulative (linear and differential) box weight histograms of the superbox as depicted in Figure 23.

4.4.2 Superbox Huddling Effect

As already discussed in subsection 2.4.4, we can talk about superboxes at the moment a primitive is aligned. These superboxes bring along a concept when we discuss huddling, namely the superbox huddling effect. We use this

term to indicate that the bit huddling is more pronounced when the mixing layer only mixes bits within a certain superbox of the state.

As discussed in [8], this superbox huddling effect is very pronounced in Rijndael. This also holds for Skinny128, which has an S-box size of 8 as well, although this effect is still slightly less than Rijndael. We can deduce this from Figure 26 and Figure 27: the branch number does not decrease, whereas the tail does rise a lot. This makes sense, since the branch number cannot decrease under the number 2. The branch number is already at its lowest possible value.

For Skinny64, we can state from Figure 26 and Figure 27 that the impact of this effect is smaller. The branch number again vacuously does not decrease and the tail does rise. However, this is a significant lower rise. The reason why this increase of the tail is less, is because of the smaller S-box size of 4. This size allows for less bit huddling as was also the case for Saturnin. If we compare these ciphers, we state that Skinny64 has more bit huddling than Saturnin. This is because the box weight histogram of the linear layer of Skinny64 is considerably higher than the one of the linear layer of Saturnin.

Note that these observations are in line with what is shown in especially Figure 27. We see that the box weight histograms of the superbox of Rijndael and Skinny128 are similar and that the box weight histograms of the superbox of Saturnin, Skinny64 and Spongent are also located very closely to each other. We also see that the first two ciphers have a higher tail in this figure than the latter three: this exactly corresponds to the effect of superbox huddling.

4.5 Two-round Trail Weight Histograms

We now move on to the two-round trail weight histograms. Unfortunately, because of insufficient computational power available, we could not determine the differential and linear trail weight of the full state nor of the superbox of Skinny128. However, we could compute the differential and linear trail weight histograms of both the superbox and full state of Skinny64. All the computed histograms of the full state are shown, together with the differential and linear trail weight histograms of Rijndael, Saturnin, Spongent and Xoodoo in Figure 28 and Figure 30. The same histograms but then of the superboxes are shown in Figure 29 and Figure 31.

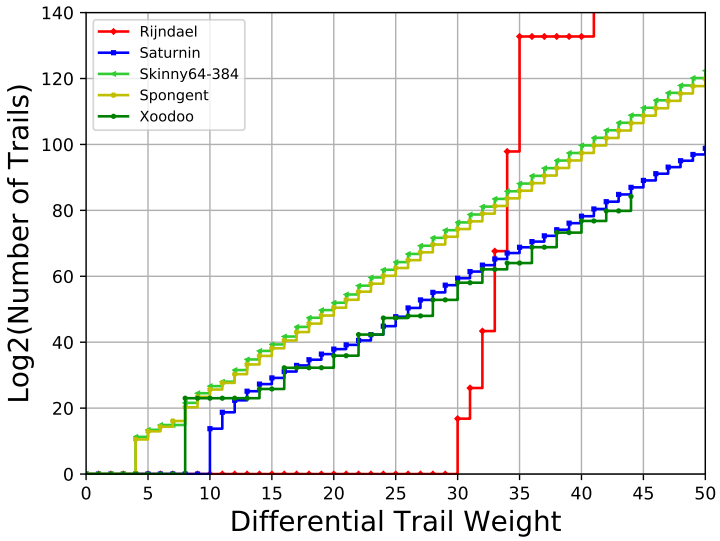


Figure 28: Two rounds: cumulative differential trail weight histograms.

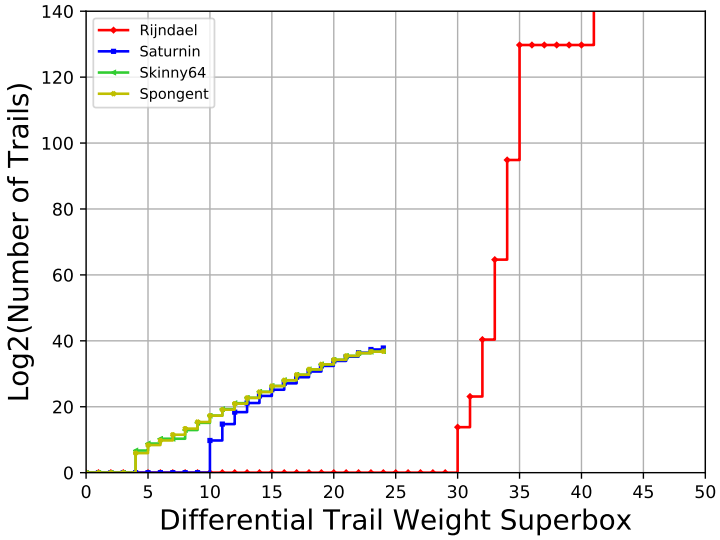


Figure 29: Two rounds: cumulative differential trail weight histograms of the superbox.

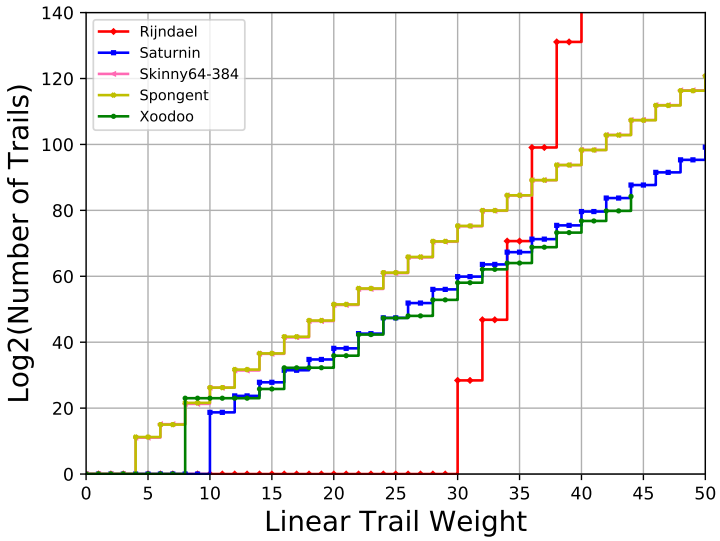


Figure 30: Two rounds: cumulative linear trail weight histograms.

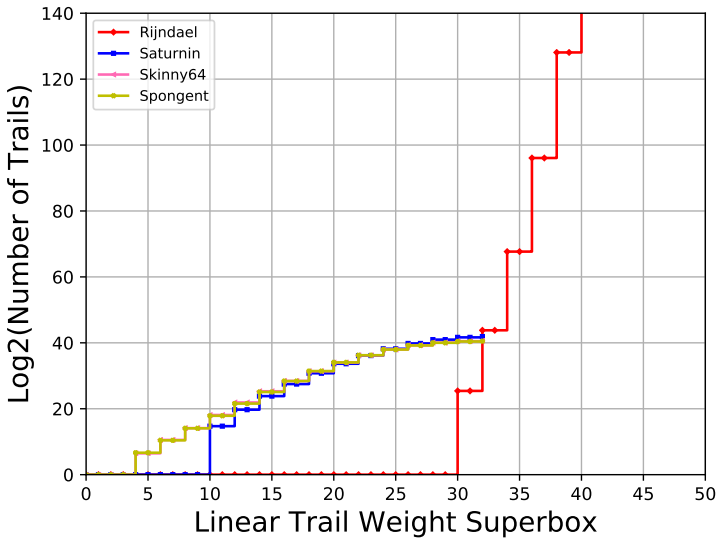


Figure 31: Two rounds: cumulative linear trail weight histograms of the superbox.

As stated in [8], the distribution of the linear trails of Rijndael is not exact. For, for instance, the differential trail weight of Skinny128 (which we unfortunately could not run), we also encountered a difficulty which first let us to rounding off values: not all our $U_f(a, b)$ values (see Definition 1) were of the form 2^x for $0 < x \leq 8$ such that taking the $\log_2$ would not give integers as result. However, this rounding problem was solved by com-

puting the differential trail weight in a different way. We first computed the $U_f(a, b)$ value for the S-box, convolved it to the size of a superbox, and then computed the corresponding weight by first dividing it by the total size and then calculating the corresponding restriction weight. For SKINNY64 we used another approach to first compute the restriction weight for an S-box directly from $U_f(a, b)$ and then convolve it. In the end, we thus theoretically managed to compute exact distributions of the differential and linear trails of both instances of SKINNY if we had enough resources: just like the other ciphers except for RIJNDAEL from [8].

In Figure 28, we see that SKINNY64 performs the worst regarding the differential trail weight. This is also the case for the linear trail weight as we can see in Figure 30. However, this histogram coincides with the histogram of SPONGENT, making them both under-performers compared to the other ciphers. Even though SKINNY64 performs the worst, this does not change the observation that the relative ranking does not change in moving from the box weight to the trail weights, as stated in [8].

If we compare Figure 29 with Figure 28, we see that not much changes. SKINNY64 has still the highest tail. However, what we do notice is that both SKINNY64 and SPONGENT are located way closer to SATURNIN if we just look at the differential trail weight histograms of the superboxes of these ciphers.

Lastly, if we look at Figure 31, we see that this does not really differ from what we could conclude from Figure 30. Again, the histogram of SKINNY64 coincides with the histogram of SPONGENT and has linear trails from correlation weight 4 and onward. Also, as we similarly stated for Figure 29, the linear trail weight histogram of SATURNIN is closer to the ones of SKINNY64 and SPONGENT if we look at the histograms of the superbox instead of to the ones of the full state.

Chapter 5

Clustering

In this chapter, we will discuss *clustering* of differential and linear trails [8]. To do so, we first present the partitions of Skinny64 and Skinny128 by giving their cluster histogram. Following this, we look into the two-round trail clustering and compare this to the two-round trail clustering for the ciphers discussed in [8].

5.1 Cluster Histograms

In this section, we present the cluster histograms of the superboxes of L and $L^\top$ (as explicitly written down in section 4.2) of both our Skinny instances. The cluster histograms of Skinny64 can be seen in Table 7 and Table 8. In Table 9 and Table 10 we show the cluster histograms of Skinny128.

In these histograms, $\tilde{w}$ denotes the weight as explained in subsection 2.7.2, "C represents the cardinality of a cluster class and N shows the number of cluster classes with that cardinality" [8]. In other words, the first entry of Table 7, (2×15), states that there are 2 clusters classes of cardinality 15.

Table 7: The cluster histogram of L of Skinny64.

$\tilde{w}$	$\#\ N \times C$
2	(2×15)
4	(8×15) (1×225)
5	(6×210)
6	(4×210) (4×225) (1×2940)
7	(4×2940) (2×3150)
8	(1×41160)

Table 8: The cluster histogram of $L^\top$ of Skinny64.

$\tilde{w}$	$\#\ N \times C$
2	(1×15)
3	(4×15)
4	(4×15) (1×210)
5	(2×15) (6×210) (2×225)
6	(5×210) (1×225) (2×2940)
7	(4×2940)
8	(1×44535)

Table 9: The cluster histogram of L of Skinny128.

$\tilde{w}$	$\# N \times C$
2	(2×255)
4	(8×255) (1×65025)
5	(6×64770)
6	(4×64770) (4×65025) (1×16451580)
7	(4×16451580) (2×16516350)
8	(1×4178701320)

Table 10: The cluster histogram of $L^{\top}$ of Skinny128.

$\tilde{w}$	$\# N \times C$
2	(1×255)
3	(4×255)
4	(4×255) (1×64770)
5	(2×255) (6×64770) (2×65025)
6	(5×64770) (1×65025) (2×16451580)
7	(4×16451580)
8	(1×4195282695)

We see in all the cluster histograms of the superboxes of both Skinny64 and Skinny128 that the states a are never in their own cluster class. There are large cluster equivalence classes for all weights for Skinny64, but there are even larger cluster equivalence classes for Skinny128. Especially for Skinny128, this means that there are quite some $b \in [a]_{\sim}$ such that $b \neq a$ for which $L(a)$ and $L(b)$ are in the same box activity class. In other words: there are quite some $b \in [a]_{\approx}$.

Ideally, for small box weights, the cluster classes are all very small. This is the case, since large cluster classes of small weights may lead to two-round trails with a large DP or LP [8]. Combining this with our previous knowledge on Skinny128 and the cluster histograms given, we can deduce from this that this cipher instance is expected to have two-round trails with a large DP and LP. Unfortunately, due to insufficient resources to run the programs of Skinny128, we cannot defend or refute this statement.

5.2 Two-round Trail Clustering

In this section, we look into the comparison of two-round differentials with differential trails and the comparison of linear approximations with linear trails of Skinny64, just as done for Saturnin, Spongent and Xoodoo in [8]. We compare our Skinny instance to these ciphers by means of the cumulative restriction and correlation weight histograms.

We start off by discussing the number of differentials and differential trails in the superbox of Skinny64. The differentials are shown in red and the differential trails are shown in blue in Figure 32. In this figure, the differentials are depicted from just above weight 2 up to and including weight 15, whereas the differential trails start at weight 4 and adopt even higher weights (which are displayed up to 24).

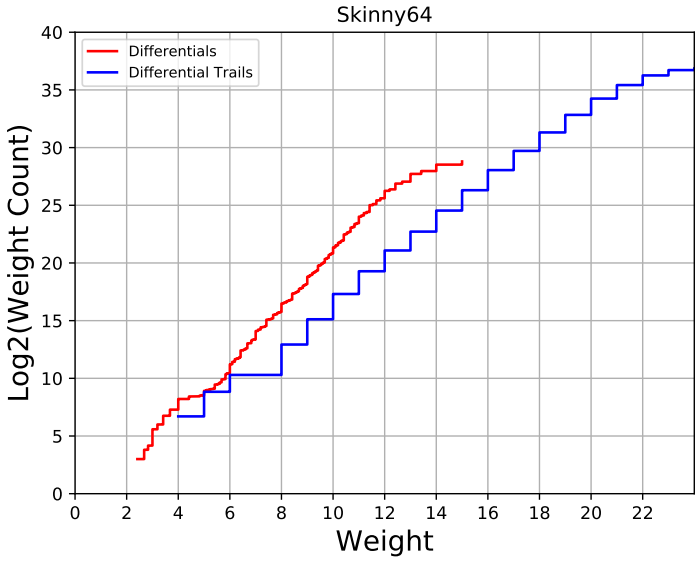


Figure 32: Differentials and differential trails in the superbox of SKINNY64.

In Figure 32, we can see that the histogram of the differentials and the histogram of the differential trails touch at two places: around weight 5 and at weight 6. What we also note is that after the weight 6, the red line stays at roughly the same distance to the blue line. Between the weights 10 and 14, the histograms are the furthest away from each other. What we also see, is that even though the lowest weight of the differential trails equals 4, the differentials lower this weight quite a bit. Considering the differentials, we can see that the lowest weight is actually around 2.5, decreasing it by a weight of 1.5.

Another thing we can read off from these histograms is that there are approximately 5 times more differentials than differential trails when we look at the weights from 0 to 15. The occurrence of more differentials than differential trails is caused by two phenomena. The first one is clustering. Since we have many differential trails but just a fixed width for the superboxes, the trails will cluster into one of those boxes. The second phenomenon that causes the difference is called *clipping* [8], which states that the weight of differentials cannot go above 15. We see this clearly happening in Figure 32: the red graph does not go beyond this weight. This thus tells us that the most differentials have a weight equal to 15 and a corresponding DP of 2^{-15}. "This is the result of the fact that any differential over a superbox has an even number of ordered pairs and hence the minimum DP is 2^{-15}, yielding weight 15" [8].

After having looked at the number of differentials and differential trails in the superbox of Skinny64, we are moving on to discussing the cumulative restriction and correlation weight histograms as explained in section 2.8. The weight histograms for the two-round differentials of Skinny64, Saturnin, Spongent and Xoodoo are shown in Figure 33. The weight histograms for the two-round linear approximations of the same four ciphers are shown in Figure 35. Next to these, we also show the cumulative restriction and correlation weight histograms of the superbox of Skinny64, Saturnin and Spongent in respectively Figure 34 and Figure 36.

If we compare Figure 33 with Figure 28, we see that the clustering of Skinny64 influences the graph quite a lot. Even though the trail weight histogram of Skinny64 is not the best, the clustering is still clearly present. We can actually also see that the clustering of Skinny64 is more pronounced than clustering for both Saturnin and Spongent.

The same about clustering can be concluded from Figure 34. The only difference between this figure and Figure 33 is that the histograms of all ciphers are converging instead of diverging.

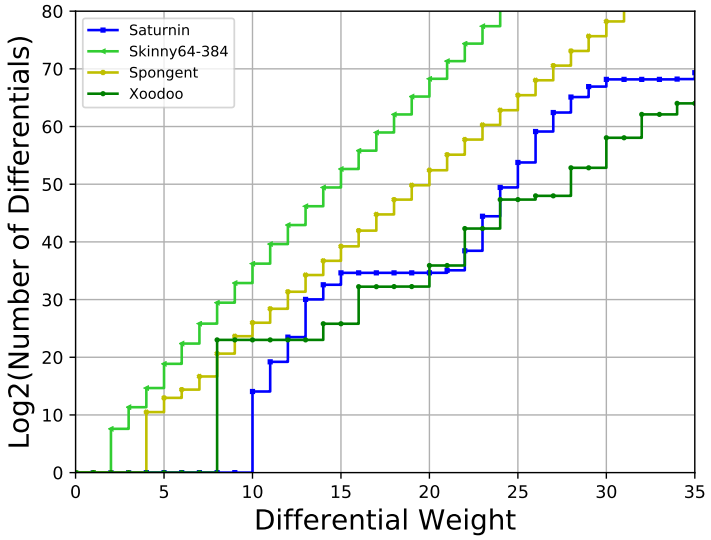


Figure 33: Two rounds: cumulative restriction weight histograms of Skinny64-384 and three out of the four ciphers investigated in [8].

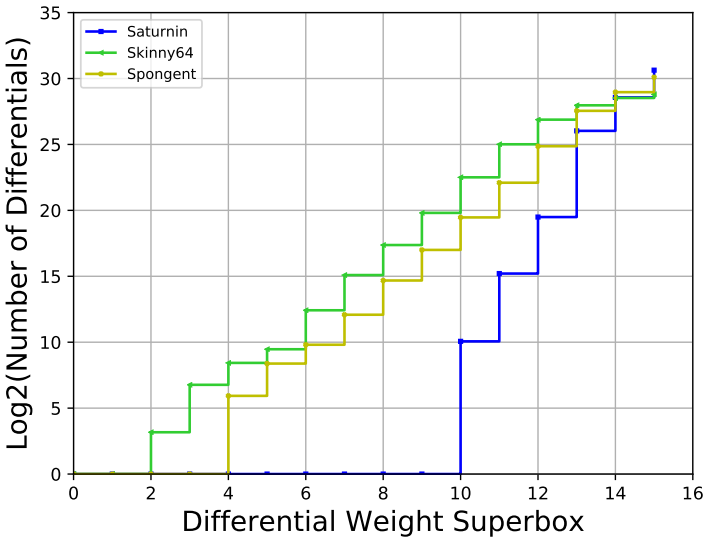


Figure 34: Two rounds: cumulative restriction weight histograms of the superbox of SKINNY64 and two out of the four ciphers investigated in [8].

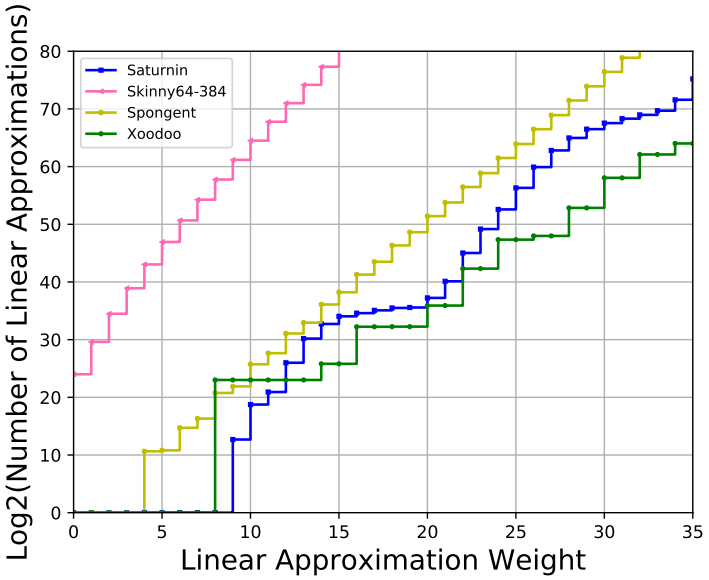


Figure 35: Two rounds: cumulative correlation weight histograms of SKINNY64-384 and three out of the four ciphers investigated in [8].

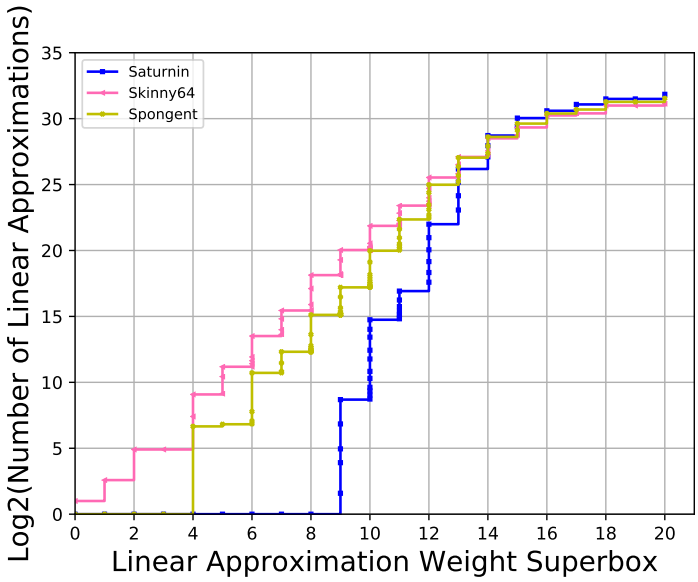


Figure 36: Two rounds: cumulative correlation weight histograms of the superbox of SKINNY64 and two out of the four ciphers investigated in [8].

If we follow the same reasoning, but then look at Figure 35 and Figure 30, we see something remarkable happening. The clustering greatly influences the graphs: instead of having SKINNY64-384 coincide with SPONGENT, SKINNY64-384 is way above the histogram of SPONGENT. We see that there are 2^{24} linear approximations with weight 0. As we can see in Figure 36, there are already 2^2 linear approximations with weight 0 in the superbox. This indicates a linear dependency in the superbox, which we discuss in more detail in subsection 5.2.1. Do note that the full state correlation weight histogram of SKINNY64-384 was obtained from that of its superboxes by first rounding the correlation weights and then convolving these histograms of its superboxes, just as done to SATURNIN and SPONGENT [8]. This rounding is actually also the reason why we see multiple markers on a vertical line in Figure 36.

5.2.1 Linear Dependencies

In the previous section, we stated that we found a linear dependency inside the superbox. Easily said, this means that there arises some linear relationship whenever we look at two-round structures (or more). So, if we just look at one application of a single S-box, nothing remarkable happens, but if we increase this to an application of two S-boxes, this will affect the linear propagation properties. If we now have some knowledge about the input bits, we will also have some knowledge about the output bits.

In [1] the following about two round structures regarding differentials and linear approximations was stated: "The above bounds are for single characteristic, thus it will be interesting to take a look at differentials and linear hulls. Being a rather complex task, we leave this as future work". In this sentence, linear hulls are what we call linear approximations. This suggests that the authors of [1] did not look into this linear dependency we found yet and we will thus elaborate on this below.

In Table 11 and Table 13 we will show the number of linear approximations for each absolute value of the correlation when the S-box of respectively SKINNY64 and SKINNY128 is applied once. The third column of this table shows the absolute value of the difference of the agreements A (the number of x for which $u^\top x = v^\top f(x)$) and disagreements D (the number of x for which $u^\top x \neq v^\top f(x)$). In Table 12 and Table 14 we show the same, but then for double application of the S-box of respectively SKINNY64 and SKINNY128.

Table 11: The absolute value of the correlation when applying an S-box of SKINNY64 once.

|Correlation|	Number of linear approximations	$\|A - D\|$
0.25	96	4
0.5	36	8
1.0	1	16

Table 12: The absolute value of the correlation when applying an S-box of SKINNY64 twice.

|Correlation|	Number of linear approximations	$\|A - D\|$
0.25	92	4
0.5	24	8
0.75	4	12
1.0	2	16

In Table 11 and Table 12, we see thus the reason for what is shown in Figure 35 and Figure 36. If we apply the S-box only once, we see just 1 linear approximation with $|\text{Correlation}| = 1.0$ and $|A - D| = 16$. If we apply the S-box twice, we see that this 1 changes into a 2: next to the trivial linear approximation, there exists some non-trivial linear approximation that yields an absolute correlation value of 1.0.

Even though we could not compute the cumulative correlation weight of SKINNY128 as done in Figure 35 and Figure 36 for SKINNY64, we could apply its S-boxes two times. As shown in Table 13 and Table 14, we see the same happening as just described for SKINNY64. This indicates that there also exists a linear dependency in the superbox of SKINNY128. We thus expect that the cumulative correlation weight of SKINNY128 would be located even higher in Figure 35 and Figure 36 since SKINNY128 also has bigger S-boxes than SKINNY64. However, because of insufficient resources, we cannot defend or refute this line of reasoning.

Table 13: The absolute value of the correlation when applying an S-box of SKINNY128 once.

$\lvert\text{Correlation}\rvert$	Number of linear approximations	$\lvert A - D\rvert$
0.03125	5810	8
0.0625	9904	16
0.09375	1846	24
0.125	5624	32
0.15625	388	40
0.1875	464	48
0.21875	118	56
0.25	892	64
0.28125	22	72
0.34375	4	88
0.375	24	96
0.40625	2	104
0.46875	2	120
0.5	52	128
1.0	1	256

Table 14: The absolute value of the correlation when applying an S-box of SKINNY128 twice.

|Correlation|	Number of linear approximations	$\|A - D\|$
0.015625	7852	4
0.03125	11550	8
0.046875	4982	12
0.0625	8208	16
0.078125	1904	20
0.09375	2564	24
0.109375	824	28
0.125	2530	32
0.140625	406	36
0.15625	478	40
0.171875	158	44
0.1875	474	48
0.203125	82	52
0.21875	180	56
0.234375	80	60
0.25	318	64
0.265625	40	68
0.28125	46	72
0.296875	20	76
0.3125	58	80
0.328125	18	84
0.34375	22	88
0.359375	8	92
0.375	22	96
0.390625	6	100
0.40625	4	104
0.421875	4	108
0.4375	8	112
0.46875	2	120
0.5	18	128
0.53125	2	136
0.5625	4	144
0.625	4	160
0.75	4	192
1.0	2	256

Chapter 6

Conclusions

In this thesis, we have investigated to what extent the design of the round function of block cipher family Skinny influences the differential and linear propagation properties. We first deduced that our primitive Skinny is aligned and after this showed that this alignment is a very important property that affects the propagation properties through the interaction of the linear and nonlinear layers. This effect turned out to have negative consequences for our investigated primitive, making it not as efficient as initially thought. We found that huddling has a quite strong negative effect on the differential and linear propagation properties of Skinny128, whereas this effect is slightly less pronounced for Skinny64. Next, we encountered that Skinny64 performed the worst on the cumulative differential trail weight and had the highest tail when looking at the cumulative linear trail weight. The last thing we found was that there is a lot of clustering in both instances of Skinny. For Skinny64 this influenced the differential propagation properties quite notably but greatly affected the linear propagation properties. We found a linear dependency inside the superbox of Skinny64 and the same kind of linear dependency was found inside the superbox of Skinny128.

In this work, we considered some parts to be out-of-scope and we also encountered programs that were infeasible to run with our available resources and/or computational power. Future research could be done in these areas, such as writing optimized assembly code for the ARM Cortex-M4 processor for Skinny or investigating, for instance, the cumulative restriction weight and correlation weight histograms of Skinny128. Another idea would be to add another (unaligned) lightweight cipher to this investigation and compare this cipher to our instances of Skinny. Last but not least, more research could be done regarding the linear dependencies found. It can be investigated which keys are considered weak due to this dependency and whether this has more side effects on the performance of this block cipher family.

Bibliography

[1] Christof Beierle, Jérémy Jean, Stefan Kölbl, Gregor Leander, Amir Moradi, Thomas Peyrin, Yu Sasaki, Pascal Sasdrich, and Siang Meng Sim. The SKINNY Family of Block Ciphers and Its Low-Latency Variant MANTIS. In Robshaw M. and Katz J., editors, *Advances in Cryptology - CRYPTO 2016*, pages 123–153. Springer Berlin Heidelberg, Heidelberg, Germany, 2016.

[2] Christof Beierle, Jérémy Jean, Stefan Kölbl, Gregor Leander, Amir Moradi, Thomas Peyrin, Yu Sasaki, Pascal Sasdrich, and Siang Meng Sim. SKINNY family of block ciphers. `https://sites.google.com/site/skinnycipher/home`, May 2021. Last accessed: May 26, 2021.

[3] Guido Bertoni, Joan Daemen, Seth Hoffert, Michaël Peeters, Gilles Van Assche, and Ronny Van Keer. XKCP (eXtended Keccak Code Package). Xoodoo and Keccak Code Package. `https://github.com/XKCP/XKCP.git`, January 2021. Last accessed: May 26, 2021.

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

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

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

[7] Andrey Bogdanov, Miroslav Knežević, Gregor Leander, Deniz Toz, Kerem Varici, and Ingrid Verbauwhede. SPONGENT: The Design Space of Lightweight Cryptographic Hashing. *IEEE Transactions on Computers*, 62(10):2041–2053, October 2013.

[8] Nicolas Bordes, Joan Daemen, Daniël Kuijsters, and Gilles Van Assche. Thinking Outside the Superbox. Cryptology ePrint Archive, Report 2021/293. https://eprint.iacr.org/2021/293, March 2021. Last accessed: May 26, 2021.

[9] Joan Boyar and René Peralta. A New Combinational Logic Minimization Technique with Applications to Cryptology. In Festa P., editor, *Experimental Algorithms*, pages 178–189. Springer Berlin Heidelberg, Heidelberg, Germany, 2010.

[10] Anne Canteaut, Sébastien Duval, Gaëtan Leurent, María Naya-Plasencia, Léo Perrin, Thomas Pornin, and André Schrottenloher. saturnin (Saturnin implementations). https://project.inria.fr/saturnin/files/2019/05/saturnin.zip, May 2019. Last accessed: May 26, 2021.

[11] Anne Canteaut, Sébastien Duval, Gaëtan Leurent, María Naya-Plasencia, Léo Perrin, Thomas Pornin, and André Schrottenloher. Saturnin: a suite of lightweight symmetric algorithms for post-quantum security. *IACR Transactions on Symmetric Cryptology*, 2020(S1):160–207, June 2020.

[12] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms*. The MIT Press, Cambridge, MA, USA, third edition, 2009.

[13] Sébastien Cottinet. Espresso algorithm source code. A modern (2017) compiliable re-host of the Espresso heuristic logic minimizer. https://github.com/classabbyamp/espresso-logic, April 2020. Last accessed: May 26, 2021.

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

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

[16] Joan Daemen and Vincent Rijmen. Plateau Characteristics. *IET Information Security*, 1(1):11–17, March 2007.

[17] Joan Daemen and Vincent Rijmen. New criteria for linear maps in AES-like ciphers. *Cryptography and Communications*, 1(1):47–69, April 2009.

[18] Ashutosh Dwivedi, Shalini Dhar, Gautam Srivastava, and Rajani Singh. Cryptanalysis of Round-Reduced Fantomas, Robin and iSCREAM. *Cryptography*, 3(1):4, January 2019.

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

[20] D. J. H. Garling. *Foundations and Elementary Real Analysis*, volume 1 of *A Course in Mathematical Analysis*. Cambridge University Press, Cambridge, UK, 2013.

[21] Lars R. Knudsen. Truncated and higher order differentials. In Preneel B., editor, *Fast Software Encryption*, pages 196–211. Springer Berlin Heidelberg, Heidelberg, Germany, 1995.

[22] Thorsten Kranz, Gregor Leander, Ko Stoffelen, and Friedrich Wiemer. Shorter Linear Straight-Line Programs for MDS Matrices - Yet another XOR Count Paper. *IACR Transactions on Symmetric Cryptology*, 2017(4):188–211, December 2017.

[23] Guozhen Liu, Mohona Ghosh, and Ling Song. Security Analysis of SKINNY under Related-Tweakey Settings. *IACR Transactions on Symmetric Cryptology*, 2017(3):37–72, September 2017.

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

[25] Patrick McGeer, Jagesh Sanghavi, Robert Brayton, and Alberto Sangiovanni Vincentelli. ESPRESSO-SIGNATURE: A New Exact Minimizer for Logic Functions. In *Proceedings of the 30th International Design Automation Conference*, DAC '93, page 618–624, New York, NY, USA, 1993. Association for Computing Machinery.

[26] National Institute of Standards and Technology. Lightweight Cryptography. `https://csrc.nist.gov/Projects/lightweight-cryptography`, January 2017. Last accessed: May 26, 2021.

[27] The University of Texas Department of Electrical and Computer Engineering. EE 382N - Espresso Manual. `http://users.ece.utexas.edu/~patt/06s.382N/tutorial/espresso_manual.html`, March 2006. Last accessed: May 26, 2021.

[28] Sangwoo Park, Soo Hak Sung, Seongtaek Chee, E-Joong Yoon, and Jongin Lim. On the Security of Rijndael-Like Structures against Differential and Linear Cryptanalysis. In Zheng Y., editor, *Lecture Notes in Computer Science*, pages 176–191. Springer Berlin Heidelberg, Heidelberg, Germany, 2002.

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

[30] Markku-Juhani O. Saarinen. Cryptographic Analysis of All 4 × 4-Bit S-Boxes. In Miri A. and Vaudenay S., editors, *Selected Areas in Cryptography*, pages 118–133. Springer Berlin Heidelberg, Heidelberg, Germany, 2012.

[31] Sumanta Sarkar and Habeeb Syed. Bounds on Differential and Linear Branch Number of Permutations. In Susilo W. and Yang G, editors, *Information Security and Privacy*, pages 207–224. Springer International Publishing, Manhattan, NY, USA, 2018.

[32] Peter Schwabe and Ko Stoffelen. All the AES You Need on Cortex-M3 and M4. In Avanzi R. and Heys H., editors, *Lecture Notes in Computer Science*, pages 180–194. Springer International Publishing, Manhattan, NY, USA, 2017.

[33] Claude Elwood Shannon. A Mathematical Theory of Communication. *Bell System Technical Journal*, 27(3):379–423, July 1948.

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

[35] Ko Stoffelen. aes-armcortexm (AES implementations). Fast AES on ARM Cortex-M3 and M4. `https://github.com/Ko-/aes-armcortexm.git`, July 2020. Last accessed: May 26, 2021.

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

[37] Henk C.A. van Tilborg. *FUNDAMENTALS OF CRYPTOLOGY - A Professional Reference and Interactive Tutorial*. Kluwer Academic Publishers, Dordrecht, Netherlands, 2000.

Appendix A

Espresso Algorithm: Minimal Sum-Of-Product (SOP) Form

The Espresso algorithm is used to obtain the minimal sum-of-product form of both SKINNY64 and SKINNY128. Below we represent OR by addition, AND by multiplication and a negation by an overline. Besides this, the ith input is specified by X_i and the jth output is specified by Y_j.

A.1 SOP Expressions

The minimal sum-of-product form of SKINNY64 is as follows:

$$
\begin{aligned}
Y_0 &= \overline{X_0X_1X_2X_3} + \overline{X_0X_1}X_2\overline{X_3} + X_0X_3 + X_1X_3\\
Y_1 &= \overline{X_0X_1X_2X_3} + \overline{X_0X_1X_2}X_3 + X_0X_1 + X_0X_2\\
Y_2 &= \overline{X_0X_1X_2}X_3 + X_0\overline{X_1X_2X_3} + X_1X_2 + X_1X_3\\
Y_3 &= \overline{X_0}X_1\overline{X_2X_3} + \overline{X_0X_1}X_2\overline{X_3} + X_0\overline{X_1X_2X_3} + X_1X_2X_3 + X_0X_2
\end{aligned}
$$

For SKINNY128, the SOP expression looks like:

$$
\begin{aligned}
Y_0 = {} & \overline{X_0X_1X_2}X_3\overline{X_4X_5}X_7 + \overline{X_0X_1X_2}X_3X_5\overline{X_7} + X_1\overline{X_2X_3X_4X_5}X_7 +\\
& \overline{X_0X_1X_2}X_3X_4\overline{X_7} + X_0\overline{X_2X_3X_4X_5}X_7 + X_1\overline{X_2X_3}X_5\overline{X_7} +\\
& X_1\overline{X_2X_3}X_4\overline{X_7} + X_0\overline{X_2X_3}X_4\overline{X_7} + X_0\overline{X_2X_3}X_5\overline{X_7} + \overline{X_0X_1}X_2\overline{X_3} +\\
& X_2\overline{X_4X_5X_7} + X_1X_2X_3 + X_2X_5X_7 + X_0X_2X_3 + X_2X_4X_7\\
Y_1 = {} & \overline{X_0X_1X_3} + X_0X_3 + X_1X_3\\
Y_2 = {} & \overline{X_4X_5X_7} + X_5X_7 + X_4X_7\\
Y_3 = {} & \overline{X_0X_1}X_2X_3\overline{X_4}X_5\overline{X_7} + \overline{X_0X_1X_2}X_3\overline{X_4}X_5X_7 + X_0X_2\overline{X_3X_4}X_5\overline{X_7} +\\
& X_0\overline{X_2X_3X_4}X_5X_7 + X_1X_2\overline{X_3X_4}X_5\overline{X_7} + X_1\overline{X_2X_3X_4}X_5X_7 +\\
& \overline{X_0X_1}X_2X_3\overline{X_5}X_7 + \overline{X_0X_1X_2}X_3\overline{X_5X_7} + \overline{X_0X_1X_2}X_3X_4\overline{X_7} +
\end{aligned}
$$

$$
\begin{aligned}
&X_0X_2\overline{X_3X_5}X_7 + X_1X_2\overline{X_3X_5}X_7 + X_0\overline{X_2X_3X_5X_7} + X_1\overline{X_2X_3X_5X_7} + \\
&X_1\overline{X_2X_3}X_4\overline{X_7} + X_0\overline{X_2X_3}X_4\overline{X_7} + \overline{X_0X_1X_3}X_4 + X_0X_3X_4 + X_1X_3X_4 + \\
&X_2X_4X_7 \\
Y_4 = {} & \overline{X_4X_5X_6}X_7 + \overline{X_4}X_5\overline{X_6X_7} + \overline{X_5}X_6\overline{X_7} + X_5X_6X_7 + X_4X_6 \\
Y_5 = {} & \overline{X_1X_5X_6} + X_1X_6 + X_1X_5 \\
Y_6 = {} & \overline{X_1}X_2X_3\overline{X_4X_5}X_6X_7 + \overline{X_0X_1X_2X_4X_5}X_6\overline{X_7} + \overline{X_0}X_1X_2\overline{X_3}X_4\overline{X_5X_6X_7} + \\
&\overline{X_0}X_1X_2\overline{X_3X_4X_5X_6}X_7 + \overline{X_0}X_1\overline{X_2}X_4\overline{X_5X_6}X_7 + \overline{X_0}X_1\overline{X_2}X_4X_5X_6X_7 + \\
&\overline{X_1}X_2X_3X_4X_6\overline{X_7} + \overline{X_0X_1X_2}X_4X_6X_7 + \overline{X_0}X_1\overline{X_2}X_3\overline{X_5X_6} + \\
&X_0\overline{X_2X_3X_4X_5}X_7 + \overline{X_0X_1X_2X_3}X_6 + X_0X_2\overline{X_4X_5X_7} + \overline{X_1}X_2X_3X_5\overline{X_7} + \\
&\overline{X_0X_1X_2X_3}X_5 + \overline{X_0X_1X_2}X_5X_7 + X_0\overline{X_2X_3}X_4\overline{X_7} + X_0\overline{X_2X_3}X_5\overline{X_7} + \\
&X_0\overline{X_1X_5X_6} + X_0X_2X_4X_7 + X_0X_2X_5X_7 + X_0X_1X_6 + X_0X_2X_3 + \\
&X_0X_1X_5 \\
Y_7 = {} & X_0X_1X_2\overline{X_3}X_4\overline{X_6X_7} + \overline{X_0X_2X_3}X_4\overline{X_5X_6X_7} + \overline{X_1}X_2\overline{X_3X_4}X_6X_7 + \\
&\overline{X_1X_2}X_3\overline{X_4}X_6X_7 + X_0X_1\overline{X_2}X_4\overline{X_6}X_7 + X_0X_1\overline{X_2}X_3\overline{X_6X_7} + \\
&X_0X_1\overline{X_2X_4X_6X_7} + \overline{X_0}X_2X_4\overline{X_5X_6}X_7 + \overline{X_0}X_2X_3\overline{X_5X_6X_7} + \\
&\overline{X_0}X_1\overline{X_4}X_6X_7 + \overline{X_0X_1}X_4\overline{X_5X_6} + \overline{X_0}X_2\overline{X_4X_6X_7} + \overline{X_0X_1X_5X_6X_7} + \\
&\overline{X_1}X_2X_3X_5\overline{X_7} + \overline{X_0X_1X_2X_3}X_5 + \overline{X_0X_1X_2}X_5X_7 + X_0\overline{X_2X_3}X_5\overline{X_7} + \\
&\overline{X_4}X_5\overline{X_6X_7} + X_0X_2X_5X_7 + X_4X_5X_6 + X_5X_6X_7 + X_0X_1X_5
\end{aligned}
$$

We know that De Morgan's laws state that $X_iX_j + X_kX_l$ can be reformulated as $\overline{\overline{X_iX_j}\ \overline{X_kX_l}}$. For us this means that "the circuits of depth two that can be derived from the SOPs above, consisting of a layer of (possibly multi-input) AND gates, followed by a layer of (possibly multi-input) OR gates, can be converted into a circuit of depth two in which each layer consists of (possibly multi-input) NAND gates" [8].

In the next section, we present the S-boxes of SKINNY64 and SKINNY128 which were used in the Espresso algorithm to derive these SOPs. We converted this hexadecimal representation to a binary representation to end up with the correct input format for the Espresso algorithm.

A.2 S-boxes of Skinny64 and Skinny128

The S-box of Skinny64 looks like

```
const uint8_t sbox[] = {
  0xc, 0x6, 0x9, 0x0, 0x1, 0xa, 0x2, 0xb, 0x3, 0x8,
  0x5, 0xd, 0x4, 0xe, 0x7, 0xf
};
```

and the S-box of Skinny128 can be written down as

```
const uint8_t sbox[] = {
  0x65, 0x4c, 0x6a, 0x42, 0x4b, 0x63, 0x43, 0x6b, 0x55, 0x75,
  0x5a, 0x7a, 0x53, 0x73, 0x5b, 0x7b, 0x35, 0x8c, 0x3a, 0x81,
  0x89, 0x33, 0x80, 0x3b, 0x95, 0x25, 0x98, 0x2a, 0x90, 0x23,
  0x99, 0x2b, 0xe5, 0xcc, 0xe8, 0xc1, 0xc9, 0xe0, 0xc0, 0xe9,
  0xd5, 0xf5, 0xd8, 0xf8, 0xd0, 0xf0, 0xd9, 0xf9, 0xa5, 0x1c,
  0xa8, 0x12, 0x1b, 0xa0, 0x13, 0xa9, 0x05, 0xb5, 0x0a, 0xb8,
  0x03, 0xb0, 0x0b, 0xb9, 0x32, 0x88, 0x3c, 0x85, 0x8d, 0x34,
  0x84, 0x3d, 0x91, 0x22, 0x9c, 0x2c, 0x94, 0x24, 0x9d, 0x2d,
  0x62, 0x4a, 0x6c, 0x45, 0x4d, 0x64, 0x44, 0x6d, 0x52, 0x72,
  0x5c, 0x7c, 0x54, 0x74, 0x5d, 0x7d, 0xa1, 0x1a, 0xac, 0x15,
  0x1d, 0xa4, 0x14, 0xad, 0x02, 0xb1, 0x0c, 0xbc, 0x04, 0xb4,
  0x0d, 0xbd, 0xe1, 0xc8, 0xec, 0xc5, 0xcd, 0xe4, 0xc4, 0xed,
  0xd1, 0xf1, 0xdc, 0xfc, 0xd4, 0xf4, 0xdd, 0xfd, 0x36, 0x8e,
  0x38, 0x82, 0x8b, 0x30, 0x83, 0x39, 0x96, 0x26, 0x9a, 0x28,
  0x93, 0x20, 0x9b, 0x29, 0x66, 0x4e, 0x68, 0x41, 0x49, 0x60,
  0x40, 0x69, 0x56, 0x76, 0x58, 0x78, 0x50, 0x70, 0x59, 0x79,
  0xa6, 0x1e, 0xaa, 0x11, 0x19, 0xa3, 0x10, 0xab, 0x06, 0xb6,
  0x08, 0xba, 0x00, 0xb3, 0x09, 0xbb, 0xe6, 0xce, 0xea, 0xc2,
  0xcb, 0xe3, 0xc3, 0xeb, 0xd6, 0xf6, 0xda, 0xfa, 0xd3, 0xf3,
  0xdb, 0xfb, 0x31, 0x8a, 0x3e, 0x86, 0x8f, 0x37, 0x87, 0x3f,
  0x92, 0x21, 0x9e, 0x2e, 0x97, 0x27, 0x9f, 0x2f, 0x61, 0x48,
  0x6e, 0x46, 0x4f, 0x67, 0x47, 0x6f, 0x51, 0x71, 0x5e, 0x7e,
  0x57, 0x77, 0x5f, 0x7f, 0xa2, 0x18, 0xae, 0x16, 0x1f, 0xa7,
  0x17, 0xaf, 0x01, 0xb2, 0x0e, 0xbe, 0x07, 0xb7, 0x0f, 0xbf,
  0xe2, 0xca, 0xee, 0xc6, 0xcf, 0xe7, 0xc7, 0xef, 0xd2, 0xf2,
  0xde, 0xfe, 0xd7, 0xf7, 0xdf, 0xff
};
```

Appendix B

Two-dimensional Histograms

This appendix provides tables, top and side views of all two-dimensional histograms provided in this thesis for the sake of clarity.

B.1 Distribution of Ordered Bit Weight Pairs of Skinny64

Linear Bit Weight Pairs

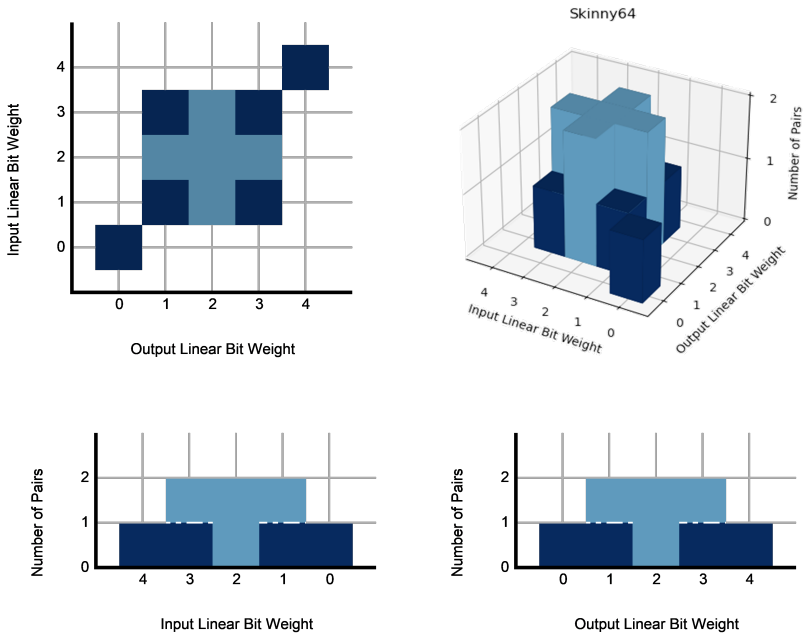


Figure 37: Two-dimensional histogram of the ordered linear $(w_{2,in}, w_{2,out})$-pairs of Skinny64 as depicted in Figure 18(b) along with the top and side views.

Differential Bit Weight Pairs

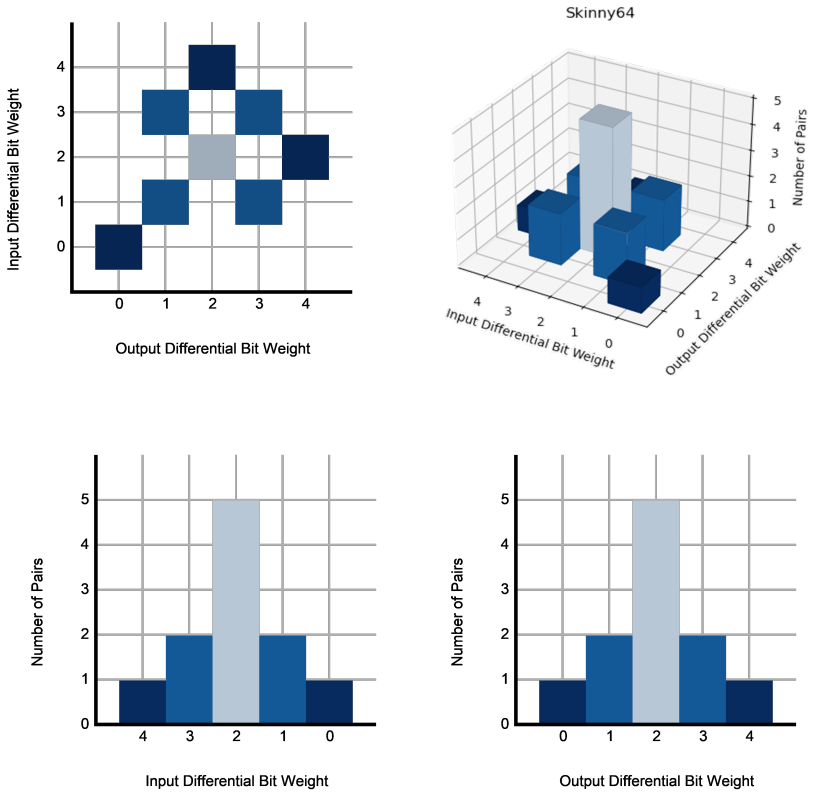


Figure 38: Two-dimensional histogram of the ordered differential $(w_{2,in}, w_{2,out})$-pairs of SKINNY64 as depicted in Figure 18(a) along with the top and side views.

Tables of Distribution of Ordered Bit Weight Pairs of SKINNY64

Table 15: The data used for the two-dimensional histograms in Figure 18(a) and Figure 18(b) for SKINNY64. On the left side we see the data used in Figure 37 and on the right side we see the data used in Figure 38.

$(w_{2,in}, w_{2,out})$	Number of Pairs
$(0,0)$	1
$(1,1)$	1
$(1,2)$	2
$(1,3)$	1
$(2,1)$	2
$(2,2)$	2
$(2,3)$	2
$(3,1)$	1
$(3,2)$	2
$(3,3)$	1
$(4,4)$	1

$(w_{2,in}, w_{2,out})$	Number of Pairs
$(0,0)$	1
$(1,1)$	2
$(1,3)$	2
$(2,2)$	5
$(2,4)$	1
$(3,1)$	2
$(3,3)$	2
$(4,2)$	1

B.2 Distribution of Ordered Box Weight Pairs of Skinny64

Linear Box Weight Pairs

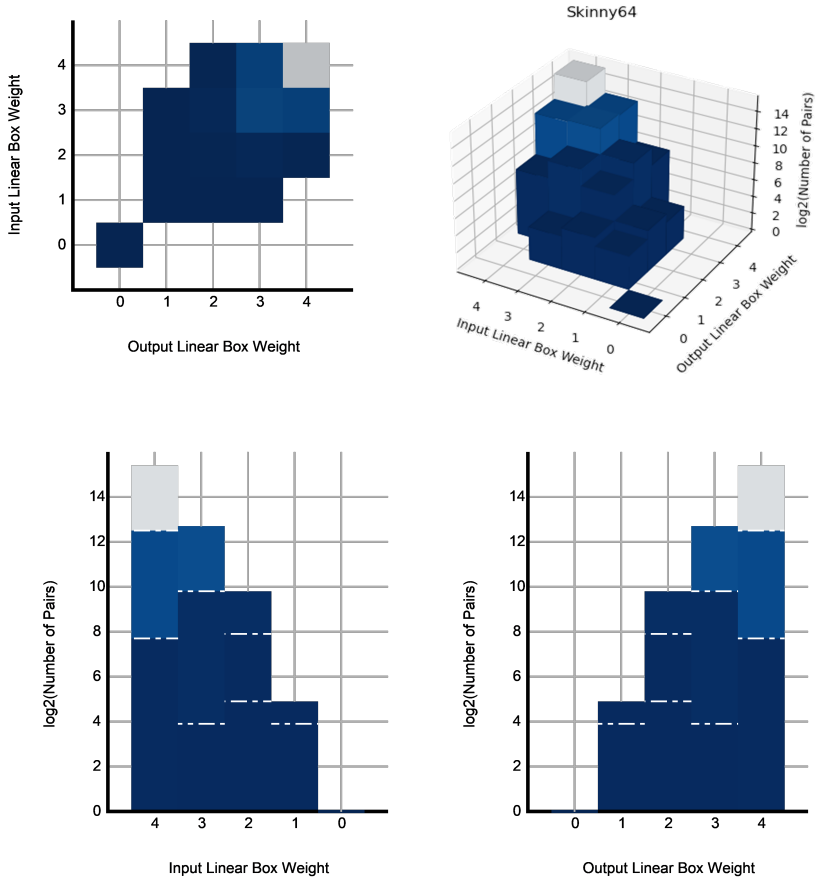


Figure 39: Two-dimensional histogram of the ordered linear $(\mathrm{w}_{\Pi,\mathrm{in}}, \mathrm{w}_{\Pi,\mathrm{out}})$-pairs of Skinny64 as depicted in Figure 24(b) along with the top and side views.

Differential Box Weight Pairs

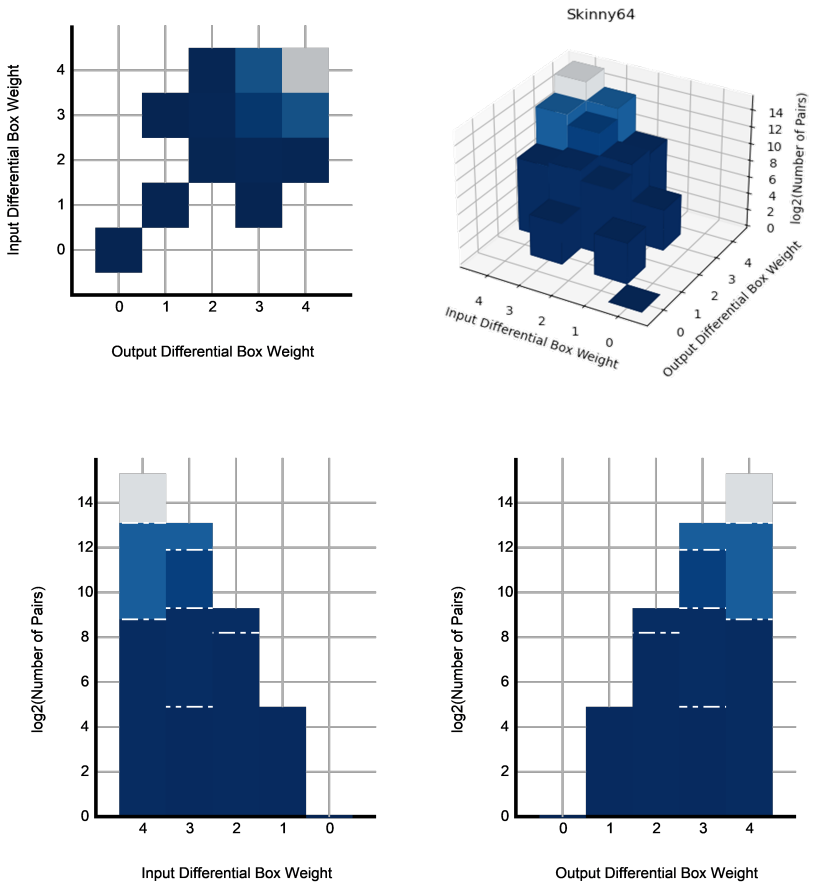


Figure 40: Two-dimensional histogram of the ordered differential $(\mathrm{w}_{\Pi,\mathrm{in}}, \mathrm{w}_{\Pi,\mathrm{out}})$-pairs of SKINNY64 as depicted in Figure 24(a) along with the top and side views.

Tables of Distribution of Ordered Box Weight Pairs of Skinny64

Table 16: The data used for the two-dimensional histograms in Figure 24(a) and Figure 24(b) for Skinny64. On the left side we see the data used in Figure 39 and on the right side we see the data used in Figure 40.

$(w_{\Pi,\text{in}}, w_{\Pi,\text{out}})$	Number of Pairs
$(0,0)$	1
$(1,1)$	15
$(1,2)$	30
$(1,3)$	15
$(2,1)$	30
$(2,2)$	240
$(2,3)$	870
$(2,4)$	210
$(3,1)$	15
$(3,2)$	870
$(3,3)$	6735
$(3,4)$	5880
$(4,2)$	210
$(4,3)$	5880
$(4,4)$	44535

$(w_{\Pi,\text{in}}, w_{\Pi,\text{out}})$	Number of Pairs
$(0,0)$	1
$(1,1)$	30
$(1,3)$	30
$(2,2)$	285
$(2,3)$	630
$(2,4)$	435
$(3,1)$	30
$(3,2)$	630
$(3,3)$	3810
$(3,4)$	9030
$(4,2)$	435
$(4,3)$	9030
$(4,4)$	41160

B.3 Distribution of Ordered Bit Weight Pairs of Skinny128

Linear Bit Weight Pairs

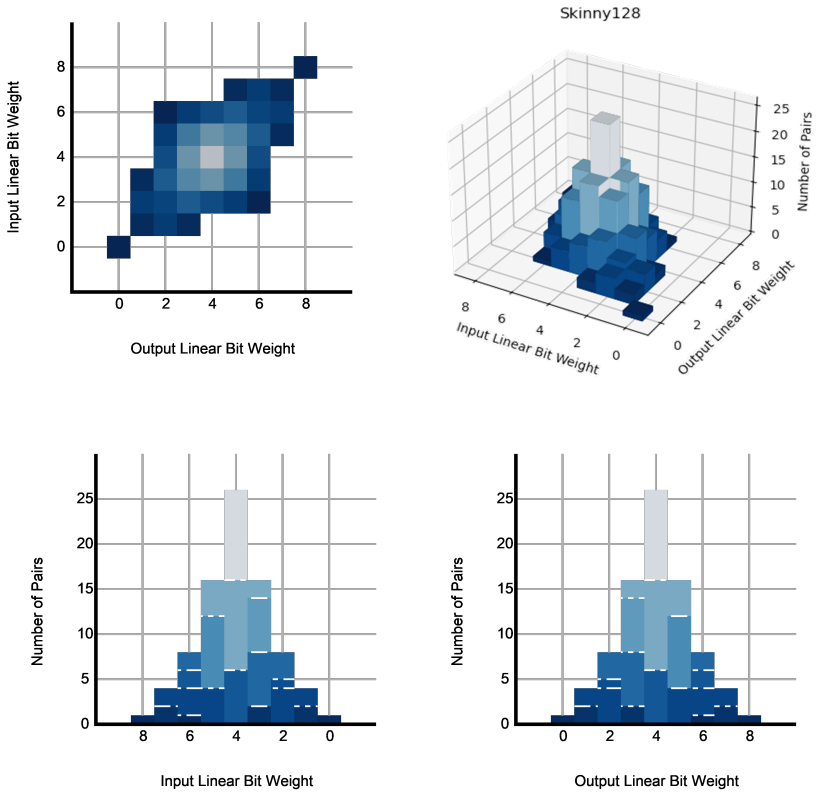


Figure 41: Two-dimensional histogram of the ordered linear $(w_{2,in}, w_{2,out})$-pairs of Skinny128 as depicted in Figure 19(b) along with the top and side views.

Differential Bit Weight Pairs

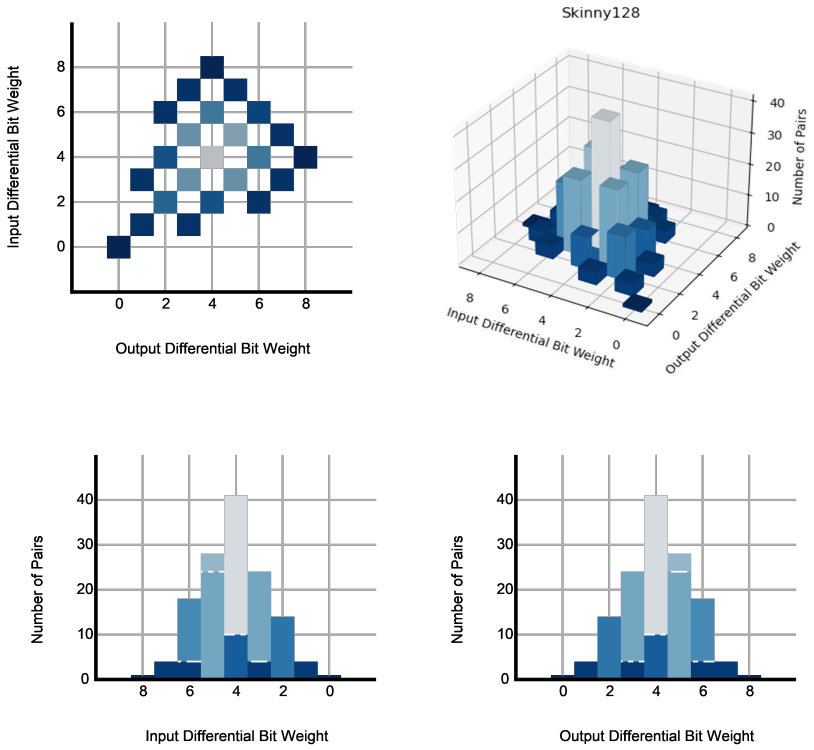


Figure 42: Two-dimensional histogram of the ordered differential $(w_{2,in}, w_{2,out})$-pairs of SKINNY128 as depicted in Figure 19(a) along with the top and side views.

Tables of Distribution of Ordered Bit Weight Pairs of SKINNY128

Table 17: The data used for the two-dimensional histograms in Figure 19(a) and Figure 19(b) for SKINNY128. On the left side we see the data used in Figure 41 and on the right side we see the data used in Figure 42.

$(w_{2,in}, w_{2,out})$	Number of Pairs
$(0,0)$	1
$(1,1)$	2
$(1,2)$	4
$(1,3)$	2
$(2,1)$	4
$(2,2)$	5
$(2,3)$	8
$(2,4)$	6
$(2,5)$	4
$(2,6)$	1
$(3,1)$	2
$(3,2)$	8
$(3,3)$	14
$(3,4)$	16
$(3,5)$	12
$(3,6)$	4
$(4,2)$	6
$(4,3)$	16
$(4,4)$	26
$(4,5)$	16
$(4,6)$	6
$(5,2)$	4
$(5,3)$	12
$(5,4)$	16
$(5,5)$	14
$(5,6)$	8
$(5,7)$	2
$(6,2)$	1
$(6,3)$	4
$(6,4)$	6
$(6,5)$	8
$(6,6)$	5
$(6,7)$	4
$(7,5)$	2
$(7,6)$	4
$(7,7)$	2
$(8,8)$	1

$(w_{2,in}, w_{2,out})$	Number of Pairs
$(0,0)$	1
$(1,1)$	4
$(1,3)$	4
$(2,2)$	14
$(2,4)$	10
$(2,6)$	4
$(3,1)$	4
$(3,3)$	24
$(3,5)$	24
$(3,7)$	4
$(4,2)$	10
$(4,4)$	41
$(4,6)$	18
$(4,8)$	1
$(5,3)$	24
$(5,5)$	28
$(5,7)$	4
$(6,2)$	4
$(6,4)$	18
$(6,6)$	6
$(7,3)$	4
$(7,5)$	4
$(8,4)$	1

B.4 Distribution of Ordered Box Weight Pairs of Skinny128

Linear Box Weight Pairs

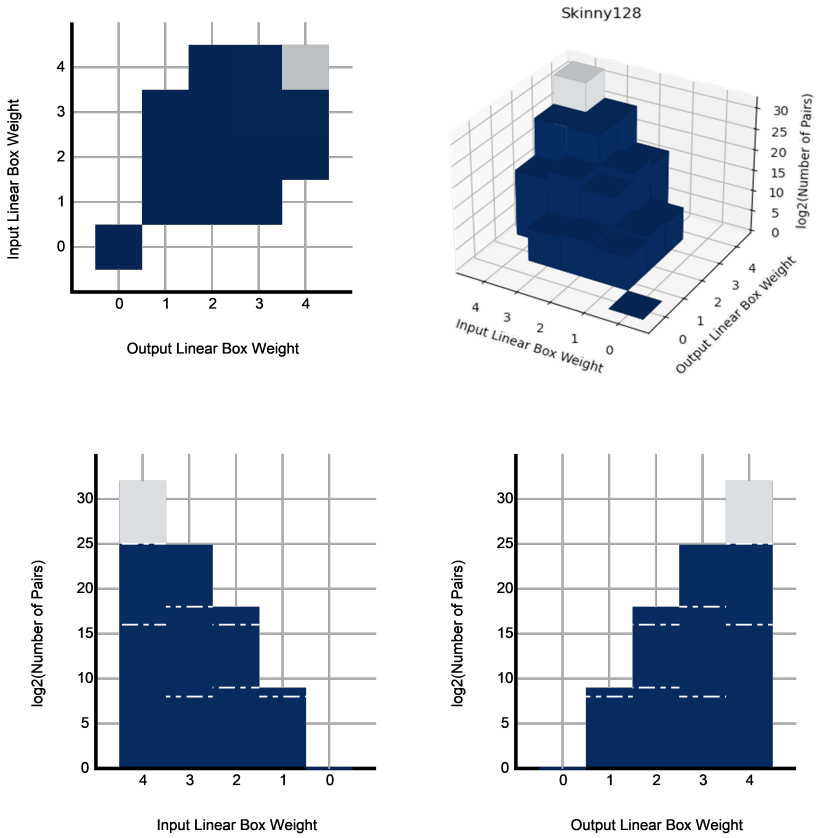


Figure 43: Two-dimensional histogram of the ordered linear $(\mathrm{w}_{\Pi,\mathrm{in}}, \mathrm{w}_{\Pi,\mathrm{out}})$-pairs of Skinny128 as depicted in Figure 25(b) along with the top and side views.

Differential Box Weight Pairs

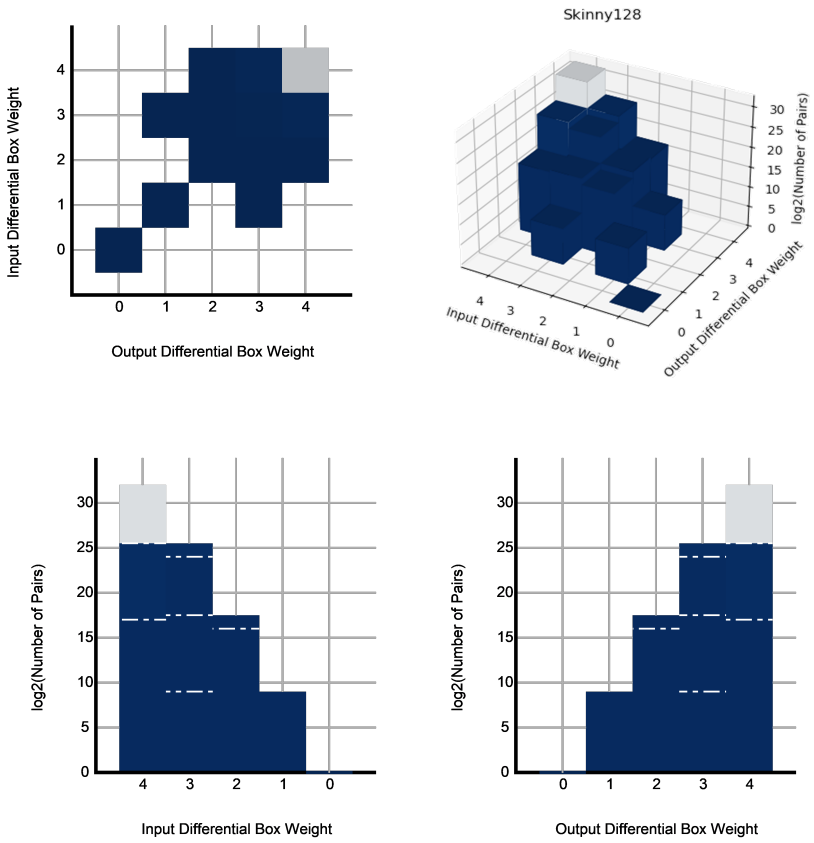


Figure 44: Two-dimensional histogram of the ordered differential $(\mathrm{w}_{\Pi,\mathrm{in}}, \mathrm{w}_{\Pi,\mathrm{out}})$-pairs of Skinny128 as depicted in Figure 25(a) along with the top and side views.

Tables of Distribution of Ordered Box Weight Pairs of SKINNY128

Table 18: The data used for the two-dimensional histograms in Figure 25(a) and Figure 25(b) for SKINNY128. On the left side we see the data used in Figure 43 and on the right side we see the data used in Figure 44.

$(w_{\Pi,\text{in}}, w_{\Pi,\text{out}})$	Number of Pairs
$(0,0)$	1
$(1,1)$	255
$(1,2)$	510
$(1,3)$	255
$(2,1)$	510
$(2,2)$	65280
$(2,3)$	259590
$(2,4)$	64770
$(3,1)$	255
$(3,2)$	259590
$(3,3)$	33162495
$(3,4)$	32903160
$(4,2)$	64770
$(4,3)$	32903160
$(4,4)$	4195282695

$(w_{\Pi,\text{in}}, w_{\Pi,\text{out}})$	Number of Pairs
$(0,0)$	1
$(1,1)$	510
$(1,3)$	510
$(2,2)$	66045
$(2,3)$	194310
$(2,4)$	129795
$(3,1)$	510
$(3,2)$	194310
$(3,3)$	16711170
$(3,4)$	49419510
$(4,2)$	129795
$(4,3)$	49419510
$(4,4)$	4178701320