Reducing Multiplicative Complexity via Conjugate Cipher

Application to the Skinny Family of Block Ciphers

Noémie Akpaki[1,2] and Nicolas David[1]

[1] Thales SIX, Gennevilliers, France, nicolas-i.david@thalesgroup.com
noemie.akpaki@thalesgroup.com
[2] Université Paris VIII, Paris, France,

Abstract. Multiplicative complexity have shown to be an important metric for efficient implementations in various contexts such as side-channel secure implementation and transciphering.
We introduce a generic framework based on conjugacy to reduce the multiplicative complexity of block ciphers. Our approach exploits the iterative structure of the block cipher to build alternative implementation based on conjugate round operations with overall smaller multiplicative complexity.
We apply this methodology to the Skinny family of block ciphers and derive optimized implementations for all versions. By carefully analyzing the interaction between the non-linear and linear layers, we construct an alternative implementation that reduces the number of multiplications. For Skinny64, we obtain a gain of 5.5 % , while for Skinny128, the gain reaches 3%.

Keywords: Block Cipher · Implementation · S-box · Linear Layer · Conjugacy · Multiplicative Complexity · SKINNY

1 Introduction

The quality of a block cipher can be assessed through various metrics, such as security and implementation efficiency. Indeed, from older designs such as DES [S+99] to more modern standards such as Ascon [DEMS21], implementation efficiency has been a major concern for both designers and users. If These two requirements are often considered distinct constraints, this work provides some insight into how the two concepts can be bridged. Indeed, while cryptanalysis sometimes leverages algebraic properties to highlight unexpected behavior, the same type of property can also be leveraged to improve the implementation of a cipher.

In many applications, the implementation efficiency is largely driven by the number of AND gates involved. This is particularly true for side-channel resistant implementations. Masking is the usual countermeasure [GP99, CJRR99]; it splits sensitive values into multiple randomized shares to prevent relevant information leakage. While protecting linear operations with Boolean masking induces a linear cost in the number of shares, masking non-linear operations such as AND gates leads to a quadratic cost [ISW03]. This makes multiplicative complexity a central metric for efficient and secure implementations. Reducing the number of multiplications without weakening security is therefore a key challenge in modern cryptographic engineering. Another use case for lowering the number of AND gates is the design of STAPs (Symmetric Techniques for Advanced Protocols) [MAPB23]. One such protocol where having symmetric ciphers with low multiplicative complexity is relevant is fully homomorphic encryption (FHE). To solve the issue of

ciphertext expansion that all FHE schemes have, one may use transciphering [NLV11] as a viable solution. Rather than homomorphically encrypting a message, it is encrypted through a symmetric cipher, with only the key being encrypted through the homomorphic scheme. The recipient of the message may then homomorphically perform the decryption circuit, yielding an homomorphic encryption of the original message. With modern FHE schemes relying on noisy ciphertexts whose noise grow as computations are performed on it (and, most notably, as multiplications are performed), it is then relevant that the symmetric cipher used minimizes the amount of operations (specifically multiplications) it requires. While our proposal may not outperform cipher designs specifically intended for this use case, we argue that finding good candidates among existing and well-studied ciphers remains of value, whether for security or compatibility reasons. Our work establishes the basis for studying existing block ciphers under the metric of low multiplicative complexity.

When working with block ciphers, these non-linear operations are traditionally aggregated within non-linear components: S-boxes. It comes as no surprise that the community's first efforts have been directed towards S-box optimization.

Optimizing S-box Multiplicative Complexity. Optimizing the multiplicative complexity of S-boxes is a well-known difficult problem. In general, finding an optimal implementation of a Boolean function with respect to multiplicative complexity is NP-hard [BMP13]. Nevertheless, for small S-boxes (typically 4 or 5 bits), dedicated methods can still find optimal or near-optimal solutions.

SAT-based methods have been particularly successful since their introduction in [CMH13]. Stoffelen [Sto16] proposed a framework capable of finding optimal implementations for 4-bit S-boxes and some 5-bit instances. This line of work was extended in [BDD+20], which covered all 5-bit S-boxes and some 6-bit quadratic S-boxes, while also considering the multiplicative depth. More recently, Zhang and Huang [ZH23] improved the efficiency of SAT-based approaches and applied them to structured 8-bit S-boxes.

However, these approaches suffer from scalability limitations. To overcome this issue, heuristic methods have been proposed. The authors of [JPST17] introduces a tool relying on a meet-in-the-middle strategy to efficiently find low-complexity implementations of larger S-Boxes than SAT-based methods. A recent contribution [BDM25] proposes a dedicated tool for quadratic Boolean functions that exploits precomputed pattern matching, allowing efficient exploration up to 9-bit S-Boxes. Beyond bit-level optimizations, another direction is to consider the number of multiplications in a field extension : for instance, [GRVV17] generalized a decomposition-based approach, introduced and developed in [CGP+12, CRV14, GR16] also useful for S-Boxes of arbitrary size. They explore a trade-off between the size of the extension field and the number of multiplications in that field. These methods improve performance but do not guarantee optimality.

The approach developed in this work is orthogonal to the ones mentionned above since it leverages the global structure of the block cipher rather than solely optimizing the confusion layer. This global approach is inspired by a cryptanalytic technique: *Conjugate ciphers*.

Cryptanalysis of Conjugate Ciphers. This technique, originally inspired by [BCL18] and first formalized in [BBF+25], involves rearranging the data to consider a conjugate version of the cipher to reveal potential vulnerabilities. Consider a composition function G and an alternating block cipher $E : k \mapsto E_k$ consisting of r round functions:

$$E_k = F_k^{(r-1)} \circ \cdots \circ F_k^{(0)},$$

The conjugate cipher E_k^G is defined as:

$$E_k^G = G \circ E_k \circ G^{-1} = (F_k^{(r-1)})^G \circ \cdots \circ (F_k^{(0)})^G,$$

where each conjugated round function is obtained by applying G and its inverse around the original round function. Cryptanalysis via conjugacy identifies particular choices of G for which the conjugated round functions exhibit security weaknesses.

Our work once again leverages conjugacy techniques but pivots from a cryptanalytic perspective to an implementation-oriented one. Specifically, we aim to discover G functions that transform the round functions and thus the resulting conjugate cipher in a way that reduces the number of multiplications required for their implementation. This approach raises several key questions:

- How should multiplicative complexity be precisely defined in this context?
- In what ways does conjugacy affect the multiplication count necessary within a round function?
- How can we identify suitable G functions that achieve our optimization goals in both theoretical and practical scenarios?

Contributions. This work provides answers to all the above questions. In particular, we provide a framework to measure multiplicative complexity and study the effect of conjugacy on this measure. In some cases, for a well-chosen *non-linear* G, it is possible to obtain a conjugate with smaller multiplicative complexity. Following this observation, we describe a framework for the conjugacy of the whole cipher, through the study of conjugate round operations. Finally, we provide an application of this methodology to the Skinny family of block ciphers [BJK+16], highlighting implementations with reduced multiplicative complexity for all versions as presented in Table 1.

Organization. This work is organized as follows: First, Section 2 lays out notation and establishes definitions and properties around multiplicative complexity and conjugacy in the general context of Boolean and vectorial Boolean functions. Then, Section 3 presents the new framework for reducing multiplicative complexity in the context of block ciphers. Finally, Section 4 presents the application of the technique to the Skinny family of block ciphers [BJK+16].

Table 1: Multiplicative Complexity of all Skinny Versions.

Skinny Version	♯ Round	Standard (Nb. Mult.)	Conjugate (Nb. Mult.)	Gain
Skinny64-64	32	2048	1936	5.5 %
Skinny64-128	36	2304	2176	5.6 %
Skinny64-196	40	2560	2416	5.6 %
Skinny128-128	40	5120	4972	3 %
Skinny128-256	48	6144	5964	3 %
Skinny128-384	56	7168	6956	3 %

2 Multiplicative Complexity of Conjugacy of Unstructured Functions

This section introduces the notations and results regarding implementations, multiplicative complexity, and conjugacy of unstructured functions. We begin by presenting the concept

of multiplicative complexity. Next, we examine how conjugacy impacts multiplicative complexity. Finally, we describe a procedure to identify conjugate candidates exhibiting low multiplicative complexity.

2.1 Multiplicative Complexity

In this section, we first define and then describe general properties of the multiplicative complexity of Boolean and vectorial Boolean functions.

Multiplicative Complexity. To define the multiplicative complexity of a Boolean function, we adopt the point of view of [Sto16], in which a vectorial Boolean function $f : \mathbb{F}_2^n \to \mathbb{F}_2^m$ can be implemented using a circuit composed of NOT, XOR, and AND gates. The *multiplicative complexity* of f, denoted by $\mu(f)$, is defined as the minimum number of AND gates required to compute f.

Remark 1. The *multiplicative complexity*, sometimes referred to as *masking complexity*, can be defined as the number of non-linear multiplications required to evaluate the polynomial representation of an S-box over $\mathbb{F}_{2^k}$. However, operations over extension fields hide the actual cost at the binary level. Indeed, implementing one multiplication in $\mathbb{F}_{2^k}$ requires several multiplications over $\mathbb{F}_2$. Winograd [Win80] established that $2k-1$ is a lower bound on the number of $\mathbb{F}_2$ multiplications required. Therefore, counting multiplications over $\mathbb{F}_2$ provides a finer and more implementation-relevant measure of complexity. Following the approach of [ZJ14], in this paper we focus on multiplicative complexity over $\mathbb{F}_2$ rather than over extension fields.

We now present two useful results about multiplicative complexity. The first establishes a triangular inequality that is at the basis of the multiplicative reduction threshold presented in the framework of Section 3. The second presents a connection between multiplicative complexity and linear invariants. The contrapositive of this result allows a speedup for the procedure presented in Subsection 2.3.

Property 1. Let $S_1 : \mathbb{F}_2^n \to \mathbb{F}_2^n$ and $S_2 : \mathbb{F}_2^n \to \mathbb{F}_2^n$ be two vectorial Boolean functions.

$$\mu(S_1 \circ S_2) \leq \mu(S_1) + \mu(S_2).$$

We note that $\mu(S_1 \circ S_2)$ is a minimum over all possible implementations. Implementing S_2 and then S_1 is an implementation candidate with cost $\mu(S_1)+\mu(S_2)$. Hence, $\mu(S_1 \circ S_2) \leq \mu(S_1) + \mu(S_2)$.

The next result presents similar ideas to the ones presented in [ZJ14]. To the best of our knowledge, this is the first time it is properly formalized.

Theorem 1. *Let $S : \mathbb{F}_2^n \to \mathbb{F}_2^n$. If $\mu(S) < n$, then there exist $\alpha, \beta \in \mathbb{F}_2^n$ and $c \in \mathbb{F}_2$ such that*

$$\forall x \in \mathbb{F}_2^n, \quad \langle \alpha, x \rangle \oplus \langle \beta, S(x) \rangle = c.$$

In other words, there exists a non-trivial linear invariant relating the input and output.

The idea of the proof is to represent S as an affine map defined on a space of dimension $n + \mu(S)$, where the last $\mu(S)$ coordinates correspond to the outputs of the multiplications. The condition $\mu(S) < n$ then implies, through a dimensional argument, the existence of a linear invariant. Details are done in Appendix A.

Theorem 1 states that a minimal number of multiplications is necessary to eliminate all linear invariants. In Subsection 2.3, we will use the contrapositive of this theorem to exclude functions that require at least n multiplications to implement.

2.2 Conjugacy

In the context of conjugate cipher cryptanalysis [Bau24], the authors use conjugacy by a *non-affine* function to disturb the security properties of the cipher that were stable under *affine equivalence*. In our context, we use the same idea: we consider conjugate ciphers to reduce a quantity that is stable under affine equivalence, namely multiplicative complexity. This section gives the definition of affine equivalence and describes conjugacy as navigation between affine equivalence classes.

Definition 1 (Affine Equivalence)**.** Two functions $f, g : \mathbb{F}_2^n \to \mathbb{F}_2^n$ are said to be *affine equivalent* if there exists affine permutations $A, B : \mathbb{F}_2^n \to \mathbb{F}_2^n$ such that

$$g = B \circ f \circ A.$$

Multiplicative Complexity. Since implementing an affine function does not require any multiplication, two affine equivalent functions have the same multiplicative complexity.

Conjugacy. Given a function S, its conjugate by a function G is defined as

$$S^G = G \circ S \circ G^{-1}.$$

If G is affine, then the multiplicative complexity remains unchanged since $\mu(S^G) = \mu(S)$. Thus, conjugacy by affine functions does not help reduce the number of multiplications. To explore different affine equivalence classes, conjugation by *non-affine* functions is necessary (see Figure 1).

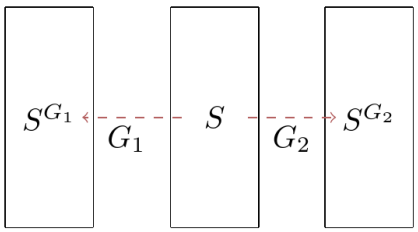

Figure 1: Conjugacy, represented by dashed red arrows, allows navigation between affine equivalence classes depicted as rectangles.

In the next section, we describe an iterative procedure that uses conjugacy to navigate between affine classes in order to identify one with low multiplicative complexity.

2.3 Conjugacy of Unstructured Functions

This part is intended to explore conjugate functions, in particular S-boxes. For an n-bit S-box, our goal is to find a vectorial Boolean function $G : \mathbb{F}_2^n \to \mathbb{F}_2^n$ such that the conjugate S-box S^G exhibits a smaller multiplicative complexity. There exist 2^{2^n} vectorial Boolean functions of n bits. Performing an exhaustive search over all possible G becomes computationally unfeasible, especially when $n \geq 4$.

Selecting G. To reduce the search space, we first investigate G functions presented in [Bau24], in line with previous cryptanalytic work presented in [BBF+25, BCL18]. This class has proven to be sparse enough to allow enumeration, and to be diverse enough to

navigate among numerous affine equivalence classes. These bijections G are parameterized by a nonlinear Boolean function g of $n-1$ variables:

$$G_g(x_1, \cdots, x_n) = (x_1 + g(x_2, \cdots, x_n), x_2, \cdots, x_n).$$

Such bijections are in fact involutive and correspond to Feistel-like structures. There are $2^{2^{n-1}}$ choices for the function G_g. As presented, this definition features a strong alignment property for all but the first coordinate. This property disappears once we compose G_g with a linear map L, effectively distributing the non-linearity to other coordinates. The linear map of choice is $L_a : \mathbb{F}_2^n \to \mathbb{F}_2^n$, defined by a parameter $a \in \mathbb{F}_2^n \setminus \{0\}$ chosen arbitrarily. The map L_a is then defined to satisfy $a := L_a^{-1}(e_0)$. The images of the $n-1$ remaining elements of the canonical basis $e_1, \cdots, e_{n-1}$ are selected to complete the rank of the matrix. This process defines a family of $2^n - 1$ distinct linear bijections L_a. In the end, we work with a family of $(2^n - 1)2^{2^{n-1}}$ functions of the form: $G_{g,a} := G_g \circ L_a^{-1}$.

An Exhaustive Procedure for 4-bit S-boxes. In the case of 4-bit S-boxes, the search space is reduced to $2^8 \cdot (16-1) = 3840$ elements. Hence, we can perform an exhaustive search by enumerating all possible pairs (g, a) with $g : \mathbb{F}_2^3 \to \mathbb{F}_2$ and $a \in \mathbb{F}_2^4$. To evaluate the multiplicative complexity of all the tested conjugates $S^{G_{g,a}}$, we rely on an external solver such as the one in [Sto16]. The solver decides whether a circuit using a fixed number of AND gates can implement a given S-Box, thereby providing an upper bound on its multiplicative complexity. As the allowed number of AND gates increases, the computation time of the solver grows; however, for 4-bit S-boxes (that cannot have a multiplicative complexity strictly superior than 5, see [ZJ14]), the multiplicative complexity can still be estimated tightly and in a reasonable amount of time. Nonetheless, this solver requires a large number of operations, and iterating over all 3840 elements is not feasible. To reduce the number of solver calls, we use the contrapositive of Theorem 1. Indeed, for each conjugate $S^{G_{g,a}}$, we first look for a linear invariant. If there is none, we skip to the next conjugate since we know that the multiplicative complexity in this case will be at least 4. If there is one, we use the solver and hope for a multiplicative complexity strictly below 4. We give in Algorithm 1 a precise description of the procedure. The Python code is provided as supplementary material.

Commentary on Table 2 We apply this approach to several 4-bit S-boxes and observe that the impact of conjugacy can be classified into three categories:

- **Multiplicative reduction.** This is the most common case. There exists a conjugate S-box exhibiting lower multiplicative complexity.
- **Multiplicative movement.** This phenomenon is illustrated by Serpent S4. In this case, the contrapositive of Theorem 1 reveals a linear invariant, while the multiplicative complexity remains unchanged. This indicates that the multiplication initially associated with the variable linearized through conjugacy has been transferred to another coordinate. Although this does not yield a reduction in multiplicative complexity, it may still be relevant for certain cryptanalytic purposes.
- **No gain.** This situation occurs for Serpent S7 and Serpent S5, where our method does not lead to an improved implementation. This may suggest either that the S-box is already optimal within its entire equivalence class or that the set of composition functions G considered here is overly restrictive.

[1] The contrapositive of Theorem 1 is not satisfied for any of the conjugates. This means that all the possible conjugates have a higher multiplicative complexity than that of the original S-box. Thus, this multiplicative complexity equals 5.

Algorithm 1 Search for low multiplicative complexity conjugates

Require: Sbox S
Ensure: Best pairs (g, a) minimizing $\mu(S^{G_{g,a}})$

Precompute all L_a^{-1} for $a \in \mathbb{F}_2^4 \setminus \{0\}$
$best \leftarrow \infty$
$solutions \leftarrow \emptyset$
for each Boolean function $g : \mathbb{F}_2^3 \to \mathbb{F}_2$ **do**
    Construct G_g
    for each $a \in \mathbb{F}_2^4 \setminus \{0\}$ **do**
        $G_{g,a} \leftarrow G_g \circ L_a^{-1}$
        Compute $G_{g,a}^{-1}$
        $S^{G_{g,a}} \leftarrow G_{g,a} \circ S \circ G_{g,a}^{-1}$
        if $S^{G_{g,a}}$ has a linear invariant **then** ▷ Contrapositive of Theorem 1
            $\mu \leftarrow$ multiplicative complexity of $S^{G_{g,a}}$
            if $\mu \leq best$ **then**
                $best \leftarrow \mu$
                $solutions \leftarrow \{(g, a)\}$
            end if
        end if
    end for
end for
return $best, solutions$

Table 2: Multiplicative complexity of 4-bit S-boxes and their associated conjugates. The first column gives the origin of each S-box, the second displays the best multiplicative evaluation cost known to implement these S-boxes. The last two columns give the results of Algorithm 1. In particular, the third column presents one of the best composition functions, and the last column gives the multiplicative cost of the conjugate S-box.

Reference	$\mu(S)$	(g, a)	$\mu(S^{G_{g,a}})$
PRESENT [BKL+07]	4	$(x_1x_2 + x_1x_3 + x_2x_3, 0001)$	3
PRINCE [BCG+12]	5	$(x_2x_3, 0001)$	4
RECTANGLE [ZBL+15]	4	$(x_1x_3, 0001)$	3
GIFT [BPP+17]	4	$(x_1x_3, 0011)$	3
SERPENT S0 [BAK98]	4	$(x_1x_3, 0010)$	3
SERPENT S1 [BAK98]	4	$(x_1x_3, 0010)$	3
SERPENT S2 [BAK98]	4	$(x_1x_3 + x_2x_3, 0010)$	3
SERPENT S3 [BAK98]	4	$(x_1x_3 + x_1x_2x_3, 0001)$	3
SERPENT S4 [BAK98]	4	$(x_2x_3 + x_1x_2x_3, 0001)$	4
SERPENT S5 [BAK98]	4	$\emptyset$ [1]	4
SERPENT S6 [BAK98]	4	$(x_1x_2, 1110)$	3
SERPENT S7 [BAK98]	4	$\emptyset$ [1]	4
MIDORI64 [BBI+15]	4	$(x_1x_2, 0010)$	3
PICCOLO [SIH+11]	4	$(x_2x_3, 0001)$	2
SKINNY64 [BJK+16]	4	$(x_2x_3, 0001)$	2
$x \to x^3$	3	$(x_1x_3, 0010)$	2

In Section 3 and Section 4, we show how to extend the evaluation cost gain observed for a single S-box to the entire cipher.

2.4 Conjugacy for Larger Functions

Foreshadowing Section 3, the multiplicative complexity of g plays a role in the complexity of the conjugate block cipher. It is therefore important to evaluate it precisely. In the next paragraph, we present a method to compute the set of all g with a given multiplicative complexity. As a consequence, when enumerating all g to construct $G_{g,a}$, it will be possible to proceed by increasing multiplicative complexity.

Enumeration process for g. For an integer k, let $\mathcal{P}_k$ denote the set of polynomials with multiplicative complexity k, i.e.,

$$\mathcal{P}_k = \{g : \mathbb{F}_2^n \to \mathbb{F}_2 \mid \mu(g) = k\}.$$

A polynomial of multiplicative complexity k can be obtained in two ways:

- **Addition.** By adding two polynomials, one of multiplicative complexity i and the other of multiplicative complexity $k - i$:

$$g = g_1 \oplus g_2.$$

- **Multiplication.** By multiplying two polynomials, one of multiplicative complexity i and the other of multiplicative complexity $k - i - 1$:

$$g = g_1 \times g_2.$$

As stated, these constructions yield polynomials of multiplicative complexity at most k. To ensure that the complexity is *exactly* k, we must verify that the resulting polynomial does not belong to $\mathcal{P}_i$ for any $i < k$. This leads to the following expression:

$$\mathcal{P}_k = \left(\bigcup_{i<k/2} \{g_1 \oplus g_2 \mid g_1 \in \mathcal{P}_i,\ g_2 \in \mathcal{P}_{k-i}\} \right) \tag{1}$$

$$\cup \left(\bigcup_{i<k/2} \{g_1 \times g_2 \mid g_1 \in \mathcal{P}_i,\ g_2 \in \mathcal{P}_{k-i-1}\} \right) \setminus \left(\bigcup_{i<k} \mathcal{P}_i \right). \tag{2}$$

Equation 1 describes an iterative procedure that constructs $\mathcal{P}_k$ from the previously computed sets $\mathcal{P}_i$. The base case $\mathcal{P}_0$, which corresponds to linear polynomials, can be computed efficiently and thus provides the initialization. This yields the process described in Algorithm 2.

The only modification with respect to the above description is that the addition loop starts at $i = 1$, and the multiplication loop includes an additional linear term g_3, so that $g = g_1 \times g_2 \oplus g_3$. This adjustment ensures that the algorithm terminates and avoids infinite loops.

3 Conjugate Block Ciphers

In this section, we present the conjugacy approach in the context of block ciphers to reduce their multiplicative cost.

Let $\mathcal{E} : k \mapsto (E_k : \mathbb{F}_2^n \to \mathbb{F}_2^n)$ be an *iterative block cipher* built as the iteration of r round functions $(F_k^{(i)})_{0 \le i \le r-1}$:

$$E_k = F_k^{(r-1)} \circ \cdots \circ F_k^{(0)}.$$

Algorithm 2 Iterative Procedure for building $\mathcal{P}_k$

Require: $k, n \in \mathbb{N}$
Ensure: Polynomial class $\mathcal{P}_k$

$\mathcal{P}_0 \leftarrow$ **Affine polynomials**
for $j = 1$ to k **do**
  $\mathcal{P}_j \leftarrow \emptyset$
  for $i = 1$ to $\lfloor j/2 \rfloor$ **do** ▷ Addition
    for $g_1 \in \mathcal{P}_i$ **do**
      for $g_2 \in \mathcal{P}_{j-i}$ **do**
        if $g_1 \oplus g_2 \notin \cup_{i<j} \mathcal{P}_i$ **then**
          $\mathcal{P}_j \leftarrow \mathcal{P}_j \cup \{g_1 \oplus g_2\}$
        end if
      end for
    end for
  end for
  for $i = 0$ to $\lfloor j/2 \rfloor$ **do** ▷ Multiplication
    for $g_1 \in \mathcal{P}_i$ **do**
      for $g_2 \in \mathcal{P}_{j-i-1}$ **do**
        for $g_3 \in \mathcal{P}_0$ **do**
          if $g_1 \times g_2 \oplus g_3 \notin \cup_{i<j} \mathcal{P}_i$ **then**
            $\mathcal{P}_j \leftarrow \mathcal{P}_j \cup \{g_1 \times g_2 \oplus g_3\}$
          end if
        end for
      end for
    end for
  end for
end for
return $\mathcal{P}_k$

When considering the conjugate cipher $E_k^{\mathcal{G}}$ by the function $\mathcal{G} : \mathbb{F}_2^n \to \mathbb{F}_2^n$, we obtain a cipher composed of the conjugate round functions:

$$E_k^{\mathcal{G}} = \mathcal{G} \circ E_k \circ \mathcal{G}^{-1} = (F_k^{(r-1)})^{\mathcal{G}} \circ \cdots \circ (F_k^{(0)})^{\mathcal{G}}.$$

To implement $\mathcal{E}$, one can use the relation $E_k = \mathcal{G}^{-1} \circ E_k^{\mathcal{G}} \circ \mathcal{G}$ to derive an implementation with the following multiplicative cost:

$$\mu(E_k^{\mathcal{G}}) + \mu(\mathcal{G}) + \mu(\mathcal{G}^{-1}) \leq r \cdot \mu(F_k^{\mathcal{G}}) + \mu(\mathcal{G}) + \mu(\mathcal{G}^{-1}), \tag{3}$$

assuming that all round functions have the same multiplicative cost.

As shown in Equation 3, finding a conjugate round function that requires fewer multiplications should lead to an implementation with a lower overall multiplicative cost (provided that the constant cost $\mu(\mathcal{G}) + \mu(\mathcal{G}^{-1})$ is compensated by the accumulated gain over the r rounds). To pursue our analysis, we need to understand the structure of conjugate round functions, and more precisely the conjugate round operations. In this work, we study round functions composed of the following operations:

- **S-box Layer.** Given an S-box size n_s, the S-box layer $\mathcal{S}$ consists of the parallel application of an S-box $S : \mathbb{F}_2^{n_s} \to \mathbb{F}_2^{n_s}$ to the $n_l = n/n_s$ parts of the internal state.
- **Key and Constant Addition.** These operations XOR either a constant C (AddC) or a round key K_i (AddK) to the internal state.
- **Linear Layer.** The linear layer $\mathcal{L}$ applies a linear transformation to the internal state.

We obtain the following upper bound for the multiplicative cost of a round function:

$$\mu(F_k^{\mathcal{G}}) \leq \mu(\mathcal{S}^{\mathcal{G}}) + \mu(\mathcal{L}^{\mathcal{G}}) + \mu(\mathrm{AddK}^{\mathcal{G}}) + \mu(\mathrm{AddC}^{\mathcal{G}}).$$

In the original round function, the multiplicative cost is entirely concentrated in the S-box layer. This is no longer the case for conjugate round functions, since conjugacy of linear operations by a non-linear $\mathcal{G}$ may introduce non-linearity. Hence, a gain on the round function requires that the reduction in the S-box layer outweighs the additional cost introduced in the other layers.

In the following, we study each conjugate operation and highlight key properties enabling multiplicative reduction. These steps are inspired by the cryptanalytic work described in [Bau24].

3.1 S-box Layer

In this section, we study the conjugate S-box layer. We begin by introducing a structure for $\mathcal{G}$ that matches the parallelism of $\mathcal{S}$.

Adding Structure to $\mathcal{G}$. Since the S-box layer $\mathcal{S}$ is a parallel application of an S-box S:

$$\begin{array}{rcl} \mathcal{S} : (\mathbb{F}_2^{n_s})^{n_l} & \to & (\mathbb{F}_2^{n_s})^{n_l} \\ (x_i)_{1 \leq i \leq n_l} & \mapsto & (S(x_i))_{1 \leq i \leq n_l}, \end{array}$$

we have $\mathcal{S} = (S|\cdots|S)$. We consider functions $\mathcal{G}$ that follow the same parallel structure. Hence, $\mathcal{G}$ is decomposed as $(G_i)_{1 \leq i \leq n_l}$ with $\mathcal{G} = (G_1|\cdots|G_{n_l})$.

This structure turns the conjugate S-box layer into a layer of conjugate S-boxes: $\mathcal{S}^{\mathcal{G}} = (S^{G_1}|\cdots|S^{G_{n_l}})$. We are thus left with the study of conjugate S-boxes, whose general case has been discussed in Subsection 2.3. As mentioned, this approach may become limited as the S-box dimension increases. Nevertheless, we can exploit the implementation structure of S-boxes to find interesting candidates for the functions G.

The Case of Iterated S-boxes. Similarly to iterative block ciphers, some S-boxes (ex: [BJK+16]) are constructed as iterations of a smaller function composed of a *multiplication layer* M and an *affine layer* A. Since outer affine layers do not add security due to affine equivalence, they are often removed for simplicity. We then obtain expressions of the form:

$$S = M \circ A \circ \cdots \circ M. \tag{4}$$

When the multiplicative layer is involutive (e.g., in-place multiplications of the form $M : (x_1, \cdots, x_m) \mapsto (x_1 \oplus m(\mathbf{x}), \cdots, x_m)$), Equation 4 expresses S as a conjugate of a simpler function by M. Consequently:

$$S^M = A \circ M \circ \cdots \circ A.$$

Maximal Gain. From Property 1, we derive the following bound on multiplicative reduction for a single S-box:

$$\mu(S^G) \geq \mu(S) - 2\mu(G),$$

which establishes optimality in the context of iterated S-boxes.

A similar inequality holds for the entire S-box layer: $\mu(\mathcal{S}^{\mathcal{G}}) \geq \mu(\mathcal{S}) - 2\mu(\mathcal{G})$, thus limiting the gain to $2\mu(\mathcal{G})$. Therefore, the additional cost introduced in subsequent layers must remain below this threshold to ensure a global reduction.

3.2 Key and Constant Addition

We now tackle key and constant addition, as they can be treated similarly. We follow the approach of [Bau24]. Since AddC and AddK represent parallel addition of material, we will keep the same notation for the n_s-bit restrictions studied in this subsection. We consider functions of the form $G_{g,a} = G_g \circ L_a^{-1}$ with:

$$G_g(x_1, x_2, \cdots, x_{n_s}) = (x_1 + g(x_2, \cdots, x_{n_s}), x_2, \cdots, x_{n_s}),$$

which encompasses both in-place multiplications and the general framework of Subsection 2.3. When $L_a^{-1} = \mathrm{Id}$, the conjugate of a key addition is:

$$\begin{aligned} G_g \circ \mathrm{AddK} \circ G_g^{-1}(x) &= G_g \circ \mathrm{AddK}(x_1 + g(x_2, \cdots, x_{n_s}), x_2, \cdots, x_{n_s}) \\ &= G_g(x_1 + k_1 + g(x_2, \cdots, x_{n_s}), x_2 + k_2, \cdots, x_{n_s} + k_{n_s}) \\ &= (x_1 + k_1 + g(x_2, \cdots, x_{n_s}) + g(x_2 + k_2, \cdots, x_{n_s} + k_{n_s}), x_2 + k_2, \cdots, x_{n_s} + k_{n_s}) \\ &= \mathrm{AddK} \circ G_{D_{\delta g}}(x), \end{aligned}$$

where $D_{\delta g}$ denotes the derivative of g in direction $\delta = (k_2, \cdots, k_{n_s})$. A similar relation holds for constant addition: $\mathrm{AddC}^G = G_{D_{\delta g}} \circ \mathrm{AddC}$ with $\delta = (c_2, \cdots, c_{n_s})$.

For the general case $L_a^{-1} \neq \mathrm{Id}$, we obtain:

$$\mathrm{AddK}^{G_{g,a}} = \mathrm{Add}(L_a^{-1}(K)) \circ G_{D_{\delta g}}.$$

Thus, addition layers require the computation of the derivative $G_{D_{\delta g}}$.

Adding Structure to g. If g is quadratic, then $D_{\delta g}$ becomes linear in $\mathbf{x}$, reducing the multiplicative cost of constant addition to zero. To obtain linear key addition, the key needs to be considered as a secret constant. This scenario fits transciphering applications but do not fit masked implementations of ciphers since round key computations are shared.

3.3 Linear Layer

In Subsection 3.1, we chose to work with functions $\mathcal{G}$ of the form $\mathcal{G} = (G_1|\cdots|G_{n_l})$ to follow the Sbox layer format. To ease our analysis, we focus on fully aligned linear layers, i.e., linear layers $\mathcal{L}$ lifted from a linear map $L : \mathbb{F}_2^{n_l} \to \mathbb{F}_2^{n_l}$, that do not perform intra-block mixing. More precisely,

$$\forall 1 \leq i \leq m, \quad \mathcal{L}(\mathbf{x})_i = L(\mathbf{x}_i),$$

where the whole internal state is depicted by $\mathbf{x} = (x^{(1)}|\cdots|x^{(n_l)})$, the first n_s bits are denoted by $x^{(1)} = (x_1^{(1)}, \cdots, x_{n_s}^{(1)})$ and the n_l bits of maginitude i are denoted by $\mathbf{x}_i = (x_i^{(1)}|\cdots|x_i^{(n_l)})$.

Matrix Approach. The analysis in Subsection 3.1 shows that $2\mu(\mathcal{G})$ is the maximal gain for the S-box layer. We now describe a case where $\mathcal{L}^{\mathcal{G}}$ requires fewer multiplications than this threshold. Consider the matrix of L whose row i is zero everywhere except for a single 1 in position j:

$$L = \begin{bmatrix} * & \cdots & * & \cdots & * \\ \vdots & \ddots & \vdots & & \vdots \\ * & \cdots & * & \cdots & * \\ 0 & \cdots & 1 & \cdots & 0 \\ * & \cdots & * & \cdots & * \\ \vdots & & \vdots & \ddots & \vdots \\ * & \cdots & * & \cdots & * \end{bmatrix}.$$

This corresponds to copying data from block j to block i.

If $\mathcal{G}$ applies G to blocks i and j (and identity elsewhere), then:

$$\mathcal{G} = (Id|\cdots|Id|G|\cdots|Id|G|\cdots|Id).$$

The implementation of $\mathcal{L}^{\mathcal{G}} = \mathcal{G} \circ \mathcal{L} \circ \mathcal{G}^{-1}$ requires:

- Two G implementations to compute the initial $\mathcal{G}^{-1}$ (since G_g is involutive).
- One G implementation on block j, since the block i is equal to $G \circ G^{-1}(x^{(j)}) = x^{(j)}$.

Hence, the total cost is $3\mu(G) < 2\mu(\mathcal{G}) = 4\mu(G)$. This technique can be repeated for each suitable row of L to further reduce the overall multiplicative complexity.

In the next section, we present an application of this method to the Skinny family of block cipher [BJK+16], which satisfies all required conditions: iterated S-boxes and fully aligned linear layers with copy operations.

4 Application to the Skinny Family of Block Ciphers

In this section, we apply the conjugate cipher approach to reduce the multiplicative cost of the Skinny family of block ciphers [BJK+16]. We first describe Skinny, then present its conjugate version.

4.1 Specifications of Skinny

Skinny is a lightweight family of tweakable iterative block ciphers based on a substitution–permutation network structure, proposed at CRYPTO 2016 [BJK+16]. The Skinny family consists of block ciphers that differ in internal state size ($n = 64$ or $n = 128$ bits) and tweakey size (n, $2n$, or $3n$), resulting in a total of six block ciphers.

a_0	a_4	a_8	a_{12}
a_1	a_5	a_9	a_{13}
a_2	a_6	a_{10}	a_{14}
a_3	a_7	a_{11}	a_{15}

where the a_i are elements of $\mathbb{F}_2^4$

Figure 2: State of SKINNY64

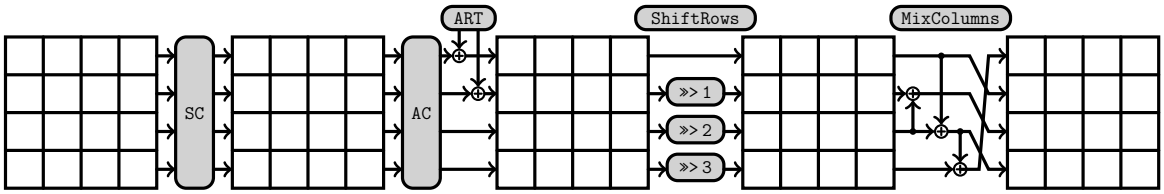

Figure 3: Round Function of SKINNY64

Internal State. Similarly to AES, the internal state depicted in Figure 2 consists of a 4×4 matrix, where each cell corresponds to either a nibble (SKINNY64) or a byte (SKINNY128).

Round function. The round function depicted in Figure 3 is composed of the following operations:

- SubCells consists in the parallel application of an S-box on each of the cells and will be fully described later in Subsection 4.2.
- AddConstants XORs a constant to the internal state.
- AddRoundTweakey XORs the round tweakey to the two top rows.
- ShiftRows rotates the cells of the second, third and fourth row by respectively one, two and three positions to the right.
- MixColumns performs the multiplication of each column by the matrix

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

In the next subsections, we follow the framework from Section 3 and show the effect of the conjugacy on the different layers of SKINNY64.

4.2 Conjugate S-box Layer

Depicted in Figure 4, the S-box SKINNY64 can be implemented as the iteration of smaller functions. This allows to follow the framework introduced in Subsection 3.1. Indeed, S follows the equation:

$$S = G \circ R \circ G \circ R \circ G \circ R \circ G, \tag{5}$$

where R is a left bit rotation and G is defined by:

$$G : (x_1, x_2, x_3, x_4) \to [x_1 + (x_3 + 1)(x_4 + 1), x_2, x_3, x_4].$$

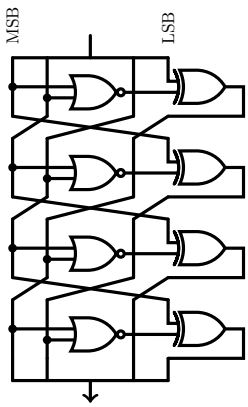

Figure 4: S-box of SKINNY64

From the stucture described by Equation 5, we see that conjugating S by G will cancel two multiplications thereby reducing the number of multiplications by 2 for each conjugate block.

The conjugated S-box $S^G = R \circ G \circ R \circ G \circ R$ requires two multiplications and corresponds to the following ANF:

$$\begin{cases} y_1 = x_2 \\ y_2 = 1 + x_1 + x_2 + x_1x_2 + x_3 \\ y_3 = 1 + x_2 + x_3 + x_2x_3 + x_4 \\ y_4 = x_1 \end{cases}$$

Multiplicative Cost . The multiplicative cost of a conjugated S-box is $\mu(S^G) = \mu(S) - 2\mu(G) = 2$. The cost for the whole S-box layer depends of the number n_G of parallel application G present in $\mathcal{G}$. Indeed

$$\mu(\mathcal{S}^{\mathcal{G}}) = n_G \cdot \mu(S^G) + (n_l - n_G)\mu(S) = 2n_G + 4(n_l - n_G)$$

The number n_G is defined by the linear layer (Subsection 4.4).

4.3 Conjugate Addition Layer

The G function extracted from the S-box layer is built upon a quadratic g. As explained in Subsection 3.2, the conjugate of addition layer remains a linear operation.

4.4 Conjugate Linear Layer

The linear layer consists in the concatenation of ShiftRows and MixColumns. We now describe the conjugate version of both operations.

ShiftRows. This operation being a permutation of cells, it actually commutes with $\mathcal{G}$. Hence, its conjugate equals itself which leads to $\mu(SR^{\mathcal{G}}) = 0$.

G	G	G	G
G	G	G	G
Id	Id	Id	Id
Id	Id	Id	Id

Figure 5: $\mathcal{G}$ function for Skinny64 with parameters $n_G = 4$ and $\mu(G) = 1$.

MixColumns. We now focus on the second operation: MixColumns. This operation applies the linear transformation described by the matrix

$$MC = \begin{pmatrix} 1 & 0 & 1 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 0 \end{pmatrix},$$

to all columns. In Subsection 3.3, we developed an approach based on finding vectors of the form all 0 but one coordinate. This can be found in the second row of the matrix. As a consequence, placing a G on the first and second row induces an additional cost for MixColumns of $3\mu(G)$ strictly below the $2\mu(\mathcal{G}) = 4\mu(G)$. A precise description of $\mathcal{G}$ in function of G is depicted in Figure 5.

Multiplicative Cost. The multiplicative cost of the conjugate linear layer is:

$$\begin{aligned} \mu(\mathcal{L}^{\mathcal{G}}) &= \mu(SR^{\mathcal{G}}) + \mu(MC^{\mathcal{G}}) \\ &= 4 \cdot 3\mu(G) \\ &= 12 \end{aligned}$$

4.5 Multiplicative Cost Skinny64

We now describe the overall multiplicative cost for Skinny64 for standard and conjugate implementation and confront the costs.

Standard Implementation. The standard implementation of the cipher requires 4 multiplications per S-box. Since there are 16 S-box applications per round, it represents a total of $16 \cdot 4 = 64$ multiplications per round. Henceforth, the multiplicative cost for an r round encryption is $64r$.

Conjugate Implementation. Bringing together the elements of this section, we obtain the following multiplicative cost for the conjugate round function:

$$\mu(\mathcal{S}^{\mathcal{G}}) + \mu(\mathcal{L}^{\mathcal{G}}) = 2 \cdot 8 + 4 \cdot 8 + 12 = 60.$$

The multiplicative cost of an r round encryption is therefore given by the iterated cost of round functions and the initial conjugate cost:

$$60r + 2\mu(\mathcal{G}) = 60r + 16.$$

Confronting both formulae, we observe that conjugate implementation becomes advantageous as soon as $r \geq 4$. If the assymptotical gain is $4/64 = 6.25\%$, Table 3 presents both implementations for all Skinny64 versions and highlights a practical gain close to 5.5% for all versions.

Table 3: Multiplicative Complexity of all SKINNY64 Versions.

SKINNY64 Version	♯ Round	Standard (Nb. Mult.)	Conjugate (Nb. Mult.)	Gain
SKINNY64-64	32	2048	1936	5.5 %
SKINNY64-128	36	2304	2176	5.6 %
SKINNY64-196	40	2560	2416	5.6 %

Security of the conjugated cipher A natural question is whether replacing the original operations of SKINNY64 with their conjugated versions could introduce structural weaknesses. Reducing the number of multiplications might reveal linear invariants for the round function. We computed these invariants for the first conjugated round of SKINNY-64-128 and found 16. Table 4 lists the 16 invariants found. However, none of them extends beyond two consecutive rounds. To the best of our knowledge, these invariants do not seem to threaten SKINNY64 security.

Table 4: Linear invariants for the conjugate of the first round of SKINNY-64-128

Idx	α **(Input mask)**	γ **(Key mask)**	β **(Output mask)**
	Key-dependent invariants		
1	0x0000000000000001	0x00000008	0x0000000000080000
2	0x0000000000000010	0x00000080	0x0000000000800000
3	0x0000000000000100	0x00000800	0x0000000008000000
4	0x0000000000001000	0x00008000	0x0000000080000000
5	0x0000000000010000	0x00080000	0x0080008000800000
6	0x0000000000100000	0x00800000	0x0800080008000000
7	0x0000000001000000	0x08000000	0x8000800080000000
8	0x0000000010000000	0x80000000	0x0000800080008000
	Key-independent invariants		
9	0x0000000100000000	0x00000000	0x0800000008000000
10	0x0000001000000000	0x00000000	0x8000000080000000
11	0x0000010000000000	0x00000000	0x0000800000008000
12	0x0000100000000000	0x00000000	0x0008000000080000
13	0x0001000000000000	0x00000000	0x8000000000008000
14	0x0010000000000000	0x00000000	0x0008000000000008
15	0x0100000000000000	0x00000000	0x0080000000000080
16	0x1000000000000000	0x00000000	0x0800000000000800

4.6 Conjugacy of Skinny128

The conjugate of SKINNY128 is very similar to the 64-bit version. The only difference of importance to us is the S-box layer.

Conjugate S-box. Alike the 4bits version, the 8bits S-box of SKINNY128 can be described by an iterative implementation depicted in Figure 6.

The S-Box S is of the form

$$S = W \circ G \circ P \circ G \circ P \circ G \circ P \circ G,$$

where

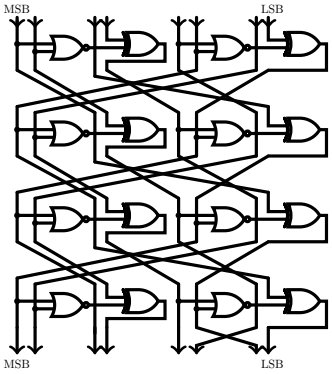

Figure 6: S-Box of Skinny128

- P is a particular permutation of eight bits explicitly defined in [BJK+16]
- W is a bit swap between x_2 and x_3
- G is the nonlinear part defined by (x_1 being the LSB) :

  $G : (x_1, x_2, x_3, x_4, x_5, x_6, x_7, x_8) \mapsto [x_1 + g(x_3, x_4), x_2, x_3, x_4, x_5 + g(x_7, x_8), x_6, x_7, x_8]$.

  where g : $\mathbb{F}_2^2 \to \mathbb{F}$ such that $g(a, b) = (a + 1)(b + 1)$

Unlike the conjugacy of the S-box of Skinny64, when it comes to Skinny128 no particular conjugate appears in the expression S due to the presence of a bit swap (W) at the last step that does not occur for Skinny64. Nonetheless, it is still possible to fit into the conjugate framework by splitting G into two functions G_1 and G_2. Indeed G acts independently and in parallel on the two halves of the input. Then, $G = G_1 \circ G_2$ ($= G_2 \circ G_1$) where G_1 processes the four first bits and G_2 the four last bits. We can rewrite the expression of the S-Box S as follows :

$$S = W \circ G_2 \circ G_1 \circ (P \circ G)^2 \circ P \circ G_1 \circ G_2$$

As W does not interfere with the second bit half we have that $W \circ G_2 = G_2 \circ W$ and then :

$$S = G_2 \circ W \circ G_1 \circ (P \circ G)^2 \circ P \circ G_1 \circ G_2 \tag{6}$$

Equation 6 exhibits a conjugate of S according to the application $G' := id \| G_2$. It is $S^{G'} = W \circ G_1 \circ (P \circ G)^2 \circ P \circ G_1$ with $\mu(S^{G'}) = 6$.

Linear Layer. $\mathcal{G}$ is built similarly to the 64-bits case. Indeed, G' is placed in the same position depicted in Figure 5, thereby representing an additional cost of $12\mu(G') = 12$ multiplications for the linear layer.

New Implementations Cost of Skinny128. For Skinny128, the standard implementation costs is $128r$ multiplications and the conjugate implementation is given by:

$$(6 \cdot 8 + 8 \cdot 8 + 12)r + 12 = 124r + 12.$$

Once again if $r \geq 4$, conjugate implementation requires fewer multiplication with an assymptotical gain of $4/128 = 3.125\%$. The practical costs are presented in Table 5, and the gain is around 3 % in all cases.

Table 5: Multiplicative Complexity of all SKINNY64 Versions.

SKINNY128 Version	♯ Round	Standard (Nb. Mult.)	Conjugate (Nb. Mult.)	Gain
SKINNY128-128	40	5120	4972	3 %
SKINNY128-256	48	6144	5964	3 %
SKINNY128-384	56	7168	6956	3 %

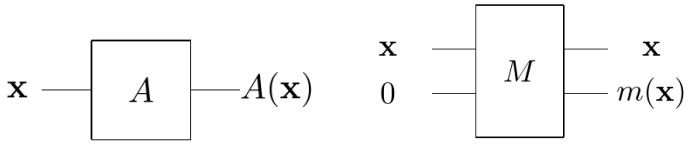

Figure 7: Circuit description of the affine layers (left) and multiplication layers (right).

5 Conclusion

In this work, we introduced a conjugacy-based framework to reduce the multiplicative complexity of block ciphers and demonstrated its effectiveness on the SKINNY family. By leveraging the iterative implementation of the S-box layer and the alignment of linear layer, we derived a composition function that leads to an alternative implementation for the cipher with strictly fewer multiplications. Our results show that this approach yields both asymptotic and practical improvements.

Future Work. Beyond the specific case of SKINNY, our work suggests that conjugacy can serve as a general tool for optimizing implementation of block ciphers. However, the efficiency of the method relies on a certain degree of structural compatibility between the nonlinear and linear layers. A natural direction for future work is therefore to extend this framework to more general settings, in particular to designs where round operations are less orthogonal or exhibit stronger interdependencies such as AES where the MIXCOLUMN operation shuffles bits within a cell. Understanding how conjugation behaves in such contexts could significantly broaden its applicability.

Another promising perspective lies in the interplay between cryptanalysis and implementation. While cryptanalysis is traditionally used to identify weaknesses, it can also reveal structural redundancies or algebraic properties that may be exploited to simplify implementations. Bridging these two viewpoints could lead to new optimization techniques, where insights from cryptanalysis directly inform the design of more efficient realizations of secure primitives.

A Proof of Theorem 1

To provide a natural proof for Theorem 1, we first define the circuit implementation.

Definition 2 (Implementation)**.** For a given function $S : \mathbb{F}_2^n \to \mathbb{F}_2^n$, an implementation is given by a sequence of $m+1$ affine layers $(A_i)_{0\leq i\leq m}$ and m multiplication layers $(M_i)_{1\leq i\leq m}$ such that:

- Each multiplication layer (Figure 7) leaves the input untouched and computes the boolean multiplication in a new variable:

$$\begin{array}{rccc} M_i : \mathbb{F}_2^{n+i-1} & \to & \mathbb{F}_2^{n+i} \\ \mathbf{x} & \mapsto & (\mathbf{x}, m_i(\mathbf{x})) \end{array}$$

  where $m_i(\mathbf{x})$ is a boolean function with $\mu(m_i) = 1$.

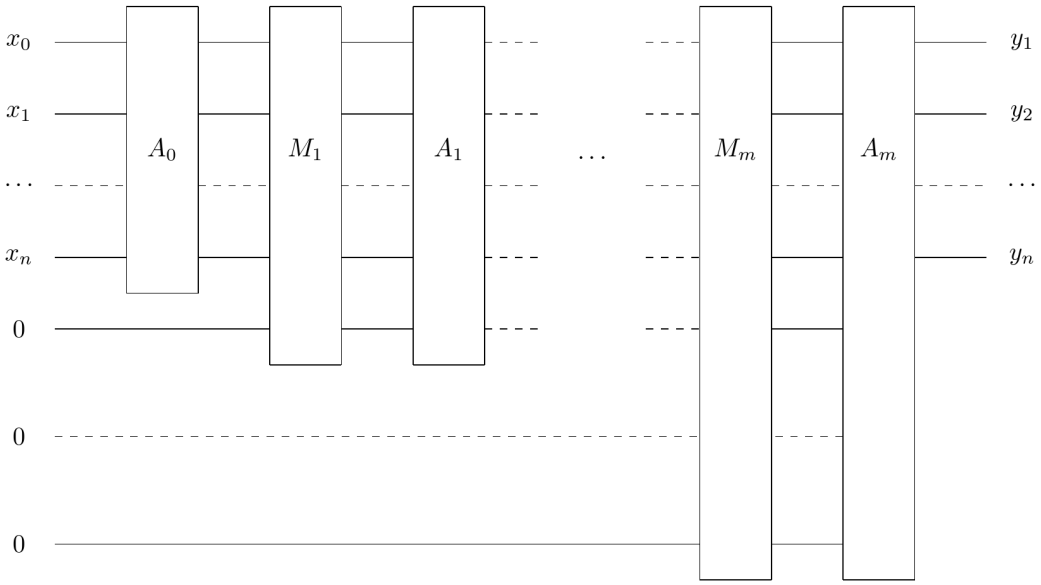

Figure 8: Implementation of the function S.

- Each affine layer (Figure 7) performs an affine transformation with the following dimensions:
$$\begin{cases} A_i : \{0,1\}^{n+i} \to \{0,1\}^{n+i} \text{ when } i < m, \\ A_m : \{0,1\}^{n+m} \to \{0,1\}^n. \end{cases}$$

The function S coincides with the concatenation of the circuits (Figure 8):

$$S = A_m \circ M_m \circ A_{m-1} \circ \cdots \circ M_1 \circ A_0.$$

For a vectorial function $S : \mathbb{F}_2^n \to \mathbb{F}_2^n$, the *multiplicative complexity*, denoted by $\mu(S)$, corresponds to the smallest size m associated with an implementation sequence $((A_i)_{0\leq i\leq m}, (M_i)_{1\leq i\leq m})$ that implements S.

Theorem 1 Let $S : \mathbb{F}_2^n \to \mathbb{F}_2^n$. If $\mu(S) < n$, then there exist $\alpha, \beta \in \mathbb{F}_2^n$ and $c \in \mathbb{F}_2$ such that

$$\forall x \in \mathbb{F}_2^n, \quad \langle \alpha, x\rangle \oplus \langle \beta, S(x)\rangle = c.$$

In other words, there exists a non-trivial linear invariant relating the input and output.

Proof. Let $m = \mu(S)$ and $((A_i)_{0\leq i\leq m}, (M_i)_{1\leq i\leq m})$ be an implementation of S. Let us consider the affine map $\mathcal{A}$ obtained by the concatenation of all affine layers:

$$\begin{array}{rcl} \mathcal{A} : \mathbb{F}_2^{n+m} & \to & \mathbb{F}_2^n \\ (\mathbf{x}, \mathbf{y}) & \mapsto & A_m\big(A_{m-1}(\cdots A_1(A_0(\mathbf{x}), y_1), \cdots, y_{m-1}), y_m\big), \end{array}$$

If $\mathbf{y}^*$ is define by induction as $y_i^* = m_i\big(A_{i-1}(\cdots A_1(A_0(\mathbf{x}), y_1^*), \ldots, y_{i-1}^*)\big)$, then the function S can be expressed as

$$S(\mathbf{x}) = \mathcal{A}(\mathbf{x}, \mathbf{y}^*)$$

Consider now the affine operator

$$\begin{array}{rcl} \mathcal{AA} : \mathbb{F}_2^{n+m} & \to & \mathbb{F}_2^{2n} \\ (\mathbf{x}, \mathbf{y}) & \mapsto & (\mathbf{x}, \mathcal{A}(\mathbf{x}, \mathbf{y})). \end{array}$$

By a dimension argument, since $m < n$, the rank of the linear function associated to $\mathcal{AA}$ is strictly less than $2n$. This implies that there exists a non-zero vector

$$\gamma = (\alpha, \beta) \in \mathbb{F}_2^{2n}$$

and a constant $c \in \mathbb{F}_2$ such that

$$\forall (\mathbf{x}, \mathbf{y}) \in \mathbb{F}_2^{n+m}, \quad \langle \gamma, (\mathbf{x}, \mathcal{A}(\mathbf{x}, \mathbf{y})) \rangle = c,$$

which is equivalent to

$$\forall (\mathbf{x}, \mathbf{y}), \quad \langle \alpha, \mathbf{x} \rangle \oplus \langle \beta, \mathcal{A}(\mathbf{x}, \mathbf{y}) \rangle = c.$$

In particular, by substituting $\mathbf{y} = \mathbf{y}^*$, we obtain

$$\forall \mathbf{x} \in \mathbb{F}_2^n, \quad \langle \alpha, \mathbf{x} \rangle \oplus \langle \beta, S(\mathbf{x}) \rangle = c,$$

which concludes the proof. □

References

[BAK98] Eli Biham, Ross J. Anderson, and Lars R. Knudsen. Serpent: A new block cipher proposal. In Serge Vaudenay, editor, *FSE'98*, volume 1372 of *LNCS*, pages 222–238. Springer, Berlin, Heidelberg, March 1998.

[Bau24] Jules Baudrin. *Algebraic properties of symmetric ciphers and of their non-linear components. (Propriétés algébriques de chiffrements symétriques et de leurs composants non-linéaires)*. PhD thesis, Sorbonne University, Paris, France, 2024.

[BBF+25] Jules Baudrin, Christof Beierle, Patrick Felke, Gregor Leander, Patrick Neumann, Léo Perrin, and Lukas Stennes. Commutative cryptanalysis as a generalization of differential cryptanalysis. *DCC*, 93(8):3243–3281, 2025.

[BBI+15] Subhadeep Banik, Andrey Bogdanov, Takanori Isobe, Kyoji Shibutani, Harunaga Hiwatari, Toru Akishita, and Francesco Regazzoni. Midori: A block cipher for low energy. In Tetsu Iwata and Jung Hee Cheon, editors, *ASIACRYPT 2015, Part II*, volume 9453 of *LNCS*, pages 411–436. Springer, Berlin, Heidelberg, November / December 2015.

[BCG+12] Julia Borghoff, Anne Canteaut, Tim Güneysu, Elif Bilge Kavun, Miroslav Knežević, Lars R. Knudsen, Gregor Leander, Ventzislav Nikov, Christof Paar, Christian Rechberger, Peter Rombouts, Søren S. Thomsen, and Tolga Yalçin. PRINCE - A low-latency block cipher for pervasive computing applications - extended abstract. In Xiaoyun Wang and Kazue Sako, editors, *ASIACRYPT 2012*, volume 7658 of *LNCS*, pages 208–225. Springer, Berlin, Heidelberg, December 2012.

[BCL18] Christof Beierle, Anne Canteaut, and Gregor Leander. Nonlinear approximations in cryptanalysis revisited. *IACR Trans. Symm. Cryptol.*, 2018(4):80–101, 2018.

[BDD+20] Begül Bilgin, Lauren De Meyer, Sébastien Duval, Itamar Levi, and François-Xavier Standaert. Low AND depth and efficient inverses: a guide on S-boxes for low-latency masking. *IACR Trans. Symm. Cryptol.*, 2020(1):144–184, 2020.

[BDM25] Marie Bolzer, Sébastien Duval, and Marine Minier. A new tool to find lightweight (and, xor) implementations of quadratic vectorial boolean functions up to dimension 9. *IEEE Transactions on Circuits and Systems I: Regular Papers*, 2025.

[BJK+16] 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 Matthew Robshaw and Jonathan Katz, editors, *CRYPTO 2016, Part II*, volume 9815 of *LNCS*, pages 123–153. Springer, Berlin, Heidelberg, August 2016.

[BKL+07] Andrey Bogdanov, Lars R. Knudsen, Gregor Leander, Christof Paar, Axel Poschmann, Matthew J. B. Robshaw, Yannick Seurin, and C. Vikkelsoe. PRESENT: An ultra-lightweight block cipher. In Pascal Paillier and Ingrid Verbauwhede, editors, *CHES 2007*, volume 4727 of *LNCS*, pages 450–466. Springer, Berlin, Heidelberg, September 2007.

[BMP13] Joan Boyar, Philip Matthews, and René Peralta. Logic minimization techniques with applications to cryptology. *Journal of Cryptology*, 26(2):280–312, April 2013.

[BPP+17] Subhadeep Banik, Sumit Kumar Pandey, Thomas Peyrin, Yu Sasaki, Siang Meng Sim, and Yosuke Todo. GIFT: A small present - towards reaching the limit of lightweight encryption. In Wieland Fischer and Naofumi Homma, editors, *CHES 2017*, volume 10529 of *LNCS*, pages 321–345. Springer, Cham, September 2017.

[CGP+12] Claude Carlet, Louis Goubin, Emmanuel Prouff, Michaël Quisquater, and Matthieu Rivain. Higher-order masking schemes for S-boxes. In Anne Canteaut, editor, *FSE 2012*, volume 7549 of *LNCS*, pages 366–384. Springer, Berlin, Heidelberg, March 2012.

[CJRR99] Suresh Chari, Charanjit S. Jutla, Josyula R. Rao, and Pankaj Rohatgi. Towards sound approaches to counteract power-analysis attacks. In Michael J. Wiener, editor, *CRYPTO'99*, volume 1666 of *LNCS*, pages 398–412. Springer, Berlin, Heidelberg, August 1999.

[CMH13] Nicolas Courtois, Theodosis Mourouzis, and Daniel Hulme. Exact logic minimization and multiplicative complexity of concrete algebraic and cryptographic circuits. *Int. J. Adv. Intell. Syst*, 6(3):165–176, 2013.

[CRV14] Jean-Sébastien Coron, Arnab Roy, and Srinivas Vivek. Fast evaluation of polynomials over binary finite fields and application to side-channel countermeasures. In Lejla Batina and Matthew Robshaw, editors, *CHES 2014*, volume 8731 of *LNCS*, pages 170–187. Springer, Berlin, Heidelberg, September 2014.

[DEMS21] Christoph Dobraunig, Maria Eichlseder, Florian Mendel, and Martin Schläffer. Ascon v1.2: Lightweight authenticated encryption and hashing. *Journal of Cryptology*, 34(3):33, July 2021.

[GP99] Louis Goubin and Jacques Patarin. DES and differential power analysis (the "duplication" method). In Çetin Kaya Koç and Christof Paar, editors, *CHES'99*, volume 1717 of *LNCS*, pages 158–172. Springer, Berlin, Heidelberg, August 1999.

[GR16] Dahmun Goudarzi and Matthieu Rivain. On the multiplicative complexity of Boolean functions and bitsliced higher-order masking. In Benedikt Gierlichs and Axel Y. Poschmann, editors, *CHES 2016*, volume 9813 of *LNCS*, pages 457–478. Springer, Berlin, Heidelberg, August 2016.

[GRVV17] Dahmun Goudarzi, Matthieu Rivain, Damien Vergnaud, and Srinivas Vivek. Generalized polynomial decomposition for S-boxes with application to side-channel countermeasures. In Wieland Fischer and Naofumi Homma, editors, *CHES 2017*, volume 10529 of *LNCS*, pages 154–171. Springer, Cham, September 2017.

[ISW03] Yuval Ishai, Amit Sahai, and David Wagner. Private circuits: Securing hardware against probing attacks. In Dan Boneh, editor, *CRYPTO 2003*, volume 2729 of *LNCS*, pages 463–481. Springer, Berlin, Heidelberg, August 2003.

[JPST17] Jérémy Jean, Thomas Peyrin, Siang Meng Sim, and Jade Tourteaux. Optimizing implementations of lightweight building blocks. *IACR Trans. Symm. Cryptol.*, 2017(4):130–168, 2017.

[MAPB23] Irati Manterola Ayala, Atharva Phanse, and Clémence Bouvier. STAP Zoo. https://stap-zoo.com/, 2023. Maintained by the STAP-Zoo Team and Zoo-keepers.

[NLV11] Michael Naehrig, Kristin Lauter, and Vinod Vaikuntanathan. Can homomorphic encryption be practical? In *Proceedings of the 3rd ACM workshop on Cloud computing security workshop*, pages 113–124, 2011.

[S+99] Data Encryption Standard et al. Data encryption standard. *Federal information processing standards publication*, 112(3), 1999.

[SIH+11] Kyoji Shibutani, Takanori Isobe, Harunaga Hiwatari, Atsushi Mitsuda, Toru Akishita, and Taizo Shirai. Piccolo: An ultra-lightweight blockcipher. In Bart Preneel and Tsuyoshi Takagi, editors, *CHES 2011*, volume 6917 of *LNCS*, pages 342–357. Springer, Berlin, Heidelberg, September / October 2011.

[Sto16] Ko Stoffelen. Optimizing S-box implementations for several criteria using SAT solvers. In Thomas Peyrin, editor, *FSE 2016*, volume 9783 of *LNCS*, pages 140–160. Springer, Berlin, Heidelberg, March 2016.

[Win80] Shmuel Winograd. *Arithmetic complexity of computations*, volume 33. Siam, 1980.

[ZBL+15] Wenling Zhang, Zhiyong Bao, Dongdai Lin, Vincent Rijmen, Bart Preneel, and Ingrid Verbauwhede. Rectangle: A bit-slice lightweight block cipher suitable for multiple platforms. *Science China Information Sciences*, 58(12):1–15, 2015.

[ZH23] Fuxin Zhang and Zhenyu Huang. Optimizing S-box implementations using SAT solvers: Revisited. Cryptology ePrint Archive, Report 2023/1721, 2023.

[ZJ14] Pavol Zajac and Matúš Jókay. Multiplicative complexity of bijective 4× 4 s-boxes. *Cryptography and Communications*, 6(3):255–277, 2014.