From Substitution Box To Threshold

Anubhab Baksi[1], Sylvain Guilley[2,3], Ritu-Ranjan Shrivastwa[2], and Sofiane Takarabt[2]

[1] Nanyang Technological University, Singapore

[2] Secure-IC, Rennes, France

[3] Télécom-Paris, Paris, France

anubhab001@e.ntu.edu.sg, sylvain.guilley@secure-ic.com,
ritu-ranjan.shrivastwa@secure-ic.com, sofiane.takarabt@secure-ic.com

Abstract. With the escalating demand for lightweight ciphers as well as side channel protected implementation of those ciphers in recent times, this work focuses on two aspects. First, we present a tool for automating the task of finding a Threshold Implementation (TI) of a given Substitution Box (SBox). Our tool returns 'with decomposition' and 'without decomposition' based TI. The 'with decomposition' based implementation returns a combinational SBox; whereas we get a sequential SBox from the 'without decomposition' based implementation. Despite being high in demand, it appears that this kind of tool has been missing so far. Second, we show an algorithmic approach where a given cipher implementation can be tweaked (without altering the cipher specification) so that its TI cost can be significantly reduced. We take the PRESENT cipher as our case study (our methodology can be applied to other ciphers as well). Indeed, we show over 31 percent reduction in area and over 52 percent reduction in depth compared to the basic threshold implementation.

Keywords: Lightweight Cryptography · Block Cipher · SBox · Side Channel Countermeasure · Threshold Implementation · PRESENT

1 Introduction

Over the last few years, we have observed a surge of research works dedicated to finding new lightweight ciphers and/or low-cost implementation of those ciphers. Recently we have also seen side channel protected implementation of the lightweight ciphers gaining traction [11,18]. While the community is proactive in advocating side channel protected implementation, surprisingly, a systematic and generic study on how to do that appears to be missing from the literature. There is an overall theory, but for the most part, a detailed study is required to be undertaken for better understanding of the context. The authors seem to come up with some ad-hoc approach (see, e.g., [18]) for the implementation. This further hinders the overdue tasks such as, finding proper algorithms, easy-to-use and publicly available tools, and various optimisations that can be employed to reduce the cost.

This work makes a humble attempt to look into the problem of finding a *Threshold Implementation* (TI) of a given SBox. The TI is a well-known concept that aims at protecting against the most common type of side channel attack which relies on power or electromagnetic leakage [6, 7, 18, 19]. Such SBoxes are probably most common choice for the non-linear component in the modern lightweight ciphers, thus an automated tool to find TI of those SBoxes is of prime importance.

Further, we observe that with a slight modification in the implementation, it is possible to reduce the cost for a TI of cipher. This does not alter the actual description of the cipher,

hence there is no need to redo the security analysis. We show how to adopt a systematic approach with the lightweight cipher PRESENT [9], and how it successfully reduces the TI cost, by more than 31% in terms of area or by more of 52% in terms of depth. This methodology is generic, hence it can be applied to other ciphers that use a similar structure.

Our Contributions

The side channel attack is considered among the major threats, though, the field of studying the protected implementations seems less than developed at multiple levels. While working on this general area, we find ourselves in a situation where we need/want to find threshold implementation of a fairly large pool of SBoxes. Effectively, we look for automated tools to do the batch processing, but to our dismay, cannot find anything suitable (except for the tool by Petkova-Nikova mentioned in Section 4). Parallel to this, the problem of finding some optimisations at the algorithmic level is another interesting direction, which seems underdeveloped too.

Ultimately, we decide to make our own tool for this purpose. As we go along, we discover a proper algorithm is missing. For the most part, the previous authors such as [18, Section III] or [6, Chapter 3] convey the idea behind threshold through examples, instead of a well-formed algorithm. Naturally, several pertinent issues remain unexplored, such as automating the process or dealing with the corner cases. We therefore look further down with adequate clarity to come up with an algorithm as well as an open-source implementation of it[1]. Our tool has two segments, one segment returns 'without decomposition' based threshold implementation (Section 3), and the other segment returns 'with decomposition' based threshold implementation (Section 4).

On the other hand, better understanding of TI motivates us to find other avenues for cost reduction. With the lightweight block cipher PRESENT [9], we show an optimisation strategy (Section 5) that uses a less resources. The basic idea stems from the concept of *affine equivalence* (Section 2.2) of SBoxes, i.e., by opting for another SBox than specified by the designers. This SBox is chosen in such as a way that the netlist optimisation tool can leverage on the new SBox's algebraic properties. However, our approach does not change the overall cipher specification (so one does not need to carry out the usual security analysis). Thus, our analysis reveals, some alternative representations are indeed more efficient to implement than the naïve implementation (which used as a baseline). Further, this strategy can be applied to any other cipher with a similar structure.

2 Background

2.1 Möbius Transform

The Möbius transform is used to find the coordinate function representation (in ANF) of a given (bijective) SBox. For the sake of better clarity, it is described here. Given an $n \times n$ SBox S, we first need to define the corresponding $2^n \times 2^n$ *Möbius matrix* (see Definition 1).

[1] The URL to the repository will be linked in a future revision.

Definition 1 (Möbius Matrix). *The $2^n \times 2^n$ Möbius matrix M_{2^n} takes elements from $\mathbb{F}_2$ and is recursively defined as: $M_{2^0} = \begin{bmatrix}1\end{bmatrix}$, $M_{2^1} = \begin{bmatrix}1 & 1\\ 0 & 1\end{bmatrix}$, and $M_{2^n} = \begin{bmatrix}M_{2^{n-1}} & M_{2^{n-1}}\\ \mathbf{0} & M_{2^{n-1}}\end{bmatrix}$ for $n \geq 2$.*

Remark 1. The Möbius matrix is self-inverse.

Example 1. One of the most commonly used Möbius matrix that corresponds to a 4-bit SBox is given by:

$$M_{2^4} = \begin{pmatrix}
1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1\\
0&1&0&1&0&1&0&1&0&1&0&1&0&1&0&1\\
0&0&1&1&0&0&1&1&0&0&1&1&0&0&1&1\\
0&0&0&1&0&0&0&1&0&0&0&1&0&0&0&1\\
0&0&0&0&1&1&1&1&1&1&1&1&1&1&1&1\\
0&0&0&0&0&1&0&1&0&1&0&1&0&1&0&1\\
0&0&0&0&0&0&1&1&0&0&1&1&0&0&1&1\\
0&0&0&0&0&0&0&1&0&0&0&1&0&0&0&1\\
0&0&0&0&0&0&0&0&1&1&1&1&1&1&1&1\\
0&0&0&0&0&0&0&0&0&1&0&1&0&1&0&1\\
0&0&0&0&0&0&0&0&0&0&1&1&0&0&1&1\\
0&0&0&0&0&0&0&0&0&0&0&1&0&0&0&1\\
0&0&0&0&0&0&0&0&0&0&0&0&1&1&1&1\\
0&0&0&0&0&0&0&0&0&0&0&0&0&1&0&1\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&1\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1
\end{pmatrix}$$

□

Definition 2 (Coordinate Function). *Suppose $F : \mathbb{F}_2^n \to \mathbb{F}_2^n$ is defined as $F(x) = (f_0(x), \ldots, f_{n-1}(x))$ for all $x \in \mathbb{F}_2^n$, where $f_i : \mathbb{F}_2^n \to \mathbb{F}_2$ for $i = 0, \ldots, n-1$. Then each f_i is called a coordinate function of F.*

We also use the following two terms, *bit-slice matrix* and *coordinate matrix*. Informally, these two can be defined as follows. The bit-slice matrix of an $n \times n$ SBox is a $n \times 2^n$ matrix where column i stores the i^{th} look-up entry as a binary column vector. The coordinate matrix is the $n \times 2^n$ binary matrix which is the result of post-multiplying the Möbius matrix M_{2^n} to the bit-slice matrix.

The coordinate matrix plays an important role in representing the SBox in its coordinate functions (Definition 2) in Algebraic Normal Form (ANF). The rows indicate the coordinate functions, while the columns indicate which product of the input variables is present. This is shown in Example 2 with respect to the SBox used in `PRESENT` [9].

Example 2 (`PRESENT` SBox). The bit-slice matrix of the `PRESENT` SBox (`C56B90AD3EF84712`) is given as follows:

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

For instance, the last column of this matrix, $(0,1,0,0)^\top$, is the last look-up entry (2) written as the binary column vector. After post-multiplying the Möbius matrix with this matrix, one gets the following as the coordinate matrix of the PRESENT SBox:

$$C_{\texttt{PRESENT}} = B_{\texttt{PRESENT}} \times M_{2^4} = \begin{pmatrix} 0\,1\,0\,0\,1\,0\,1\,0\,1\,0\,0\,0\,0\,0\,0\,0 \\ 0\,0\,1\,0\,0\,0\,0\,1\,1\,0\,1\,1\,1\,1\,0\,0 \\ 1\,0\,0\,1\,1\,0\,0\,0\,1\,1\,1\,1\,0\,1\,0\,0 \\ 1\,1\,1\,0\,0\,0\,1\,1\,1\,0\,0\,1\,0\,1\,0\,0 \end{pmatrix}$$

Following the conventional notation, we denote the input variables as x_0, x_1, x_2, x_3 and the coordinate functions as y_0, y_1, y_2, y_3. Then the rows of the $C_{\texttt{PRESENT}}$ denote (in sequence) y_0, y_1, y_2, y_3, and the columns denote (in sequence) whether or not $(1, x_0, x_1, x_0x_1, x_2, x_0x_2, x_1x_2, x_0x_1x_2, x_3, x_0x_3, x_1x_3, x_0x_1x_3, x_2x_3, x_0x_2x_3, x_1x_2x_3, x_0x_1x_2x_3)$ is present. For instance, consider the top row, where 1 is present corresponding to columns $\{x_0, x_2, x_1x_2,\ x_3\}$, it means that: $y_0 = x_0 \oplus x_1x_2 \oplus x_2 \oplus x_3$.

In this way, all the coordinate functions of the PRESENT SBox can be computed:

$$\begin{aligned} y_1 &= x_0x_1x_2 \oplus x_0x_1x_3 \oplus x_0x_2x_3 \oplus x_1x_3 \oplus x_1 \oplus x_2x_3 \oplus x_3, \\ y_2 &= x_0x_1x_3 \oplus x_0x_1 \oplus x_0x_2x_3 \oplus x_0x_3 \oplus x_1x_3 \oplus x_2 \oplus x_3 \oplus 1, \\ y_3 &= x_0x_1x_2 \oplus x_0x_1x_3 \oplus x_0x_2x_3 \oplus x_0 \oplus x_1x_2 \oplus x_1 \oplus x_3 \oplus 1. \end{aligned}$$

Note that the algebraic degree of the three coordinate functions is 3 and some of the monomials are common (indeed reducing the number of monomials of algebraic degree 3 is the working factor in our work at Section 5).

□

Remark 2. The bit-slice matrix (and consequently the coordinate matrix too) of a bijective SBox is of full row-rank.

Remark 3. Sometimes the coordinate functions of an SBox (written in algebraic normal form) are called "SBox to ANF" in the literature.

Remark 4. For a bijective SBox, the last column of its coordinate matrix is null.

Remark 5. The maximum of the algebraic degrees of the coordinate functions of an SBox is termed as the algebraic degree of the SBox.

2.2 Affine Equivalence of SBoxes

We call SBoxes S_0 and S_1 *Affine Equivalent* (AE) if there exist two non-singular binary matrices A_0, A_1 (each with compatible dimension) such that $S_1 = A_0 \circ S_0 \circ A_1$; i.e., $S_1(x) = A_0S_0(A_1(x \oplus c_0)) \oplus c_1$, for all inputs x and for some binary vectors c_0, c_1 of compatible dimension.

To find affine equivalent SBoxes of a given $n \times n$ SBox, binary non-singular matrices are required. The General Linear group over $\mathbb{F}_q$ of degree n, denoted by $\mathrm{GL}(n, \mathbb{F}_q)$, consists of $n \times n$ non-singular matrices with each entry is from $\mathbb{F}_q$. It is known thanks to Euler that, the order of $\mathrm{GL}(n, \mathbb{F}_q)$ is, $\prod_{k=0}^{n-1}(q^n - q^k)$. Therefore, the number of 3×3 and 4×4 binary (so, $q = 2$) non-singular matrices are 168 and 20160, respectively.

2.3 Side Channel Attack and Countermeasure

Side channel attacks, particularly those relying on information from power consumption or electromagnetic emanation, are of prominent concern while dealing with the physical security of the ciphers [6, 20, 24, 31]. It has been systematically shown that a cipher with sufficient classical security claims falls short against an adversary equipped with a side channel attack set-up. It therefore goes without saying, understanding the attacks and finding low-cost countermeasures are among the top research priorities by/for the community.

Side-channel attacks are based on the connection between a (a priori or learned) model and any intermediate variable in the implementation that might be leaking. Therefore, the countermeasures attempt to destroy the linkage of the model and the intermediate variables. *Masking* [24, Chapter 9] is considered a prominent countermeasure. A masking scheme randomly distributes each intermediate to introduce randomness in a way that the overall algorithmic flow in the cipher is unchanged, but the randomised operations makes the side channel leakage free from the intermediate variables. Depending on the strength of the attacker, various *degrees* of masking can be adopted.

Threshold Implementation

The threshold implementation (TI) is a form of masking, and is among the top recommendations against the side channel attacks [6, 8, 18, 19, 25, 27, 28, 29] specially for protecting the hardware implementation. Aside from a protection against the hardware implementation, TI is also claimed to work with the software implementation of a cipher [16, 33].

Typically, the TI of an affine function is considered straightforward, while that of a non-linear (in most block ciphers, the only non-linear component is the SBox) function is considered a strenuous task to accomplish. The TI of a given SBox can be realised either through *without decomposition* (the SBox is implemented as a combinational circuit) or *with decomposition* (the SBox is implemented as a sequential circuit, typically it allows for smaller and more shallow netlists at the expense of pipelining) [18].

3 Threshold without Decomposition (Combinational SBox)

In essence, the without decomposition based threshold implementation takes the coordinate function (in ANF) form of an SBox, then converts into a suitable implementation satisfying a set of conditions. We consider the usual notation here, i.e., for an $n \times n$ SBox S we denote the input variables as $x_0, x_1, \ldots, x_{n-1}$ and the output variables as $y_0, y_1, \ldots, y_{n-1}$. Then it substitutes each x_i and each y_j by $d+1$ shares (where $d \geq$ the algebraic degree of the SBox). In the process, each the x_i and y_i variables are respectively replaced by the new variables $x_{i,j}$ and $y_{i,j}$; where $i \in \{0, 1, \ldots, n-1\}$ and $j \in \{0, 1, \ldots, d\}$. For simplicity, we write an SBox S in terms of its coordinate functions (in this case, $y_0, y_1, \ldots, y_{n-1}$) as:

$$S : \begin{cases} y_0 \\ y_1 \\ \vdots \\ y_{n-1} \end{cases}$$

Based on the existing literature [6, 12, 18, 19, 29], the 'without decomposition' based TI of S satisfies the following conditions:

(α) **Sharing of input variables.** $\bigoplus_{j=0}^{d} x_{i,j} = x_i$, for $i = 0, 1, \ldots, n-1$.
(β) **Sharing of output variables/Correctness.** $\bigoplus_{j=0}^{d} y_{i,j} = y_i$ for $i = 0, 1, \ldots, n-1$.
(γ) **Non-completeness.** $x_{i,j}$ is missing in $y_{k,j}$ for $j = 0, 1, \ldots, d$; for all $i, k \in \{0, 1, \ldots, n-1\}$ where $i \neq k$.
(δ) **Uniformity.** All non-zero entries in the 'x_i $(\forall i)$ versus $y_{i,j}$ $(\forall i, j)$ frequency distribution table' are equal.

After this, $d+1$ separate SBoxes $S_0, S_1, \ldots, S_d$ are implemented in parallel, where these SBoxes are given by the following arrangement of the coordinate functions:

$$
\begin{aligned}
S_0 : &\begin{cases} y_0 & = y_{0,0} \\ y_1 & = y_{1,0} \\ & \vdots \\ y_{n-1} & = y_{n-1,0} \end{cases} \\
S_1 : &\begin{cases} y_0 & = y_{0,1} \\ y_1 & = y_{1,1} \\ & \vdots \\ y_{n-1} & = y_{n-1,1} \end{cases} \\
&\vdots \\
S_d : &\begin{cases} y_0 & = y_{0,d} \\ y_1 & = y_{1,d} \\ & \vdots \\ y_{n-1} & = y_{n-1,d} \end{cases}
\end{aligned}
$$

Thanks to the ingenious arrangement, each of the SBoxes $(S_0, \ldots, S_d)$ computes some share of the original SBox (S). When combined, the coordinate functions of S (namely, $y_0, \ldots, y_{n-1}$) can be realised even if one $x_{i,j}$ variable in each $S_0, \ldots, S_d$ is randomised. Since one input variable $x_{i,j}$ is randomised, it makes the corresponding side channel leakage random, making it hard for the attacker to exploit secret information from the leakage (this outlines the philosophy of the side channel protection). At the beginning of the (protected) cipher execution, some input variables are fed random inputs and at the end all the shares are combined to get the intended cipher output. In other words the fundamental concept can be described as running multiple randomised modules (each is a bit different from the others) among which the inputs to the cipher are distributed. One module does not give exploitable information to the attacker, still the combined output from the modules results in the desired cipher output.

Remark 6 (Note on uniformity). As stated in [6, Chapter 3.3], the absence of the uniformity property (Condition (δ)) in a threshold circuit does not leak side channel information by

itself; but if it is used to drive another circuit, this second circuit may leak side channel information. However, we could not find any experimental evaluation in the literature, and this could be an interesting problem for future study.

3.1 Need for a Well-developed Algorithm

As noted (Section 1), the corresponding engineering methods to apply TI does not appear to be developed enough. No specific algorithm is available to the best of our understanding/finding, rather the concept is described mostly through some cherry-picked examples [6, 7, 18]. This is somewhat surprising, specially given the state-of-the-art explores advanced side channel attack (e.g., [21, 22]). The problems that arise from explaining a concept through examples instead of a developing a proper algorithm can be manifested in a number of ways, as listed next.

Problem 1 (Ambiguous sharing). Since the concept is mostly communicated through examples, it is in general hard to come up with a proper algorithm. To illustrate the point, we look into the example given in [18, Section III.A]. The 3-variable quadratic function $f(x, y, z) = xy \oplus z$ is shown with the following 3-shares:

$$\begin{aligned} f_1 &= z_2 \oplus x_2y_2 \oplus x_2y_3 \oplus x_3y_2, \\ f_2 &= z_3 \oplus x_1y_3 \oplus x_3y_1 \oplus x_3y_3, \\ f_3 &= z_1 \oplus x_1y_1 \oplus x_1y_2 \oplus x_2y_1. \end{aligned}$$

This sharing is not unique. For instance, the following is also a valid sharing of f_1 and f_2 (f_3 is unchanged):

$$\begin{aligned} f_1 &= \mathbf{z_3} \oplus x_2y_2 \oplus x_2y_3 \oplus x_3y_2, \\ f_2 &= \mathbf{z_2} \oplus x_1y_3 \oplus x_3y_1 \oplus x_3y_3. \end{aligned}$$

□

Problem 2 (Corner case: Low algebraic degree of a coordinate function). Consider the 3-bit SBox, `03214756`, which is given by the following coordinate functions:

$$\begin{aligned} y_0 &= x_0 \oplus x_1x_2, \\ y_1 &= x_0 \oplus x_1x_2 \oplus x_1, \\ y_2 &= x_2. \end{aligned}$$

Note that the coordinate function y_2 is of algebraic degree 1 whereas the rest are of algebraic degree 2 (i.e., not every coordinate function has the maximum algebraic degree). Thus, it is possible to consider 3-share masking for y_0 and y_1, but only 2-share is sufficient for y_2. Based on the existing literature this $(3, 3, 2)$-share is not considered valid. However, to the best of our understanding, this unequal sharing can thwart side channel leakage and will incur lower cost than the usual 3-sharing.

The same problem appears with some of the mainstream SBoxes, like `PRESENT` (given in Example 2), `PICCOLO` (`E4B238091A7F6C5D`) [34], `PYJAMASK-128` (`2D397BA6E0F4851C`) [17],

which are respectively given by the following coordinate functions (parenthesised numbers indicate the algebraic degree):

$$y_0 = x_0x_1x_2 \oplus x_0x_1 \oplus x_0x_2 \oplus x_0x_3 \oplus x_1x_2x_3 \oplus x_1x_3 \oplus x_1 \oplus x_2 \oplus x_3 \ (3),$$
$$y_1 = x_0x_1 \oplus x_0 \oplus x_1x_2x_3 \oplus x_1x_2 \oplus x_1x_3 \oplus x_2x_3 \oplus x_3 \oplus 1 \ (3),$$
$$y_2 = x_1x_2 \oplus x_1 \oplus x_2 \oplus x_3 \oplus 1 \ (2),$$
$$y_3 = x_0 \oplus x_2x_3 \oplus x_2 \oplus x_3 \oplus 1 \ (2);$$

and

$$y_0 = x_0x_1x_2 \oplus x_0x_1 \oplus x_0x_2 \oplus x_0x_3 \oplus x_0 \oplus x_1 \oplus x_2x_3 \oplus x_2 \ (3),$$
$$y_1 = x_0x_2 \oplus x_0 \oplus x_2x_3 \oplus 1 \ (2),$$
$$y_2 = x_0x_1x_2 \oplus x_0x_1 \oplus x_0 \oplus x_1x_2x_3 \oplus x_1x_2 \oplus x_2 \oplus x_3 \ (3),$$
$$y_3 = x_0 \oplus x_1x_2 \oplus x_3 \ (2).$$

□

Problem 3 (No automation). In a private communication, it is made known by the authors of [18] that the without decomposition based threshold implementation for the `GIFT` SBox (`1A4C6F392DB7508E`) [4] is computed manually. It is a laborious task, as it requires to compute the entire expression consisting of a few hundred to a few thousands (if not more) of monomials. In general, this task is tedious to carry out manually beyond 3-bit SBoxes and borderline impossible starting from 5-bit SBoxes.

□

Problem 4 (Lack of further optimisation). In certain cases, there could be opportunity to further optimise the threshold expressions. For example, one may look for factorisation so that the netlist can be further optimised (without compromising the side channel protection). Other logic gates than {AND, XOR} can also be used to reduce hardware cost.

□

3.2 Our Approach

We adopt a deterministic sharing approach, which we refer to as "unequal sharing". Here 'unequal' indicates that the assigned shares do not have (roughly) equal number of monomials. We would like to note that, there is prevailing theory that requires the number of monomials have to (roughly) equal. From we understand after going through the existing literature, having unequal number of monomials does not leak any secret information through side channel. This is based on a greedy algorithm (we try to assign as many monomials without running into a conflict to the initial shares) for simplicity and a basic overview of of our approach can be found in Algorithm 1.

It is possible to construct *equal sharing*, where all the shares of a coordinate function gets (roughly) equal number of monomials, although it is somewhat complex and hard to do in a deterministic way. However, equal sharing may have the advantage that the overall circuit depth/latency can be reduced.

Overall, the algorithm first generates the sharing of the input variables (Step 2), then substitutes it to the original coordinate functions (Step 4) to compute Y. After this, it iterates over all the monomials of Y to see if some shared input variable $x_{i,j}$ can be used in the expression of $y_{i,j}$ (Steps 9 to 17). Note that, the naïve AND count in the coordinate functions of the SBox directly affects the total number of monomials in the shares (more naïve AND count means more monomials in the coordinate functions of the SBox, this in turn leads to more monomials in the shares).

Since the underlying algorithm is greedy, i.e., an assignment is done as long as it does not cause any problem, the coordinate functions of the shared SBoxes with lexicographic priority get more monomials (which we call unbalanced sharing). In general, an algorithm which attempts to make balanced sharing gets quite complex (specially if the algorithm is deterministic) and it apparently does not provide any extra side channel security, hence this is kept as future work.

Because of the algorithm itself, the conditions required for TI are automatically satisfied (an additional sanity check is performed in Step 18), possibly except for uniformity (Condition (δ)). Hence, we provide a separate code that checks for uniformity, and if needed, some monomial rearrangement is done.

Algorithm 1: Unequal (deterministic) sharing

Input: An $n \times n$ SBox given as coordinate functions ($y_0, \ldots, y_{n-1}$ with input variables $x_0, \ldots, x_{n-1}$)
Output: Assignment of $d+1$ SBoxes given as coordinate functions

1: **for** $i \leftarrow 0$ to $n-1$ **do** ▷ Iterate over all input variables
2: $\quad X_i \leftarrow x_{i,0} \oplus x_{i,1} \oplus \cdots \oplus x_{i,d}$ ▷ Share input variables
3: **for** $i \leftarrow 0$ to $n-1$ **do** ▷ Iterate over all output variables
4: $\quad Y_i \leftarrow$ Substitute $x_j = X_j$ in y_i for all j ▷ Share output variables
5: **for** $i \leftarrow 0$ to $n-1$ **do**
6: $\quad Z_i \leftarrow [\underbrace{0, 0, \ldots, 0}_{d+1 \text{ times}}]$
7: $\quad$ **for** $s \leftarrow 0$ to d **do** ▷ Iterate over all shares
8: $\quad\quad z \leftarrow 0$
9: $\quad\quad$ **for** each monomial $m \in Y_i$ **do**
10: $\quad\quad\quad flag \leftarrow$ `False`
11: $\quad\quad\quad$ **for** each $u \in \{x_{0,s}, x_{1,s}, \ldots, x_{n-1,s}\}$ **do**
12: $\quad\quad\quad\quad$ **if** $u \in m$ **then**
13: $\quad\quad\quad\quad\quad flag \leftarrow$ `True`
14: $\quad\quad\quad$ **if** $flag =$ `False` **then**
15: $\quad\quad\quad\quad z \leftarrow z \oplus m$
16: $\quad\quad\quad\quad Y_i \leftarrow Y_i \oplus m$
17: $\quad\quad Z_i[s] \leftarrow Z_i[s] \oplus z$
18: $\quad$ **assert** $Y_i = 0$ ▷ Sanity check
19: $\quad$ **warn** if (at least) one of the coordinate functions Z_i is a constant
20: $\quad$ **return** sharing of y_i as Z_i

Warning for Zero-sharing Our algorithm (Step 19) throws a warning if some $y_{i,j}$ is a constant (happens if some coordinate function has less algebraic degree). As an illustration, reconsider the SBox described in Problem 2 (`03214756`). By exactly following Algorithm 1,

the sharing is given as (notice that a warning is given for $y_{2,2}$):

$$\begin{aligned} y_{2,0} &= x_{2,1} \oplus x_{2,2}, \\ y_{2,1} &= x_{2,0}, \\ y_{2,2} &= \mathbf{0}. \end{aligned}$$

Since this violates the uniformity condition (Condition (δ)), this can be resolved, e.g., by changing to the following sharing:

$$\begin{aligned} y_{2,0} &= x_{2,1}, \\ y_{2,1} &= x_{2,0}, \\ y_{2,2} &= x_{2,2}. \end{aligned}$$

The resolution to zero-sharing is not enforced (to have future compatibility with unequal number of shares, as outlined in Problem 2). Rather, our source code contains an optional argument which can be set should the user wish resolution.

3.3 Results

Minimal Order (Algebraic Degree + 1) Sharing Our approach can deal with 3×3 and 4×4 SBoxes without any difficulty. Further, 5×5 SBoxes, such as that of `ASCON`'s [14] works well (one may note from [30] that `ASCON` is recently selected as the primary choice in the LWC project run by NIST). Results with some SBoxes are summarised in Table 1, where we show the number of monomials along with STM 130nm ASIC cost in terms of gate equivalent (rounded off to nearest integer). Each of the SBoxes is used in a cipher, save for `048AFC691EBD7532` which is presented in [15, Section 3.4]. The number of shares are taken as the algebraic degree of the SBox plus 1.

Higher Order ($\geq$ Algebraic Degree + 2) Sharing Our tool can work with higher number of shares as well, without any actual change in the algorithm flow. We show some examples in Table 2, where the number of shares is taken as the algebraic degree of the SBox plus 2. It is worth noting that `ASCON` has recently been selected as the NIST LWC winner [30].

Detailed Example with `DEFAULT` LS SBox (Minimal Order) We take the `DEFAULT` LS SBox (`037ED4A9CF18B265`) [2,3] and show its miniaml order sharing as obtained by running through Algorithm 1. The coordinate functions are given by:

$$\begin{aligned} y_0 &= x_0 \oplus x_1 \oplus x_2, \\ y_1 &= x_0x_1 \oplus x_0x_2 \oplus x_0 \oplus x_1x_3 \oplus x_1 \oplus x_2x_3, \\ y_2 &= x_1 \oplus x_2 \oplus x_3, \\ y_3 &= x_0x_1 \oplus x_0x_2 \oplus x_1x_3 \oplus x_2x_3 \oplus x_2 \oplus x_3. \end{aligned}$$

Table 1: Threshold (without decomposition) cost of some SBoxes

		Shares	# Monomials	Hardware☆
GIFT [4]	1A4C6F392DB7508E	4	265★	526
PRESENT [9]	C56B90AD3EF84712	4	666	1132
PRINCE [10]	BF32AC916780E5D4	4	731	991
PICCOLO [34]	E4B238091A7F6C5D	4	399	645
SKINNY-64 [5]	C6901A2B385D4E7F	4	398	723
TWINE [35]	C0FA2B9583D71E64	4	626	723
PYJAMASK-128 [17]	2D397BA6E0F4851C	4	373	640
QARMA [1]	ADE6F735980CB124	4	562	826
NOEKEON Gamma [13]	7A2C48F0591E3DB6	4	387	697
DEFAULT [2,3] LS	037ED4A9CF18B265	3	102	60
DEFAULT [2,3] Non-LS	196F7C82AED043B5	4	213	412
Gao-Roy-Oswald [15]	048AFC691EBD7532	4	988	1658
ASCON★ [14]		3	160	85

☆: Gate equivalent in STM 130nm ASIC library (HCMOS9GP)
★: Used in [18, Appendix B]
★: 4B1F141A15921B58121D361C1E137E0D111810C11916AF17

Table 2: Higher order threshold (without decomposition) cost of some SBoxes

		Shares	# Monomials	Hardware☆
GIFT [4]	1A4C6F392DB7508E	5	451	432
SKINNY-64 [5]	C6901A2B385D4E7F	5	682	681
QARMA [1]	ADE6F735980CB124	5	967	983
ASCON★ [14]		4	257	142

☆: Gate equivalent in STM 130nm ASIC library (HCMOS9GP)
★: 4B1F141A15921B58121D361C1E137E0D111810C11916AF17

Since the algebraic degree is 2, minimum 3 shares are needed (based on the state-of-the-art literature). A 3-share implementation following Algorithm 1 (with the coordinate functions as 0 are resolved by taking one monomial from its previous coordinate function) is given as:

$$
\begin{aligned}
y_{0,0} &= x_{0,1} \oplus x_{0,2} \oplus x_{1,1} \oplus x_{1,2} \oplus x_{2,1} \oplus x_{2,2},\\
y_{0,1} &= x_{0,0} \oplus x_{1,0},\\
y_{0,2} &= x_{2,0},\\
y_{1,0} &= x_{0,1}x_{1,1} \oplus x_{0,1}x_{1,2} \oplus x_{0,1}x_{2,1} \oplus x_{0,1}x_{2,2} \oplus x_{0,1} \oplus x_{0,2}x_{1,1} \oplus x_{0,2}x_{1,2} \oplus x_{0,2}x_{2,1}\\
&\quad \oplus x_{0,2}x_{2,2} \oplus x_{0,2} \oplus x_{1,1}x_{3,0} \oplus x_{1,1}x_{3,1} \oplus x_{1,1}x_{3,2} \oplus x_{1,1} \oplus x_{1,2}x_{3,0} \oplus x_{1,2}x_{3,1}\\
&\quad \oplus x_{1,2}x_{3,2} \oplus x_{1,2} \oplus x_{2,1}x_{3,0} \oplus x_{2,1}x_{3,1} \oplus x_{2,1}x_{3,2} \oplus x_{2,2}x_{3,0} \oplus x_{2,2}x_{3,1} \oplus x_{2,2}x_{3,2},\\
y_{1,1} &= x_{0,0}x_{1,0} \oplus x_{0,0}x_{1,2} \oplus x_{0,0}x_{2,0} \oplus x_{0,0}x_{2,2} \oplus x_{0,0} \oplus x_{0,2}x_{1,0} \oplus x_{0,2}x_{2,0} \oplus x_{1,0}x_{3,0}\\
&\quad \oplus x_{1,0}x_{3,1} \oplus x_{1,0}x_{3,2} \oplus x_{1,0} \oplus x_{2,0}x_{3,0} \oplus x_{2,0}x_{3,1} \oplus x_{2,0}x_{3,2},\\
y_{1,2} &= x_{0,0}x_{1,1} \oplus x_{0,0}x_{2,1} \oplus x_{0,1}x_{1,0} \oplus x_{0,1}x_{2,0},\\
y_{2,0} &= x_{1,1} \oplus x_{1,2} \oplus x_{2,1} \oplus x_{2,2} \oplus x_{3,0} \oplus x_{3,1} \oplus x_{3,2},\\
y_{2,1} &= x_{1,0},\\
y_{2,2} &= x_{2,0},
\end{aligned}
$$

$$
\begin{aligned}
y_{3,0} &= x_{0,1}x_{1,1} \oplus x_{0,1}x_{1,2} \oplus x_{0,1}x_{2,1} \oplus x_{0,1}x_{2,2} \oplus x_{0,2}x_{1,1} \oplus x_{0,2}x_{1,2} \oplus x_{0,2}x_{2,1} \\
&\quad \oplus x_{0,2}x_{2,2} \oplus x_{1,1}x_{3,0} \oplus x_{1,1}x_{3,1} \oplus x_{1,1}x_{3,2} \oplus x_{1,2}x_{3,0} \oplus x_{1,2}x_{3,1} \oplus x_{1,2}x_{3,2} \\
&\quad \oplus x_{2,1}x_{3,0} \oplus x_{2,1}x_{3,1} \oplus x_{2,1}x_{3,2} \oplus x_{2,1} \oplus x_{2,2}x_{3,0} \oplus x_{2,2}x_{3,1} \oplus x_{2,2}x_{3,2} \oplus x_{2,2} \\
&\quad \oplus x_{3,0} \oplus x_{3,1} \oplus x_{3,2}, \\
y_{3,1} &= x_{0,0}x_{1,0} \oplus x_{0,0}x_{1,2} \oplus x_{0,0}x_{2,0} \oplus x_{0,0}x_{2,2} \oplus x_{0,2}x_{1,0} \oplus x_{0,2}x_{2,0} \oplus x_{1,0}x_{3,0} \oplus \\
&\quad x_{1,0}x_{3,1} \oplus x_{1,0}x_{3,2} \oplus x_{2,0}x_{3,0} \oplus x_{2,0}x_{3,1} \oplus x_{2,0}x_{3,2} \oplus x_{2,0}, \\
y_{3,2} &= x_{0,0}x_{1,1} \oplus x_{0,0}x_{2,1} \oplus x_{0,1}x_{1,0} \oplus x_{0,1}x_{2,0}.
\end{aligned}
$$

4 Threshold with Decomposition (Sequential SBox)

The theory for a decomposition based TI requires finding two other SBoxes such that the composition of these SBoxes is the target SBox. Given an $n \times n$ SBox S, we want to find two $n \times n$ SBoxes F and G such that

(i) $F \circ G \equiv S$ (i.e., $F(G(x)) = S(x)\ \forall x$);
(ii) the algebraic degrees of both F and G are less than or equal to the algebraic degree of S (preferably the algebraic degrees of both F and G are less than the algebraic degree of S).

For instance, consider [18, Section III.A]; where given $S =$ `1A4C6F392DB7508E` (cubic), the authors find $F =$ `4DF71A285CE60B39` (quadratic) and $G =$ `5638127C9EF0DAB4` (quadratic).

Despite being quite popular [18, 26, 32], it appears that there exists only one (publicly available) tool which is a courtesy of Petkova-Nikova[2]. This tool, available exclusively as executable files, can find decomposition based threshold for a given 3×3 or 4×4 SBox.

Here we describe a simple idea to find such decomposition in Algorithm 2. In short, we generate F and G in a way that $F \circ G \equiv S$ is always satisfied (by randomly constructing F, then adjusting G according to F and S); then we decide whether to keep or discard depending on the algebraic degree requirement. The algorithm is inherently randomised, so we continue from the beginning until the algebraic degree requirement is satisfied.

Algorithm 2: Decomposition (of a given SBox into two SBoxes)

Input: An $n \times n$ SBox S
Output: Two $n \times n$ SBoxes F and G such that $F \circ G \equiv S$

1: $R \leftarrow \emptyset$ ▷ Initialise R as empty set
2: **for** $i \leftarrow 0$ to $2^n - 1$ **do** ▷ Iterate over all SBox inputs
3: $\quad r \xleftarrow{\$} \{0, 1, \ldots, 2^n - 1\} \setminus R$ ▷ Select r uniformly without repeat
4: $\quad F(i) \leftarrow r$, $G(r) \leftarrow S(i)$ ▷ $F : i \mapsto r$, $G : r \mapsto S(i)$
5: $\quad$ Insert r to R ▷ $R \leftarrow R \cup \{r\}$
6: **if** algebraic degree of F or $G >$ that of S **then**
7: $\quad$ Discard F and G, and go back to Step 1
8: **return** F, G

[2] Hosted at Svetla Petkova-Nikova's official web-page: https://homes.esat.kuleuven.be/~snikova/ti_tools.html.

Overall, Algorithm 2 hugely simplifies the problem (cf. the complication in [32, Section 3.2] or [18, Section III.A]), and seems to work well for larger ($> 4 \times 4$) SBoxes in practice. If needed, our idea can be extended to more than two decomposed SBoxes.

5 Further Optimisation Based on Affine Equivalence

In this part, we attempt to reduce the cost of an existing TI by optimising its SBox. In essence, we redesign the cipher with an affine equivalent SBox (so that the cipher specification is unchanged) which reduces the TI cost. The SBox is implemented as combinational only circuit (i.e., no register) per coordinate functions. We show our method on the lightweight block cipher `PRESENT` [9], though it can be applied to any other cipher with similar construction.

We study in this section the impact of affine equivalence on the SBox to reduce the cost of TI. Notice that the concept of affine equivalence we propose here is with respect to the representation of the cipher, not about changing the specification of the cipher. In other words, only the implementation changes, but the cipher description remains unchanged.

5.1 Motivation and Basic Observation

The complexity of threshold implementations directly relates to the number of monomials in the coordinate function of the SBox. Let us consider the `PRESENT` [9] (which is also a standard, ISO/IEC 29192-2:2012[3]) SBox, `C56B90AD3EF84712`. From its coordinate functions given in Example 2, we notice the following properties:

- 8 monomials of degree 3,
- 7 monomials of degree 2,
- 10 monomials of degree 1, and
- 2 monomials of degree 0 (constant 1).

Upon a closer inspection, however, we observe some monomials are duplicates. The following factors contribute to the total cost:

1. **Computation:** Number of unique monomials of given degree (i.e., each monomial of a given degree is counted only once even if its multiplicity is higher).
2. **Reduction:** Number of XORs in the coordinate functions (represented in ANF).

About the `PRESENT` SBox, the contributing components are:

- 3 unique monomials of degree 3 (namely, $x_0x_1x_2$, $x_0x_1x_3$, and $x_0x_2x_3$),
- 5 unique monomials of degree 2, (the only absent one being x_0x_2),
- 4 monomials of degree 1, and 1 monomial of degree 0 (constant 1),
- 23 XORs.

Since the cost for TI for a higher degree monomial is much higher than a lower degree monomial, our aim here is to minimise the number of unique monomials of degree 3, then of degree 2, etc.

[3] https://webstore.ansi.org/Standards/ISO/ISOIEC291922012-1383736.

5.2 Improving Efficiency with Affine Equivalent SBox

From an implementation cost standpoint, it can be beneficial to represent the PRESENT block cipher in an equivalent notation where the SBox S is traded for one of its affine equivalent SBox, $S' = A^{-1} \circ S \circ A$, where A is an invertible affine mapping operating in $\mathbb{F}_2^4$. We reiterate that we do not aim at altering the cipher functionality, simply its representation. The usage of the affine mapping is compensated before and after the SBox application.

While detailed description of the cipher is skipped here for space constraint, an overview is given in Figure 1 for a quick reference. The use of the affine equivalent SBox is illustrated

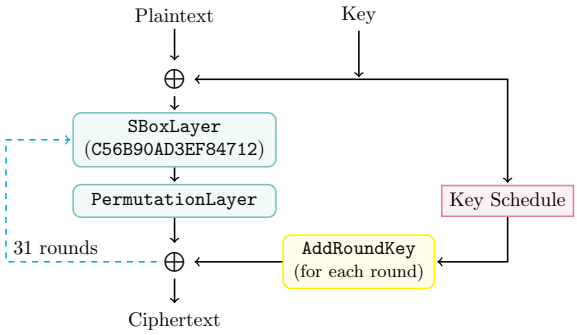
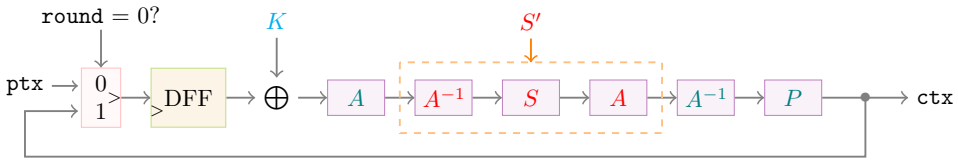


Fig. 1: PRESENT encryption (schematic)

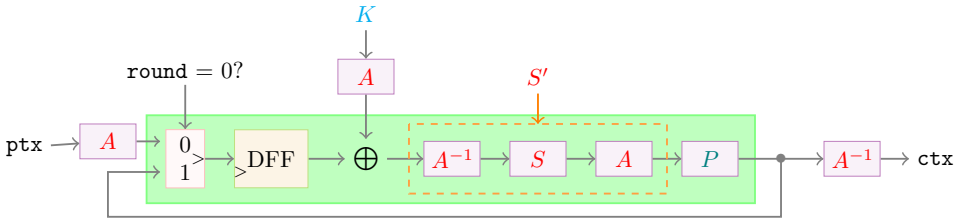
in Figure 2. In the part (a) of the figure, a simplified block diagram of PRESENT is depicted, with the main components:

- Ports are plaintext (denoted as, "ptx"), round key (denoted as "K"), a selection signal indicating whether the encryption starts (denoted as, "round $= 0$?") or not, and an output ciphertext denoted as "ctx").
- Building blocks are the affine equivalent SBox S, the permutation layer P, a multiplexer allowing to input the plaintext or to iterate, and a DFF barrier storing the result computed till that particular round.

This can be implemented with a shorter critical path by pushing the conversion to/from affine representation outside of the main path. This is represented in the Figure 2(b): Assuming that the plaintext and the round keys are applied the affine transformation, then a regular datapath can be used, provided finally the ciphertext is applied to inverse affine mapping. This equivalent representation leverages the fact that all elements in the PRESENT cipher (permutation layer and multiplexer) are linear. Notice that the scheme in Figure 2(b) is correct only if A is linear. If it is affine, then the constant of the transformation shall be adapted for each operation, in particular the outer ones. The critical path, highlighted as the green box in Figure 2(b), is no different than the original critical path. The architecture shown in Figure 2 is thus suitable for masked TI with combinational SBox (i.e., without decomposition).



(a) PRESENT leveraging $S' = A^{-1} \circ S \circ A$



(b) Equivalent PRESENT leveraging $S' = A^{-1} \circ S \circ A$

Fig. 2: PRESENT leveraging affine equivalent SBox

5.3 Results

Efficiency Based on Algebraic Property For 1000 random choices of A, we count the number of monomials in each degree, and we get statistics as depicted in Figure 3. More specifically, Figure 3(a) shows the relative frequency distribution for number of unique monomials for each individual degrees. For instance, there exists a unique monomial for degree 0 (constant 1) 100% of the cases. Figure 3(b) shows the probability distribution of XOR count.

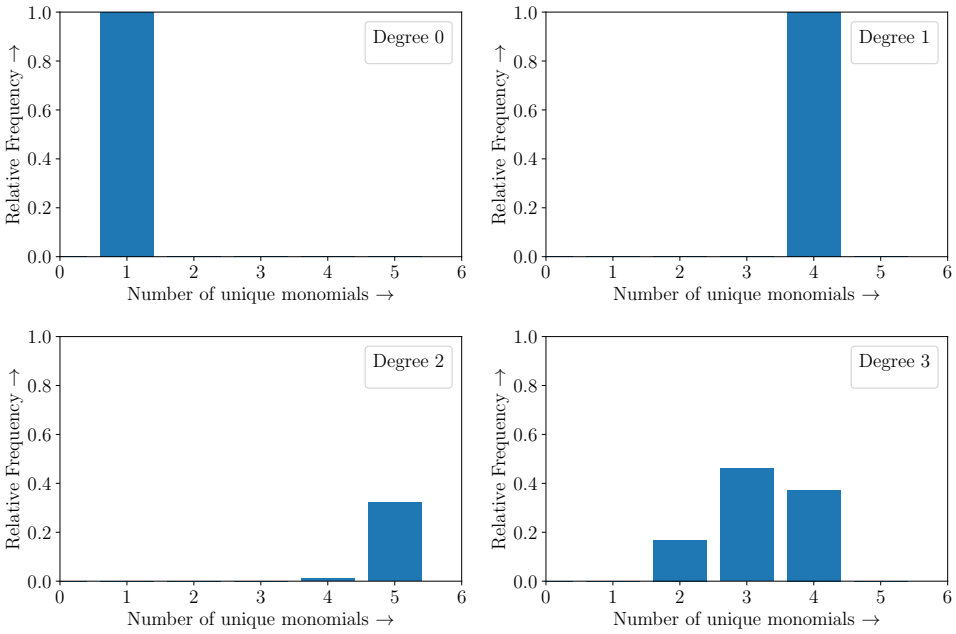
It can be seen that it is possible to reduce the number of monomials of degree 3 to only 2. We observe that there is no preferred choice for the constant in the affine transformation.

In our case, the transformed SBox is found as follows. The binary matrix multiplied to obtain the linear part, and the constant binary vector are given respectively by: $\begin{pmatrix} 1&0&0&0 \\ 1&1&1&0 \\ 1&1&0&0 \\ 1&0&1&1 \end{pmatrix}$, $\begin{pmatrix} 0 \\ 1 \\ 1 \\ 1 \end{pmatrix}$. The transformed SBox, `4EC20B1A5F3D9867`, has the coordinate functions:

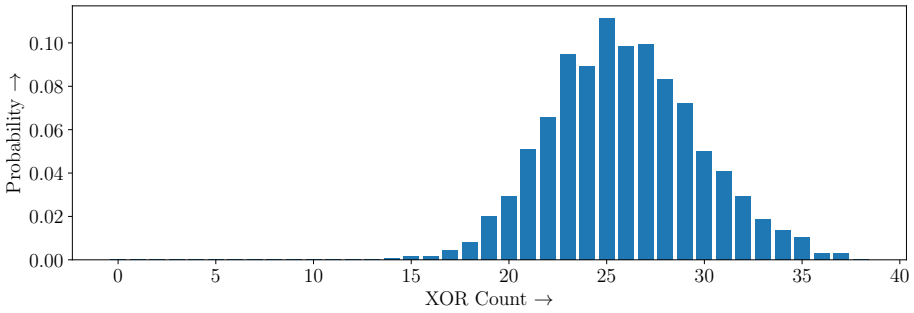
$$
\begin{aligned}
y_0 &= x_0x_2 \oplus x_1x_2 \oplus x_3, \\
y_1 &= x_0x_2x_3 \oplus x_0 \oplus x_1x_3, \\
y_2 &= x_0x_1x_2 \oplus x_0x_1 \oplus x_1x_3 \oplus x_2 \oplus 1, \\
y_3 &= x_0x_2x_3 \oplus x_0 \oplus x_1x_2 \oplus x_1x_3 \oplus x_1 \oplus x_2x_3.
\end{aligned}
$$

Therefore, we manage to get a transformed SBox with the following properties in its coordinate functions:

- 2 unique monomials of degree 3 ($x_0x_2x_3$ and $x_0x_1x_2$),



(a) Monomials



(b) XOR

Fig. 3: Statistics for affine equivalent SBox search for `PRESENT`

- 5 unique monomials of degree 2 ($x_0x_2, x_1x_2, x_1x_3, x_0x_1$ and x_2x_3),
- 4 monomials of degree 1, and 1 monomial of degree 0 (constant 1),
- only 13 XORs.

Statistics on Netlist Parameters Now we study the real impact of affine transformation to the property of the netlist; namely, we are interested in the algebraic features that drive the netlist properties. We study the netlist area (number of monomials vs. GE) and its logical depth (number of monomials vs. critical path). For the sake of simplicity, every gate is attributed a unitary area and propagation time.

The goal is to figure out which property of the algebraic expression determines most the area and/or depth in the netlist. The following features are considered:

- The number of unique monomials of various degrees (3, 2, 1, and 0) – mostly, the number of unique monomials of degree 3 is to be minimised.
- XOR count is to be reduced to make area usage low.

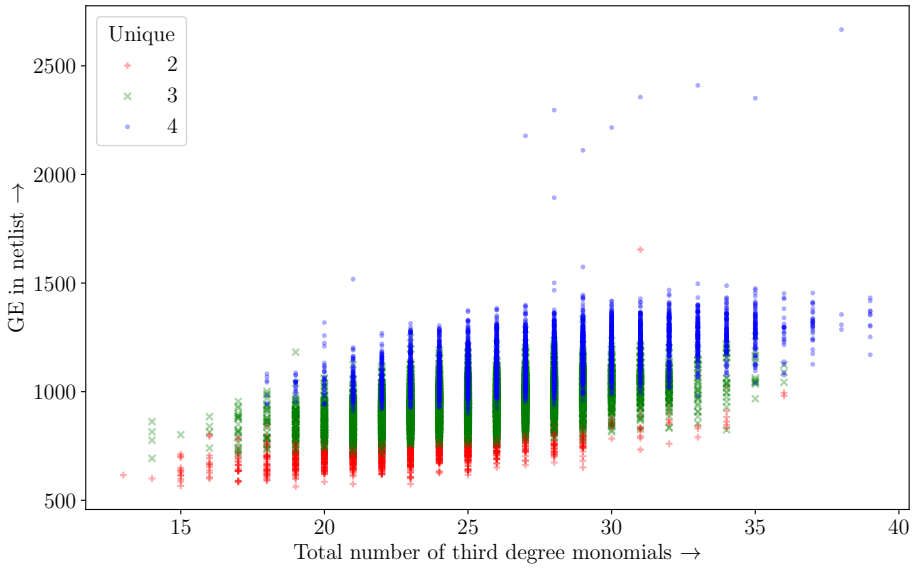


Fig. 4: Area (GE) vs. third degree monomials (SBoxes AE to `PRESENT` SBox)

The results are summarised in Figures 4, 5 and 6; based on 10000 random selections of affine transformation $A : \mathbb{F}_2^4 \to \mathbb{F}_2^4$; where we group based on number of unique monomials for better readability. It can be seen that, as expected, reducing the unique number of monomials of degree 3 is the main parameter to reduce the gate count and the depth of the netlists. The smallest netlist has 589 gates (of depth 18), and the most shallow one has depth 11 (and 728 gates). For comparison, the reference netlist (i.e., when A is the identity matrix) has 863 gates and a depth of 23. Hence an area reduction of more than 31% or a depth reduction of more than 52% is observed by applying our methodology. Further, as it can be seen from Figure 6, larger netlists also have (slightly) longer critical path, on average.

6 Conclusion

This work takes a deeper looks into the problem of finding threshold implementation of SBoxes. The first main contribution of this work is to present an open-source tool for automating the task for threshold implementation for a large pool of SBoxes. Our tool returns 'without decomposition' (Section 3) and 'with decomposition' (Section 4) based implementations. Despite being quite popular, such a tool seems overdue. The second main contribution (Section 5) comes from an alternate representation of the `PRESENT` SBox [9] so that the TI cost can be reduced. The idea is to replace the original SBox by one of its affine equivalent SBox (so that the cipher description remains unchanged), but this new SBox

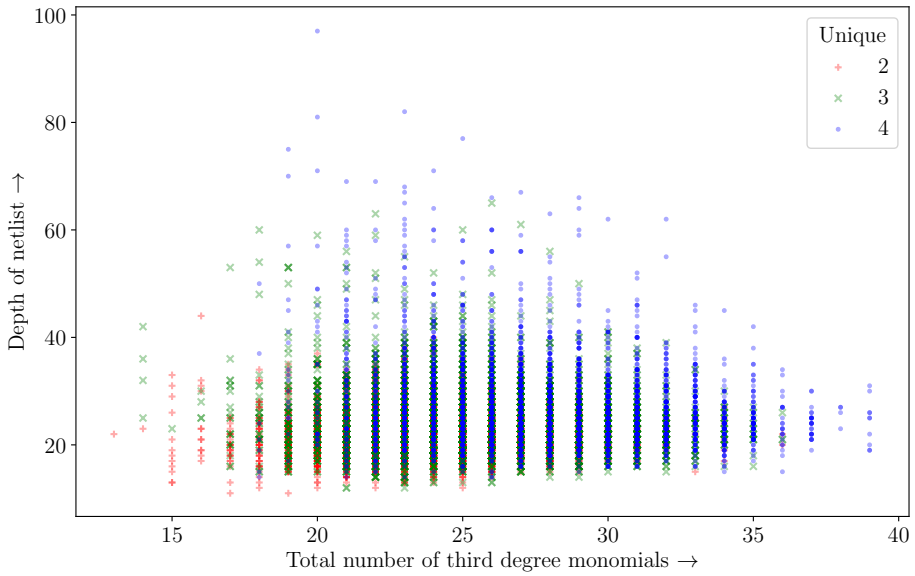


Fig. 5: Logical depth vs. third degree monomials (SBoxes AE to `PRESENT` SBox)

has lower threshold cost. Overall, we show over 31% improved area and over 52% improved depth compared to the naïve implementation.

One interesting follow-up work could be to find SBoxes with lower AND count (but with other desirable cryptographic properties) so that the cipher is more suitable for adopting TI. Besides, as noted in Remark 6, it would be interesting to evaluate the amount of side channel leakage from the circuit which takes input from another circuit not obeying the uniformity property. As the main objective in Section 5 is to find another SBox, works like [23] can be incorporated in the search procedure in the future.

References

1. Avanzi, R.: The qarma block cipher family – almost mds matrices over rings with zero divisors, nearly symmetric even-mansour constructions with non-involutory central rounds, and search heuristics for low-latency s-boxes. Cryptology ePrint Archive, Report 2016/444 (2016) https://eprint.iacr.org/2016/444. 11
2. Baksi, A.: Classical and Physical Security of Symmetric Key Cryptographic Algorithms. PhD thesis, School of Computer Science & Engineering, Nanyang Technological University, Singapore (2021) https://dr.ntu.edu.sg/handle/10356/152003. 10, 11
3. Baksi, A.: DEFAULT: Cipher-Level Resistance Against Differential Fault Attack. Springer, Singapore (2022) https://link.springer.com/chapter/10.1007/978-981-16-6522-6_8. 10, 11
4. Banik, S., Pandey, S.K., Peyrin, T., Sasaki, Y., Sim, S.M., Todo, Y.: Gift: A small present. Cryptology ePrint Archive, Report 2017/622 (2017) https://eprint.iacr.org/2017/622. 8, 11
5. Beierle, C., Jean, J., Kölbl, S., Leander, G., Moradi, A., Peyrin, T., Sasaki, Y., Sasdrich, P., Sim, S.M.: The SKINNY family of block ciphers and its low-latency variant MANTIS. IACR Cryptology ePrint Archive **2016** (2016) 660 11
6. Bilgin, B.: Threshold Implementations As Countermeasure Against Higher-Order Differential Power Analysis. PhD thesis, Katholieke Universiteit Leuven and University of Twente (2015) https://www.esat.kuleuven.be/cosic/publications/thesis-256.pdf. 1, 2, 5, 6, 7

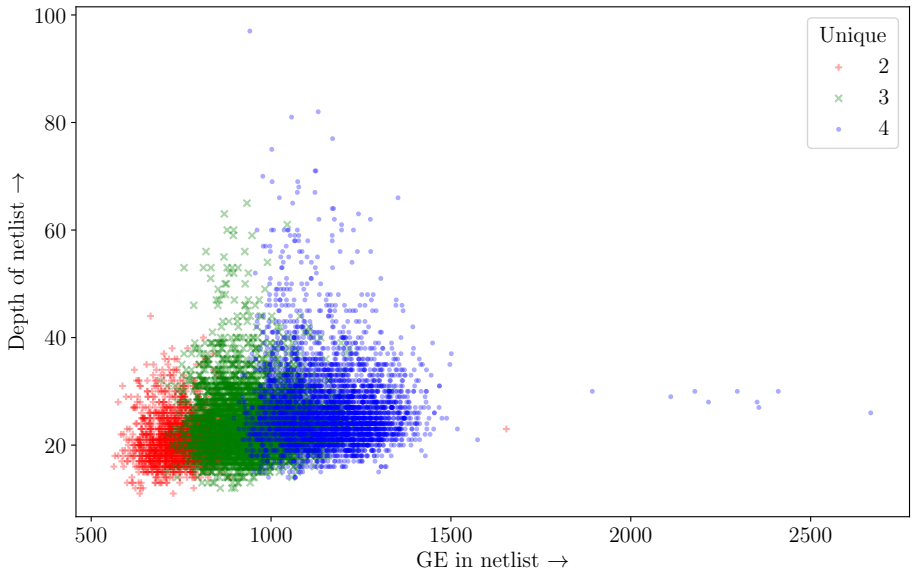


Fig. 6: Area (GE) vs. logical depth for third degree monomials (SBoxes AE to `PRESENT` SBox)

7. Bilgin, B., Nikova, S., Nikov, V., Rijmen, V., Stütz, G.: Threshold implementations of all 3x3 and 4x4 s-boxes. IACR Cryptol. ePrint Arch. (2012) 300 http://eprint.iacr.org/2012/300. 1, 7
8. Bilgin, B., Nikova, S., Nikov, V., Rijmen, V., Tokareva, N., Vitkup, V.: Threshold Implementations of Small S-boxes. Cryptography and Communications **7**(1) (2015) 3–33 5
9. Bogdanov, A., Knudsen, L.R., Leander, G., Paar, C., Poschmann, A., Robshaw, M.J., Seurin, Y., Vikkelsoe, C.: PRESENT: An ultra-lightweight block cipher. In: CHES. Volume 4727., Springer (2007) 450–466 2, 3, 11, 13, 17
10. Borghoff, J., Canteaut, A., Güneysu, T., Kavun, E.B., Knežević, M., Knudsen, L.R., Leander, G., Nikov, V., Paar, C., Rechberger, C., Rombouts, P., Thomsen, S.S., Yalçin, T.: Prince - a low-latency block cipher for pervasive computing applications (full version). Cryptology ePrint Archive, Report 2012/529 (2012) https://ia.cr/2012/529. 11
11. Caforio, A., Collins, D., Glamocanin, O., Banik, S.: Improving first-order threshold implementations of skinny. Cryptology ePrint Archive, Report 2021/1425 (2021) https://ia.cr/2021/1425. 1
12. Daemen, J.: Changing of the guards: a simple and efficient method for achieving uniformity in threshold sharing. Cryptology ePrint Archive, Report 2016/1061 (2016) https://ia.cr/2016/1061. 6
13. Daemen, J., Peeters, M., Assche, G.V., Rijmen, V.: Nessie Proposal: NOEKEON (2000) http://gro.noekeon.org/Noekeon-spec.pdf. 11
14. Dobraunig, C., Eichlseder, M., Mendel, F., Schläffer, M.: Ascon v1.2. Submission to NIST (2019) https://csrc.nist.gov/CSRC/media/Projects/lightweight-cryptography/documents/round-2/spec-doc-rnd2/ascon-spec-round2.pdf. 10, 11
15. Gao, S., Roy, A., Oswald, E.: Constructing ti-friendly substitution boxes using shift-invariant permutations. In Matsui, M., ed.: Topics in Cryptology – CT-RSA 2019, Cham, Springer International Publishing (2019) 433–452 10, 11
16. Gaspoz, J., Dhooghe, S.: Threshold implementations in software: Micro-architectural leakages in algorithms. Cryptology ePrint Archive, Paper 2022/1546 (2022) https://eprint.iacr.org/2022/1546. 5
17. Goudarzi, D., Jean, J., Kölbl, S., Peyrin, T., Rivain, M., Sasaki, Y., Sim, S.M.: Pyjamask v1.0. (2019) 7, 11
18. Jati, A., Gupta, N., Chattopadhyay, A., Sanadhya, S.K., Chang, D.: Threshold implementations of gift: A trade-off analysis. IEEE Trans. Inf. Forensics Secur. **15** (2020) 2110–2120 1, 2, 5, 6, 7, 8, 11, 12, 13

19. Kutzner, S., Nguyen, P.H., Poschmann, A.: Enabling 3-share threshold implementations for any 4-bit s-box. Cryptology ePrint Archive, Report 2012/510 (2012) https://eprint.iacr.org/2012/510. 1, 5, 6
20. Lomné, V.: Power and Electro-Magnetic Side-Channel Attacks: Threats and Countermeasures. PhD thesis, Docteur de l'Université Montpellier II (2010) https://sites.google.com/site/victorlomne/research. 5
21. Lomné, V., Prouff, E., Rivain, M., Roche, T., Thillard, A.: How to Estimate the Success Rate of Higher-Order Side-Channel Attacks. In Batina, L., Robshaw, M., eds.: Cryptographic Hardware and Embedded Systems - CHES 2014 - 16th International Workshop, Busan, South Korea, September 23-26, 2014. Proceedings. Volume 8731 of Lecture Notes in Computer Science., Springer (2014) 35–54 7
22. Lomné, V., Prouff, E., Roche, T.: Behind the scene of side channel attacks. Cryptology ePrint Archive, Report 2013/794 (2013) https://eprint.iacr.org/2013/794. 7
23. Lu, Z., Mesnager, S., Cui, T., Fan, Y., Wang, M.: An stp-based model toward designing s-boxes with good cryptographic properties. Des. Codes Cryptogr. **90**(5) (2022) 1179–1202 18
24. Mangard, S., Oswald, E., Popp, T.: Power analysis attacks - Revealing the secrets of smart cards. Springer (2007) 5
25. Moradi, A., Schneider, T.: Side-channel analysis protection and low-latency in action - case study of prince and midori. Cryptology ePrint Archive, Paper 2016/481 (2016) https://eprint.iacr.org/2016/481. 5
26. Müller, N., Moos, T., Moradi, A.: Low-latency hardware masking of PRINCE. In Bhasin, S., Santis, F.D., eds.: Constructive Side-Channel Analysis and Secure Design - 12th International Workshop, COSADE 2021, Lugano, Switzerland, October 25-27, 2021, Proceedings. Volume 12910 of Lecture Notes in Computer Science., Springer (2021) 148–167 12
27. Nikova, S., Nikov, V., Rijmen, V.: Decomposition of permutations in a finite field. Cryptogr. Commun. **11**(3) (2019) 379–384 5
28. Nikova, S., Rechberger, C., Rijmen, V.: Threshold implementations against side-channel attacks and glitches. In: International conference on information and communications security, Springer (2006) 529–545 5
29. Nikova, S., Rijmen, V., Schläffer, M.: Secure hardware implementation of nonlinear functions in the presence of glitches. J. Cryptol. **24**(2) (2011) 292–321 5, 6
30. NIST: Lightweight Cryptography Standardization Process: NIST Selects Ascon (February 7 2023) https://csrc.nist.gov/News/2023/lightweight-cryptography-nist-selects-ascon. 10
31. Peeters, E.: Advanced DPA Theory and Practice: Towards the Security Limits of Secure Embedded Circuits. 1 edn. Springer-Verlag New York (2013) 5
32. Poschmann, A., Moradi, A., Khoo, K., Lim, C., Wang, H., Ling, S.: Side-Channel Resistant Crypto for Less than 2, 300 GE. J. Cryptology **24**(2) (2011) 322–345 12, 13
33. Sasdrich, P., Bock, R., Moradi, A.: Threshold implementation in software - case study of PRESENT. In Fan, J., Gierlichs, B., eds.: Constructive Side-Channel Analysis and Secure Design - 9th International Workshop, COSADE 2018, Singapore, April 23-24, 2018, Proceedings. Volume 10815 of Lecture Notes in Computer Science., Springer (2018) 227–244 5
34. Shibutani, K., Isobe, T., Hiwatari, H., Mitsuda, A., Akishita, T., Shirai, T.: Piccolo: An Ultra-Lightweight Blockcipher. In: Cryptographic Hardware and Embedded Systems - CHES 2011 - 13th International Workshop, Nara, Japan, September 28 - October 1, 2011. Proceedings. (2011) 342–357 7, 11
35. Suzaki, T., Minematsu, K., Morioka, S., Kobayashi, E.: Twine: A lightweight, versatile block cipher. (2011) https://www.nec.com/en/global/rd/tg/code/symenc/pdf/twine_LC11.pdf. 11