Overshooting the Threshold: $(td + n)$-Masking

Vincent Grosso[1] and Carlos Andres Lara-Nino[2]

[1] CNRS/Laboratoire Hubert Curien, Saint-Étienne, France
[2] Universitat Rovira i Virgili, Tarragona, Spain

Abstract. Masking is one of the most widespread countermeasures against side-channel attacks. However, its integration into hardware implementation is subject to physical hazards that can mitigate its security. To counter glitches, the most studied physical hazard, an effective solution is to resynchronize the signals by integrating additional hardware layers into the architecture. However, these solutions have an impact on the performance of the implementation. A solution to avoid these limitations is to use more shares to compute higher-degree functions. We study the cost of this approach, denominated $(td + n)$-masking. We first derive optimal dependence structures for the creation on non-complete sharings, which allow us to obtain efficient implementation of substitution boxes. As a case study, we use these elements to create a second-order masked architecture for the PRESENT cipher. We perform multiple TVLA tests to validate our approach. Our results confirm that the strategy is efficient in terms of performance, at the cost of increased hardware resources.

Keywords: Side-Channel Analysis · Low Latency · High Frequency · Low Randomness · Masking · SNI security

1 Introduction

Since their introduction in the late 1990s, Side Channel Analysis attacks (SCAs) [Koc96, KJJ99] have been a significant threat against cryptographic implementations. SCAs take advantage of the physical characteristics of a device (*e.g.* power consumption, execution time) to recover sensitive information. These physical characteristics depend on the internal state, which itself depends on the data being manipulated by the device (the sensitive information). Hence, there exists a link between the physical characteristics and the sensitive data. SCAs aim to exploit this link, threatening the security of cryptographic implementations.

Various implementation techniques have been designed to decrease the effectiveness of SCAs. Masking is one of the most popular countermeasures; it consists of applying secret sharing and performing computation on shares so that not all the shares are used simultaneously. Masking was originally presented in [GP99], and subsequently security models were developed. The most used is the probing model [ISW03], which uses the implementation order d as a security parameter. This order is the smallest statistical moment which can be exploited in a statistical attack (*i.e.* the smallest moment dependent on the secret). To resist dth-order attacks, an implementation requires at least a $(d + 1)$ secret sharing of all its sensitive data. The security and composability of all the basic field operations is ensured. Thus any function can be masked. The probing security model allows for the creation of efficient security proofs and can be linked to more realistic models, such as the noisy leakage model [DDF14]. Although presented in Boolean circuits, the probing model is best suited for software implementations. Hence, masking has been mostly applied in this domain.

Physical defects (*i.e.* glitches) have been exploited to break masked hardware implementations [MPG05, MPO05]. Indeed, glitches can break the independence of leakage of

the shares, which is necessary for any security reduction. This can reduce the effective order of the countermeasure. To limit the impact of glitches on $(d + 1)$ masking, various authors suggest using registers to limit their propagation [RBN+15, GMK16, FGP+18, CS21]. Nevertheless, the addition of multiple layers of registers to an architecture impacts its latency and area of implementation.

Recently, technological solutions have been explored to reduce the latency of masked implementations. These approaches avoid additional register layers by using locally asynchronous globally synchronous (LAGS) elements [SBHM20, SBB+22, NGPM22]. Using locally asynchronous circuits to synchronize signals and remove glitches allows to decrease the latency, but this comes at the cost of a reduction of the frequency. The frequency penalty comes from the addition of communication signals between the different parts of the circuit to ensure that glitches will not appear in the calculation.

Another approach for preventing glitches is based on multi-party computation on silicon and the so-called threshold implementations (TI) [NRR06, NRS09]. TIs were presented as efficient and glitch resistant first-order masked implementations. Efficiency was achieved thanks to the capability to compute over function sharings rather than decomposing the algorithm into Boolean operations. However, TIs required more shares than $d + 1$-masking as the algebraic degree of the function t affects the number of shares s. For first order, it is either $s = t + 1$ or $s = t + 2$ [PAB+23] compared to $s = 2$ for $(d + 1)$-masking. The advantage is that the function can be computed in one cycle, compared to at least $\mathcal{O}(log_2(t))$ cycles for $(d+1)$-masking. We use big O notation because the number of cycles per multiplication depends on the chosen method. TIs have been shown to achieve good performance/area trade-offs in various settings [BGG+17, BBK+13, PMK+11].

Creating TIs to resist higher-order and multivariate attacks is not trivial. First, the security order d increases the number of shares as $s \geq td + 1$. This greatly increases their implementation costs. Second, the *uniformity* of TIs is not sufficient to guarantee security against multivariate attacks [Rep15]. To solve this problem, the output of the TI gadgets must be *refreshed* [RBN+15]. However, refresh functions that use less randomness than SNI-refreshing have demonstrated some weaknesses [MMSS19]. Thus, synchronization is needed after each refresh operation. Consequently, the number of shares obtained from an TI gadget (equivalent to the number of function sharings) has a direct impact on the randomness requirement and the number of registers.

As discussed above, the implementation cost of TIs depends on the number of input shares, called here s_i, and the number of output shares (or function sharings), from here on called s_o. This distinction is necessary because some works have proposed strategies for creating TIs which play with both parameters [Pet19]. In that work, the author introduces non-complete set coverings as dependence structures for the creation of higher-order TIs. It is shown that these combinatory objects can guarantee *correctness* and the independence of sharings through *non-completeness*. In this case, defining $s_i > td + 1$ allows to reduce the number of function sharings s_o. Ideally, an efficient implementation will have $s_i \approx s_o$. However, for multiple $\{t, d\}$ instances $s_i \neq s_o$. Let $n = \max(s_i, s_o) - td$, thus we define this approach as $(td + n)$-masking. Our overall goal is to minimize n for a given $\{t, d\}$ pair, observing that $n \geq 1$.

In this work, we explore the efficiency of $(td + n)$-masking for creating low-latency masked implementations. We show that despite the increased implementation costs, these designs are interesting for low-latency and high-frequency applications. We show that these constructions offer alternative trade-offs compared to compositions which rely on extra registers (additional latency) or LAGS (low frequency). We summarize the different trade-offs that can be achieved with higher-order masking in hardware in Figure 1. The dashed ellipse corresponds to the contributions of this paper.

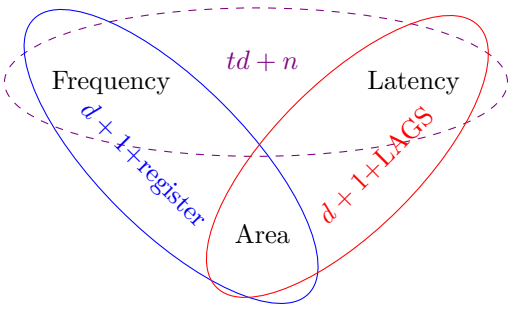


Figure 1: Representation of different Frequency-Latency-Area trade-offs for higher-order hardware masking.

Our contributions

In this work, we study the efficiency of $(td+n)$-masking for creating high-performance masked hardware implementations. To derive results on the optimality of the number of function sharings, we start by redefining the non-complete set coverings introduced by Petrides in [Pet19] in first-order logic. The theory of non-complete set coverings allows us to find the optimal $s_i \approx s_o$ balance, and its definition as first-order logic allows us to approach the problem as an optimization problem. We explore different solutions for the resulting optimization problem. Two are exact solutions that give a minimal bound on the implementation cost. However, finding the exact solution can be computationally expensive. Thus, we also present a greedy approach that can efficiently find a good solution, although not necessarily optimal.

After identifying optimal or good non-complete set coverings, we describe the process for the sharing of any function. We explore the sharing of 4-bit substitution boxes as a case study. The resulting designs are described in VHDL and prototyped in an FPGA to obtain implementation costs. The implementation results allow us to demonstrate that $(td+n)$-masking can yield low-latency and high-frequency masked hardware implementations, at the costs of increased area requirements.

To validate the soundness of the proposed approach, we also developed a second-order masked implementation of the PRESENT cipher and prototyped it in an FPGA. We used the ChipWhisperer system to perform TVLA with 10M power traces and show that leakages are not found. We complimented this study with the application of TVLA over the noiseless shares' outputs and show that the proposed degree of protection is met.

Paper organization

The remainder of the paper is organized as follows. Section 2 describes higher-order threshold implementations and non-complete set coverings. Section 3 presents a redefinition of optimal non-complete set coverings and reformulates the challenge as an optimization problem. Section 4 provides results for efficient non-complete set coverings and masked 4-bit substitution boxes. In Section 5, we discuss the refresh strategy for $(td+n)$-masking. Section 6 presents a second-order masked implementations of PRESENT using one of the masked substitution boxes from Section 4 and the efficient refresh strategy introduced in Section 5. In Section 7 we present TVLA experiments and results, which validate our approach. Lastly, Section 8 provides final remarks and concludes the paper.

2 Background

We denote sets using capital letters, with $\#(S)$ denoting the cardinality of a set S and $\emptyset$ denoting the empty set. We use the term t-set to refer to subsets of S with cardinality t. We use $\binom{k}{r}$ for the number of ways to choose a sample of r elements from a set of k distinct objects where order does not matter and replacements are not allowed. Let $\mathbb{F}_2$ represent the binary field and $\mathbb{F}_2^k$ the k-dimensional vector space over $\mathbb{F}_2$. Let $x, y \in \mathbb{F}_2$, then $x \wedge y$, $x \vee y$ and $x \oplus y$ represent the application of the AND, OR, and XOR gates, respectively. The application of the NOT gate over the input $x \in \mathbb{F}_2$ is represented by $\neg x$. Let X, Y two sets, then $X \cup Y$, $X \cap Y$, $X \subset Y$, and $X \subsetneq Y$ represent the union, intersection, inclusion, and proper inclusion of subsets, respectively.

2.1 Threshold implementations

Threshold implementations were introduced in [NRR06] as a solution for first-order masking in the presence of glitches. They attracted attention from academia and industry due to their simplicity and resistance in hardware implementation. In a few words, the idea behind TIs is to blind the input data x, represented by the sharing $(x_1, x_2, \ldots, x_{s_i})$, such that $x = \bigoplus^{s_i} x_i$. The t-degree function f is also split into function sharings $(f_0, f_2, \ldots, f_{s_o-1})$. This property is known as *non-completness*, and it is important to resist glitches. The reason behind this is that an adversary that can observe the circuit corresponding to a shared function f_i learns nothing about the secret from the definition of secret sharing. Indeed, the information the adversary can learn from a sub-circuit is independent of the secret since at least one share will be missing.

In order to perform useful computations, the shared functions should achieve *correctness*. This means that the outputs of the shared functions should correspond to a sharing of the output $y = f(x)$. Let $(y_1, y_2, \ldots, y_{s_o})$ represent the output sharing such that $y = \bigoplus^{s_o} y_i$. Then $y_i = f_i((x_1, x_2, \ldots, x_{s_i-1}))$, to provide an example.

Finally, to perform computations in a round-based approach (as required in most cryptographic algorithms), the output sharing should achieve *uniformity*. That means that the output of the function sharings $(f_0, f_2, \ldots, f_{s_o-1})$ ought follow the same distribution as the sharing of the output of the unshared function f. Hence, the output of one round can be used as input for the next round. This implies that an adversary could not exploit the leakages of different rounds. Otherwise issues may arise [Rep15].

Threshold implementations have been extensively studied in the literature. Various block-cipher implementations have been developed to validate this approach for first-order security [MPL+11, PMK+11, SBM18]. TIs were later generalized for higher-order security in [BGN+14]. However, the uniformity property could not achieve security against multivariate attacks [Rep15]. Thus, to offer resistance against multivariate attacks, TI gadgets must be complemented with SNI-refreshing [BBD+16]. Consequently, when considering higher-order TIs the main challenge is to find a non-complete separation of input shares to compute the shared functions in a higher non-complete manner, such that the number of output is not too large.

2.2 Non-completness of threshold implemenations

In [Pet19], Petrides studies the non-completeness in terms of constrained set coverings. He first introduces a formalism on the desirable properties of the sets and then derives constructions for non-complete input slicing. He also provides bounds for the number of output shares s_o as a function of the number of input shares s_i, the algebraic degree of the function t, and the security order d. He shows that non-complete set coverings are desirable constructions to split shares in a d non-complete manner for higher-order threshold implementations. We recall his definition for non-complete set covering.

Definition 1 (2.1 [Pet19])**.** Given positive integers s_i, t, d a non-complete set covering $S^{nc}_{(s_i,t,d)}$ is a set of subsets of $\mathcal{U}_{s_i} = \{1, 2, \ldots, s_i\}$ such that:

1. each t-subset of $\mathcal{U}_{s_i}$ is a subset of at least one element of $S^{nc}_{(s_i,t,d)}$,
2. each element of $S^{nc}_{(s_i,t,d)}$ has size at least t, and
3. no fewer than $d+1$ elements of $S^{nc}_{(s_i,t,d)}$ cover $\mathcal{U}_{s_i}$.

The first constraint allows correctness, the second constraint prevents non-optimal solutions (non-necessary for security), and the third one is the one that actually captures non-completeness.

We also recall some results from [Pet19] which are useful for our case. First, if we fix a minimal number of input shares (*i.e.*, $s_i = td+1$) then the number of output shares s_o will be maximal. This is reflected in the bound provided in Proposition 1 for the $td+1$ case.

Proposition 1 (2.8 [Pet19])**.** $s_o = \#\left(S^{nc}_{(td+1,t,d)}\right) = \binom{td+1}{t}$.

Evidently, such a generic construction would lead to a large number of sub-functions (and output shares). Thus, a large number of randomness would be required to perform the SNI-refreshing of the output. For efficient implementation, we may prefer to increase the number of input shares in order to reduce the number of output shares. Ideally, to keep the number of shares constant ($s_i = s_o$).

Proposition 2 (2.4 [Pet19])**.** *For every* $S \in S^{nc}_{(s_i,t,d)}$ *it holds that*

$$\#(S) \leq (s_i - ((d-1)t)) - 1.$$

Proposition 2 gives an upper bound on the cardinality of the elements of $S^{nc}_{(s_i,t,d)}$. This is useful for our approach as it reduces the search space and the associated exploration costs. Petrides claims that $S^{nc}_{(s_i,t,d)}$ with the smallest cardinality are of particular interest since they will reduce the implementation costs. He denotes $\hat{S}^{nc}_{(s_i,t,d)}$ an $S^{nc}_{(s_i,t,d)}$ with the smallest possible cardinality.

Corollary 1 (2.11 [Pet19])**.** *For* $s_i \geq td+1$ *we have* $\#\left(\hat{S}^{nc}_{(s_i,t,d)}\right) \geq \#\left(\hat{S}^{nc}_{(s_i+1,t,d)}\right)$.

Corollary 1 is useful when considering the bound on the number of elements in $\hat{S}^{nc}_{(s_i,t,d)}$. This allows us to reduce the search space, which can lead to more solutionable systems.

3 Search of non-complete set coverings

In this section, we present different manners to represent the non-complete set covering problem as an optimization problem. The different representations serve various purposes, such as finding an optimal solution, proving lower bounds, or finding a "good" solution (and thus setting an upper bound).

3.1 First-order logic

We first redefine the non-complete set covering problem given in Definition 1 to adjust the second constraint. The others remain the same, but are rewritten in first-order logic [Bar77].

Definition 2 (Non-complet set covering)**.** Given positive integers s_i, t, d a non-complete set covering $S^{nc}_{(s_i,t,d)} \subset \mathcal{P}(\mathcal{U}_{s_i})$ such that:

1. $\forall C \in \mathcal{P}(\mathcal{U}_{s_i}), \#(C) = t \Rightarrow \exists D \in S^{nc}_{(s_i,t,d)}, C \subset D$,

2. $\forall D, D' \in S^{nc}_{(s_i,t,d)}, D \neq D' \Rightarrow D \nsubseteq D' \wedge D' \nsubseteq D$,

3. $\forall I \subset S^{nc}_{(s_i,t,d)}, \#(I) = d \Rightarrow \cup_{i \in I} i \neq \mathcal{U}_{s_i}$.

The first and third properties are equivalent to those presented in [Pet19], while the second is stricter. Note that for property 3, we only consider equality rather than the inequality defined by Petrides. If the property is false for a set strictly smaller than d, then it is possible to construct a set of size d that contradicts it (except for degenerate cases that are not of interest).

We can show that the properties reformulated in Definition 2 imply those of Definition 1. Assume $D \in S^{nc}_{(s_i,t,d)}$ and $\#(D) < t$, $\exists E \in \mathcal{P}(\mathcal{U}_{s_i})$ such that $D \subset E$ and $\#(E) \geq t$. From property 1) $\exists F \in S^{nc}_{(s_i,t,d)}$ such that $E \subset F$. We thus have $D \subset F$, which would contradict property 2). Definition 2 states that non-complete set coverings do not contain subsets of cardinality smaller than t. Thus, the same proofs from Petrides apply for Corollary 1. Therefore, we retain the upper $(s - ((d-1)t) - 1)$ and lower (t) bounds on the sets to be considered. These bounds reduce the search space. We denote the set of possible subsets in $S^{nc}_{(s_i,t,d)}$ by $\mathcal{P}(\mathcal{U}_{s_i}^{\setminus(t,d)})$.

To find "good" non-complete set coverings, our method enumerates all the elements of $\mathcal{P}(\mathcal{U}_{s_i})$ (excluding trivial solutions) and selects the one to have a non-complete set covering with the smallest cardinality. Since we list all elements in $\mathcal{P}(\mathcal{U}_{s_i})$, this method has exponential complexity in s_i. Thus, we consider only small values of s_i. However, these also represent the interesting cases for masked implementations.

3.2 Conjunctive normal form

From the properties in Definition 2 we can derive an algebraic system of constraints. This system can be represented in CNF and then we may use an off-the-shelf solver to find possible non-complete set coverings. Each element of $B \in \mathcal{P}(\mathcal{U}_{si})$ is represented as a Boolean variable b. We represent each possible subset with respect to the cardinality bounds with a literal; any other sets are not present in $S^{nc}_{(s_i,t,d)}$ and thus are not considered. This produces a bijection between the literals and the sets. Then, for each property, we derive a subsystem. In the following, we present the CNF representation of the three properties.

Property 1: correctness

Specify for each literal a representing a set A of size t. List all sets that contain A and state that one of these literals should be true. The Algorithm 1 presents how such clauses can be constructed.

Algorithm 1 Clause construction for the presence of all t-subsets.

Require: A set A of size t and its associated literals a.
Ensure: A clause that ensures that at least 1 chosen set contains A.

```
Clause← a
for B ∈ P(U_si^{\(t,d)}) do
    if A ⊊ B then
        clause← clause ∨b
```

From Algorithm 1, we can see that the number of literals in the clause is $\#(literals) = \sum_{i=1}^{s_i - dt - 1} \binom{s_i - t}{i}$. Evidently, the number of literals in the clause grows exponentially with the size of the universe $\mathcal{U}_{s_i}$.

Property 2: uniqueness

For all possible sets, verify whether the set contains the set under consideration; if so, add a clause stating that at most one of the two sets should be in $S^{nc}_{(s_i,t,d)}$. The Algorithm 2 presents how such clauses can be constructed for a set A.

Algorithm 2 Clauses construction for the uniqueness of all t-subsets.

Require: A set A and its associated literals a.
Ensure: Clauses that ensure that A is selected only if not set containing A is selected.
1: Clauses$\leftarrow True$
2: **for** $B \in \mathcal{P}(\mathcal{U}_{s_i}^{\setminus(t,d)})$ **do**
3: **if** $A \subsetneq B$ **then**
4: clauses$\leftarrow$ clauses $\wedge \neg a \vee \neg b$

Here, we can see that the number of clauses grows exponentially with the size of the universe. However, each clause is composed of only two literals, and thus is easily manageable by most solvers.

Property 3: non-completeness

For all possible sets of sets, verify whether the union of sets contains the universe; if so, add a clause saying that one of the literals representing these sets should be false.

Algorithm 3 Clause construction for d-order non-completeness.

Require: .
Ensure: .
1: clauses$\leftarrow True$
2: **for** $\{B_i\}_{i\in I} \subset \mathcal{P}(\mathcal{U}_{si}^{\setminus(t,d)}), \#(I) \leq d$ **do**
3: **if** $\cup_i B_i = \mathcal{U}_{si}$ **then**
4: clause$\leftarrow False$
5: **for** $B_i \in \{B_i\}$ **do**
6: clause $\leftarrow \vee \neg b_i$
7: clauses$\leftarrow$ clauses $\wedge$ clause

Here, we can see that the number of clauses grows exponentially with the security order d, but each clause is made up of at most d literals.

Cardinality

To limit the cardinality of the non-complete set coverings, we decide to add an additional constraint. With this we can deduce a lower bound on the number of output shares when the system cannot be solved. Otherwise, the solver produces a solution that respects the cardinality constraint.

The CNF representation of cardinality constraints is a common problem. Indeed, the basic solution will add $\binom{s}{d}$ clauses (one for each combination of d elements among s) with d literals each. Thus, the system description grows exponentially with the number of input shares. In the literature there are solutions that achieve linear growth by using additional literals. An efficient solution was proposed by Sinz [Sin05], we recall the equations proposed in Algorithm 4.

The idea of Sinz was to represent the cardinality constraint as a sequential counter and fix the maximum weight. Then, he used the Tseytin transformation that allows us to represent any Boolean circuit as a CNF system [Tse83].

Algorithm 4 Clause construction for cardinality constraint.

Require: $(a_1, \ldots, a_n)$ a word of size n and a constraint on the number of litterals assigned to true k.
Ensure: A CNF clauses.

clauses$\leftarrow \neg a_1 \vee b_{1,1}$
for $1 < j \leq k$ **do**
    clauses$\leftarrow$ clauses $\wedge \neg b_{1,j}$
for $1 < j < n$ **do**
    clauses$\leftarrow$ clauses$\wedge \neg a_i \vee b_{i,1}$
    clauses$\leftarrow$ clauses$\wedge \neg b_{i-1,1} \vee b_{i,1}$
    for $1 < j < k$ **do**
        clauses$\leftarrow$ clauses$\wedge \neg a_i \vee \neg b_{i-1,j-1} \vee b_{i,j}$
        clauses$\leftarrow$ clauses$\wedge \neg b_{i-1,j} \vee b_{i,j}$
    clauses$\leftarrow$ clauses $\wedge \neg a_i \vee b_{i-1,k}$
clauses$\leftarrow$ clauses $\wedge \neg a_n \vee b_{n-1,k}$

3.3 Greedy search

For a large number of input shares the SAT solutions require large amounts of memory and time to find an optimal solution. This is a problem when computing resources are limited. Thus, we choose to build a system that may output a non-complete set covering without guarantees on the optimality of the number of output shares. This is particularly useful when considering large order or large algebraic degrees.

The main idea of this approach is to list all the sets which are required from property 1 of Definition 2. Then, we build a list of sets complying with condition 3 either by adding a new element of the first list or by merging a new element in an existing set. Finally, we update the list of all sets by removing elements that are now covered by the selected elements. Unfortunately, we did not manage to find an efficient solution for all possible cases. Thus, we decide to implement a randomized approach and select the most efficient solution in the end. Our approach to build the list of sets is described in Algorithm 5.

Algorithm 5 Random search for non-complete set covering.

Require: s_i number of input shares, t algebraic degree of the target function, d non-complete order.
Ensure: A non-complete set covering.

List_set_to_have $\leftarrow$list of all combinations of t elements in $\{0, \ldots, s_i\}$
$Shuffle(List_set_to_have)$
$List_set_selected.append(List_set_to_have.pop(0))$
while $List_set_to_have \neq \emptyset$ **do**
    $element = List_set_to_have.pop(0)$
    for $se \in List_set_selected$ **do**
        Replace se by $se \cup element$
        if *List_set_selected* verify Defn. 1(3) and Prop. 2 **then**
            Remove from *List_set_to_have* elements that are subset of $se \cup element$
            Break
        else
            Replace $se \cup element$ by se in *List_set_selected*
    if *element* not in an element of *List_set_selected* **then**
        $List_set_selected.append(element)$

We repeated this experiment a fixed number of times and kept only the solution with the smallest cardinality for *List_set_selected*.

4 Non-complete set coverings for $(td + n)$-Masking

Non-complete set coverings allow to simplify the creation of masked architectures at the cost of additional hardware resources. In this Section we describe how this approach can contribute to the creation of high performance masked architectures.

4.1 Optimal non-complete set coverings

We have used the results in Section 3 to find new non-complete set coverings with lower cardinalities than those presented in [Pet19]. Moreover, in some cases we can also conclude that these constructions are optimal (in virtue of being found by using the algorithms introduced in the previous section).

We are interested in the $t = 3$ cases as the functions of interest (S-boxes) can be decomposed in cubic functions. We also focus on higher-order constructions ($d \geq 2$) as non-complete set converings do not take into account uniformity, and thus require additional refresh gadgets. It is known that higher-order threshold implementations are vulnerable against multivariate attacks when considering only uniformity [Rep15]. Therefore, if the refresh requirement is already satisfied we can provide higher order security at no additional cost.

4.1.1 $t = 3, d = 2$

Sharing for degree three is of paramount importance for cryptography. Cubic functions allow to perform the sharing of 4-bit S-boxes in a single cycle. Furthermore, 8-bit S-boxes can be decomposed in two cubic functions. In particular, Arribas et al. suggest to decompose the inversion (x^{254}) in the AES S-box into two cubic functions $\left(x^{26}\right)^{49}$ [WM19, WDM20, AZN21]. In that work, the authors propose a first-order masked implementation but they do not report any results for higher-order masking.

Petrides in [Pet19] presented several non-complete set coverings for this case:

- $s_i = 7, t = 3, d = 2, s_o = 35$, claimed to be optimal.
- $s_i = 8, t = 3, d = 2, s_o = 18$.
- $s_i = 9, t = 3, d = 2, s_o = 13$.

Note that the large values of s_o would lead to an important number of input shares for the subsequent layer in a hardware implementation. We report hereafter some interesting constructions.

- $s = 8, t = 3, d = 2, s_o = 17$, which is optimal. In this case $n = 11$.

$$S^{nc}_{(8,3,2)} = \left\{ \begin{array}{c} \{0,1,2,6\},\{0,1,3,4\},\{0,1,5,6\},\{0,1,6,7\},\{0,2,3,5\},\{0,2,4,7\}, \\ \{0,3,6,7\},\{0,4,5,7\},\{0,4,6,7\},\{1,2,3,4\},\{1,2,5,7\},\{1,3,4,5\}, \\ \{1,3,4,6\},\{1,3,4,7\},\{2,3,6,7\},\{2,4,5,6\},\{3,5,6,7\} \end{array} \right\}$$

- $s = 9, t = 3, d = 2, s_o = 12$, with a lower bound on its cardinality. Here $n = 6$.

$$S^{nc}_{(9,3,2)} = \left\{ \begin{array}{c} \{0,1,2,4,5\},\{0,1,2,4,5\},\{0,1,6,7,8\},\{0,2,3,6,8\},\{0,2,4,7,8\}, \\ \{0,3,5,7,8\},\{0,4,5,6,7\},\{1,2,3,5,7\},\{1,2,5,6,8\},\{1,3,4,7,8\}, \\ \{2,3,4,6,7\},\{3,4,5,6,8\} \end{array} \right\}$$

- $s = 10, t = 3, d = 2, s_o = 10$, which allows for a constant datapath. Indeed, $n = 4$.

$$S^{nc}_{(10,3,2)} = \left\{ \begin{array}{c} \{0,1,2,3,6,8\}, \{0,1,2,4,7,9\}, \{0,1,2,5,8,9\}, \{0,1,3,4,5,8\}, \\ \{0,2,4,5,6,7\}, \{0,3,6,7,8,9\}, \{1,3,5,6,7,9\}, \{2,3,4,5,6,9\}, \\ \{2,3,4,7,8,9\}, \{1,4,5,6,7,8\} \end{array} \right\}$$

- We also report a construction with $s = 12, t = 3, d = 2, s_o = 9$, such that $n = 6$.

$$S^{nc}_{(12,3,2)} = \left\{ \begin{array}{c} \{0,1,2,4,6,10,11\}, \{0,1,2,3,4,7,8,9\}, \{0,1,3,5,6,7,9,11\}, \\ \{0,2,5,7,8,9,10,11\}, \{0,3,4,6,8,9,10,11\}, \{0,4,5,6,7,8,9,10\}, \\ \{1,2,3,5,6,8,9,10\}, \{1,3,4,5,7,8,10,11\}, \{2,3,4,5,6,7,9,11\} \end{array} \right\}$$

Note that to maintain a consistent width in a hardware datapath, an implementer may chose to use $S^{nc}_{(10,3,2)}$ which maintains the same number of outputs and inputs. Alternatively, they may chose to concatenate two layers based on $S^{nc}_{(9,3,2)}$ and $S^{nc}_{(12,3,2)}$ which can lead to consistency in the width of the linear layers. We note that un-sharing the intermediate values to use a single S-Box is not ideal as an additional refresh layer would be required and it may lead to unintended leakages.

As proven by [Pet19], constraining $s_i = 7$ leads to $s_o = 35$. It is unlikely that this construction is useful for practical use. Thus, we aimed at reducing the number of output shares by increasing the number of input shares. For example, in the case $s_i = 8$ we managed to obtain a non-complete set covering of cardinality $s_o = 17$. The system was unsatisfiable for lower bounds on s_o. These results can be obtained with CNF applying a cardinality constraint as described in Section 3.

Given $\hat{S}^{nc}_{(8,3,2)} > 16$, the non-complete set covering presented earlier is optimal in terms of the number of sub-circuits. This implies that a large amount of hardware resources would be required for an implementation. Moreover, we did not manage to find a non-complete set covering $S^{nc}_{(17,3,2)}$ with $s_o = 8$. The solver did not conclude after 72 hours. To reduce the number of output shares, we thus needed to increase the number of input shares. For $s_i = 9$ we managed to find a solution with 12 output shares. We also found that the system was unsatisfiable for $s_o = 9$, thus it follows that $\hat{S}^{nc}_{(9,3,2)} > 9$. For $s_o = 10$ and $s_o = 11$ the solver did not find a solution after 72 hours.

To use this solution in a hardware implementation for AES, we would need the second cubic function to use $S^{nc}_{(12,3,2)}$ and satisfy $s_o = 9$. And we found a non-complete set covering with these characteristics. The non-constant number of shares can be combined in a way that the width of the datapath is consistent for the non-linear layers. This representation leads to nine shares for the linear part. A similar strategy was proposed in [ABP+18] for a first-order masked implementation of Keccak.

If a fixed number of shares is preferred, at least ten shares would be needed since $\#(\hat{S}^{nc}_{(9,3,2)}) > 9$. We searched for a $S^{nc}_{(10,3,2)}$ with a cardinality constraint $s_o \geq 10$, as in previous experiments we were unable to find an $S^{nc}_{(9,3,2)}$ with a lower number of elements than 12. The constant number of share can be useful to use a common randomness source across different gadgets [FPS17]. We also note that this constructions obtained the minimum value of n for 3rd-degree and 2nd-order.

4.1.2 $t = 3, d = 3$

We conducted several experiments to find non-complete set coverings for third order of security. Unfortunately, the solver could not find any interesting results for implementation. Solutions were found for for $s_i = 12$ with $s_o \in \{67, 71, 72, 76, 80, 90, 100\}$. We also found solutions for the cases $s_i = 15$ ($s_o = 46$), $s_i = 16$ ($s_o = 40$), and ${}_s i = 17$ ($s_o = 35$). For a

larger number of input shares the enumeration part in the system constructions required to much time/memory to find any results. Observe that the value of n is quite large in these cases.

4.1.3 $t = 2, d = 3$

To reduce the number of shares for order 3, we also considered degree 2. However, this would not contribute to reduce the latency of an implementation. Indeed, using degree 2 requires almost 2 steps for most 4-bit S-Box and 3 steps for degrees higher or equal to 5 (thus also for the AES).

- $s_i = 9, t = 2, d = 3, s_o = 11$ and $n = 5$.

$$S^{nc}_{(9,2,3)} = \left\{ \begin{array}{c} \{2,3,6\},\{8,1,6,0\},\{0,8,3,7\},\{8,4,5\},\{1,2,7\}, \\ \{0,2,4\},\{3,4,6\},\{0,1,3,5\},\{1,4,7\},\{5,6,7\},\{8,2,5\} \end{array} \right\}$$

- $s_i = 10, t = 2, d = 3, s_o = 11$ and $n = 5$.

$$S^{nc}_{(10,2,3)} = \left\{ \begin{array}{c} \{1,5,6\},\{8,9,3,6\},\{2,6,7\},\{0,5,7\},\{1,3,4,7\},\{0,1,2,3,8\}, \\ \{0,2,4,6\},\{2,3,4,5,8\},\{0,9,5\},\{8,9,4,1\},\{8,9,2,7\} \end{array} \right\}$$

- $s_i = 11, t = 2, d = 3, s_o = 10$ and $n = 5$.

$$S^{nc}_{(11,2,3)} = \left\{ \begin{array}{c} \{8,2,4,6\},\{0,4,8,9,10\},\{8,3,5,7\},\{0,5,6,7,10\},\{9,2,3,6\}, \\ \{9,3,4,5\},\{1,3,4,7,10\},\{0,1,2,7,9\},\{0,1,3,6,8\},\{1,2,10,5\} \end{array} \right\}$$

We observe that non-complete set coverings are large for higher-order. It is possible to use $S^{nc}_{(11,2,3)}$ with an additional null output or switching between 11 and 10 shares. Nevertheless, these strategies would require quite a large number of shares for a low degree function compared to classical AND-based masked implementation.

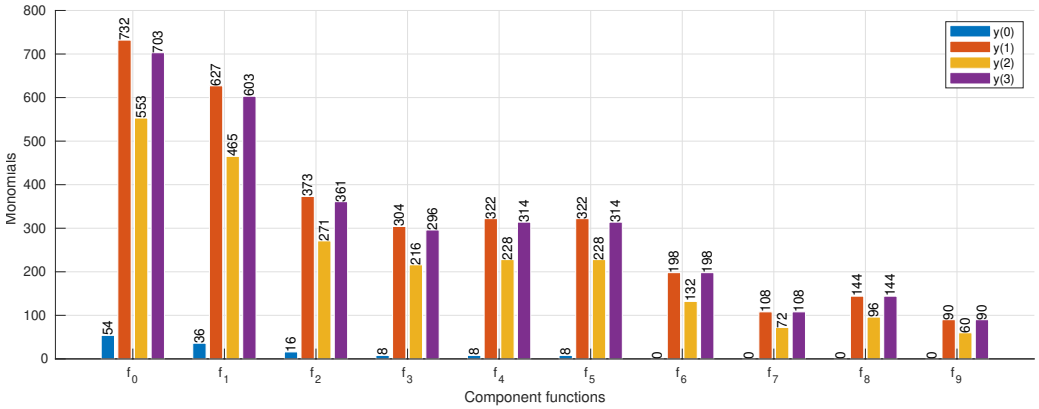


Figure 2: Number of monomials in the shared sub-functions of the 4-bit S-Box from (1)

4.2 Generation of the shared functions

There are various ways to construct the component function from an S-Box and a non-complete set covering. First, we have to choose a formula to represent the S-Box. We can select a Boolean formula to avoid complex field multiplications. We can reduce the number of candidates to those that contain monomials of degree at most t (for example CNF representations may have higher degrees). Hence, we choose the Algebraic Normal Form (ANF) representation of the function.

From the ANF, we then build the component functions. For each monomial in the ANF, we calculate its shared version using the number of shares required. Then, we distribute the shared monomial between the component functions according to the chosen non-complete set covering. Contrary to the construction of first order TIs, we do not need to explore position of these shares in different component functions to achieve uniformity. Recall that SNI refreshing is anyways required to guarantee security against multivariate attacks.

As an example, observe the ANF representation of a 4-bit S-Box provided in (1). And assume we want to create a sharing of this S-Box using the non complete set covering $S^{nc}_{(10,3,2)}$ reported in Subsection 4.1.1. Let $X = \{x(0), x(1), x(2), x(3)\} \in \mathbb{F}_2^4$ represent its input and $Y = \{y(0), y(1), y(2), y(3)\} \in \mathbb{F}_2^4$ its output. First, note how the monomials with the greatest algebraic degree are cubic. Thus, a non complete set covering with $t = 3$ is enough to ensure correctness of this implementation. Also note that $s_i = 10$ implies that any cubic monomial in (1) will produce 1,000 monomials for the masked function. We can calculate that the shared representation of $Y = \{y(0), y(1), y(2), y(3)\}$ will contain $\{130, 3220, 2321, 3131\}$ monomials, respectively. These monomials will be split among the $s_o = 10$ component functions. The distribution of the "expanded" monomials within the shares is illustrated in Fig. 2. For simplicity, the implementation prioritizes allocating the monomials in the first viable sub-function (respecting the non-completeness condition of the non-complete set covering). Therefore, some sub-functions will be quite large and others will be relatively small or even zero (notably the shares of $y(0)$). This has the effect that for the masked S-Box four of its outputs will always be zero. Therefore, the output space is effectively reduced to 36 shares instead of 40.

$$
\begin{aligned}
y(0) &= x(0) + x(2) + x(1)x(2) + x(3) \\
y(1) &= x(1) + x(0)x(1)x(2) + x(3) + x(1)x(3) + x(0)x(1)x(3) + x(2)x(3) + x(0)x(2)x(3) \\
y(2) &= 1 + x(0)x(1) + x(2) + x(3) + x(0)x(3) + x(1)x(3) + x(0)x(1)x(3) + x(0)x(2)x(3) \\
y(3) &= 1 + x(0) + x(1) + x(1)x(2) + x(0)x(1)x(2) + x(3) + x(0)x(1)x(3) + x(0)x(2)x(3)
\end{aligned}
\tag{1}
$$

5 Refreshing

We use SNI refreshing to guarantee the uniformity of the output of the shared gadgets. Our idea is to use a refresh gadget according to the chosen number of output shares. This function would require $\frac{s_o(s_o-1)}{2}$ random bits, as described in Equation 2.

$$
\begin{bmatrix}
0 & r_{0,1} & r_{0,2} & r_{0,3} & \dots & r_{0,s_o-1} \\
r_{0,1} & 0 & r_{1,2} & r_{1,3} & \dots & r_{1,s_o-1} \\
r_{0,2} & r_{1,2} & 0 & r_{2,3} & \dots & r_{2,s_o-1} \\
 & \vdots & & \vdots & \dots & \vdots \\
r_{0,s_o-1} & r_{1,s_o-1} & r_{2,s_o-1} & r_{3,s_o-1} & \dots & 0
\end{bmatrix}
\tag{2}
$$

We observe, however, that the refresh gadget presented in Equation 2 achieves $s_o - 1$ SNI security, while the security targeted in our implementations is $d << s_o$.

6 Implementation costs

In this section we detail the implementation of second-order protected architecture for PRESENT. This cipher is ideal for our case study as its S-Box can can be represented as cubic functions as shown in (1). Thus, we can generate a masked architecture without increasing its latency.

6.1 S-Box masking

We first evaluated the area and performance of three different masked S-Box which generated using the non-complete set coverings described in Subsection 4.1.1. We implemented these gadgets for the Alveo U200 Data Center Acceleration card, using the AMD-Xilinx 2020.2.2 toolchain to generate the configuration files for the FPGA. In order to preserve the integrity of the shares, we described each function as an individual component. Then we directed the synthesizer to preserve the hierarchy of the hardware description. These implementation results are shown in Table 1. From this analysis, the S-Box derived from the $S^{nc}_{(9,3,2)}$ obtains the smallest footprint. However, as it must be used in conjunction with an S-Box derived from the $S^{nc}_{(12,3,2)}$, its area advantage is nullified. On the other hand, the symmetric configuration obtained with the $S^{nc}_{(10,3,2)}$ shows a similar area profile while requiring only one type of such S-Box.

Table 1: Results on the Alveo U200 Data Center Accelerator Card (xcu200-fsgd2104-2-e)

S-Box	LUT	CLB	CLB %	f_{max} (MHz)
$S^{nc}_{(9,3,2)}$	905	140	0.09%	575
$S^{nc}_{(10,3,2)}$	1112	167	0.11%	575
$S^{nc}_{(12,3,2)}$	1670	249	0.17%	510

6.2 Hardware refreshing

By selecting an S-Box sharing created with the $S^{nc}_{(10,3,2)}$, as $s_i = s_o = 10$ each bit of the state will be represented using ten shares. This means that a PRESENT nibble (4-bit) will have a length of 40-bit, and the complete state will be 640-bit. The encryption key will also be masked using this approach. We employ an additional masked S-Box for the key schedule and store the masked key on an 800-bit register. To ensure uniformity of the masked S-Box output, each bit of the state and the key must be refreshed with random data before being stored in their registers. In total 64+80 refresh functions are used.

Given the refresh function from Section 5, 180 random bits are required each round to update a single data nibble. Another 180 random bits are also required to mask each key nibble. Since an iterative architecture for PRESENT processes the 16 nibbles of the state in parallel, then $(180 + 180) \times 16$ random bits are required each cycle. With 31 rounds, such an implementation consumes $(180 + 180) \times 16 \times 31$ random bits, or ∼22kB of random data.

The production of large volumes of true random numbers falls outside of the scope of this work. Instead, we use pseudo-random sequences to demonstrate the effectiveness of the proposed masking approach. We have opted for maximal period LFSR-based generators [Alf96] for this end. We use three LFSR with lengths larger than 120 bits to create a 360-bit pseudo-random sequence. In total 48 LFSRs are used to produce the sixteen 360-bit pseudo-random sequences required by a masked PRESENT architecture.

This strategy allows to produce the necessary volume of pseudo-random data so long as certain restrictions are observed:

- Upon reset, all LFSRs are clocked 2^{20} times to ensure that their output is uniform.
- The LFSRs are clocked every time their output is used.
- The PRNG is not reset until a complete set of traces has been captured.
- No more than $2^{120} - 2^{20}$ traces are captured.

The proposed architecture for generating the necessary pseudo-random sequences for a single nibble of PRESENT is illustrated in Fig. 3. Note that this design is just for demonstrative purposes. In a practical application the random data must be produced by a standard-compliant TRNG module.

6.3 Unprotected PRESENT architecture (PRE)

One of the most important constraints of a protected hardware implementation is ensuring that the state is stored in its register only after it has been refreshed. This is simplified in iterative architectures as the data is registered once per round. Thus, we opted to employ the iterative PRESENT architecture shown in Fig. 4. We employ the 80-bit key variant since we are interested in this architecture simply as case study. This design will be referred as PRE throughout this section.

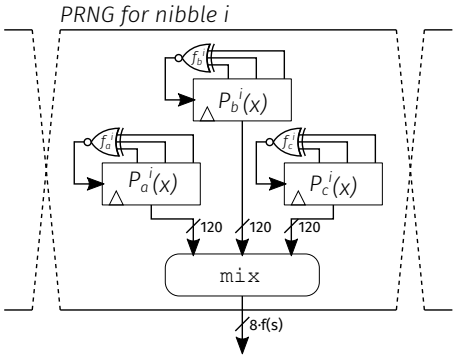


Figure 3: The PRNG architecture for a single nibble of PRESENT

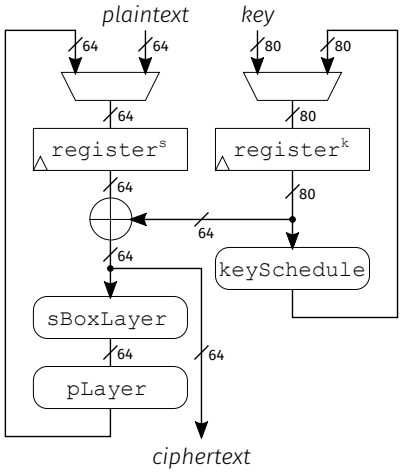


Figure 4: An iterative architecture of PRESENT

When implemented on the Alveo U200, the PRE architecture requires just 220LUT/180FF, which is in line with the state of the art for PRESENT [LNDPMS17]. On the same platform, the architecture achieves a maximum frequency of 600 MHz. Note that this is close to the maximum frequency of the fastest masked S-Box.

6.4 Masked PRESENT architecture (PREM)

We have created a protected architecture for PRESENT using the masked S-Box and the refresh gadget presented in earlier. This design is referenced as PREM. Upon loading the plaintext, the refresh function will ensure that all the input bits are stored as ten shares. The S-Box layer is simply 16 masked S-Box instead of regular S-Box. The bit-level

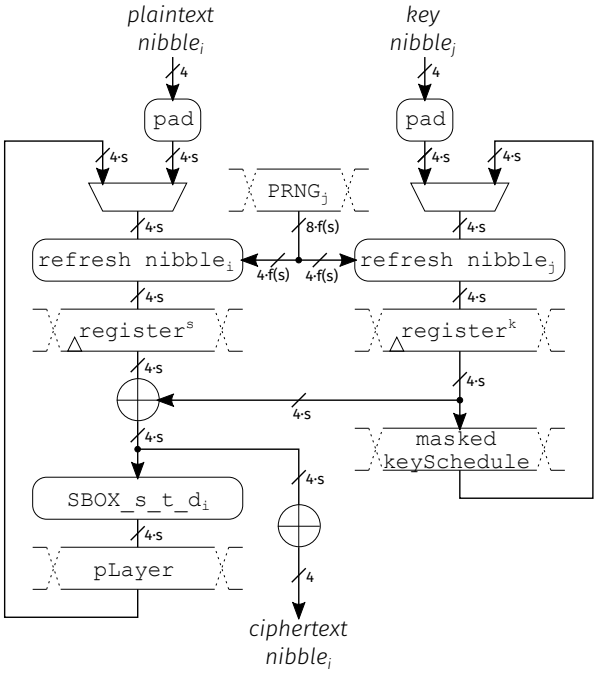


Figure 5: An iterative protected architecture of PRESENT. Note that we illustrate the operations over a single nibble.

permutation of PRESENT is unaffected, but now it has to operate at the level of shares. Lastly, the output is obtained by simply XORing the ten shares of each bit to obtain a bit of the ciphertext. The masked key schedule is straightforward. It employs an additional masked S-Box and produces a blinded round-key each cycle. This architecture is illustrated in Fig. 5. We prototyped this architecture for the Alveo U200 Data Center Accelerator card using the AMD-Xilinx Vivado 2020.2.2 toolchain. The implementation results are presented in Table 2.

Table 2: Results on the Alveo U200 Data Center Accelerator Card (xcu200-fsgd2104-2-e)

Arch.	FF	LUT	CLB	CLB %	f_{max} (MHz)	Cycles
PRE	179	298	60	0.04%	600	33
PREM	1,491	23.066	4,631	3.13%	250	33
PRNG*	7,867	146	1,866	1.26%	-	1

* Includes 18 PRNG modules.

As expected, the large size of the masked S-Box has an impact on the resource utilization of the protected architectures. In an iterative design, 16 of such S-Box are used in parallel. Another S-Box is required in the key schedule. As a result, the protected implementation is two orders of magnitude larger than the unprotected design. However, note that this cost represents only a small fraction ($\sim 3\%$) of the resources available in modern FPGAs.

Another detail to be observed is that the maximum frequency of the masked architecture is halved w.r.t. the unprotected architecture. This can be justified as the larger critical path of the masked S-Box is added to the design without additional registers. However, we note that 250 MHz is also the maximum operational frequency for the DDR controllers in AMD-Xilinx Ultrascale+ devices.

7 Testing

In this Section, we conduct tests to verify the effectiveness of the proposed masking approach. We rely on the Test Vector Leakage Assessment methodology [SM16] to evaluate the masked PRESENT architecture described in Section 6. We also use TVLA on the noiseless model of the masked S-Box of PRESENT to show that the proposed degree of protection is met.

TVLA is a popular method to corroborate whether there is a relationship between the inputs of a cryptographic algorithm and its side channel footprints. It relies on Student's t-test for this end. In our study, the inputs of interest are the plaintexts ($\mathcal{P}$) and the encryption keys ($\mathcal{K}$). Several authors suggest to use non-specific t-tests as they can detect a wide range of leakages, independently of how precise is our understanding of the architecture under analysis. The drawback is that they require large sets of observations. On the other hand, specific t-tests can detect punctual leakages with greater sensibility. However, they require more analysis effort. In this work we focus on the first case.

7.1 Test of the hardware prototypes

All the architectures were implemented on the NewAE CW305 platform, which integrates an AMD-Xilinx Artix-7 FPGA (xc7a100-tftg256-2). We then used a ChipWhisperer-Lite card to perform the trace acquisition. For each architecture we obtained two sets of power traces. The details about these data are summarized in Table 3. These sets follow the standard TVLA methodology and alternate randomly between fixed (F) or random (R) plaintexts (D) and keys (K). These data are publicly available in the associated artifact (to be created).

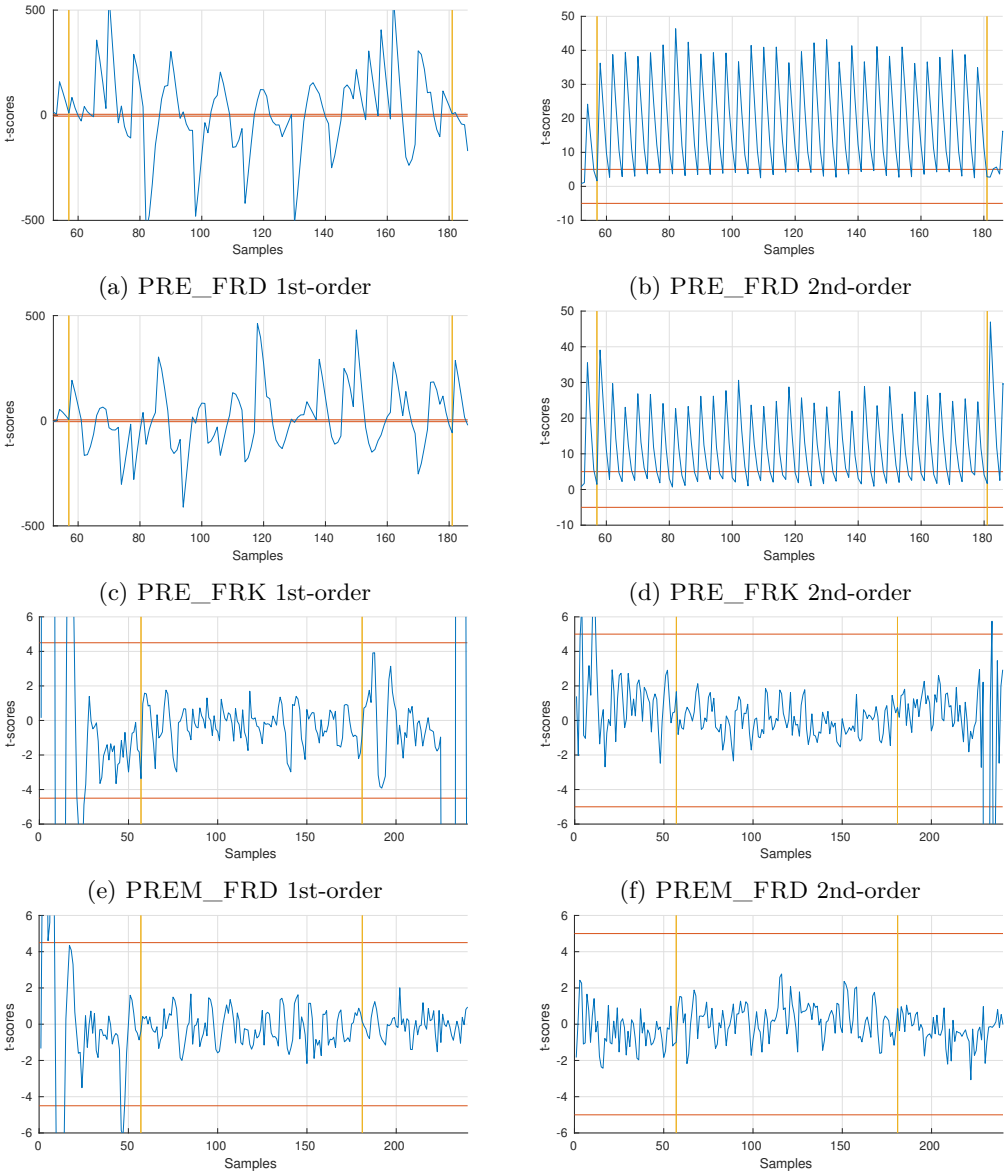
Table 3: The different datasets employed in our leakage assessment methodology

Arch.	Dataset	D	K	Traces	Samples	Victim (MHz)	Sensor (MSpS)
PRE	PRE_FRD	F/R	F	10e6	240	10	40
	PRE_FRK	R	F/R	10e6	240	10	40
PREM	PREM_FRD	F/R	F	10e6	240	10	40
	PREM_FRK	R	F/R	10e6	240	10	40

As the S-Box used by the protected architectures of PRESENT offers an order of coverage $d = 2$, we also evaluated their second order leakage. For this, instead of the raw leakages, we used the variance of the original sets for a second order analysis. These results are presented in Fig. 6. For the non-protected implementation there were considerable leakages throughout the execution trace. In contrast, the protected architectures only show leakages in their initial and final processing instants. These times correspond to the load of the plaintext, its sharing, and the un-sharing operation which produces the final ciphertext. A similar phenomenon is observed for the second order leakages.

The second step on the TVLA methodology requires the acquisition of a new dataset where the values of the key are changed from a fixed value to random keys (FRK). In this model, the plaintexts are always aleatory. With this second evaluation we sought to investigate the leakage associated with the key schedule. As described before, we use a masked S-Box and additional refresh functions for protecting the key datapath. Therefore, as expected, there were no leakages associated with the key schedule either.

For the case of second order leakage evaluation we took an additional verification step. We wanted to ensure that there were no bivariate leakages. In this case we constructed new traces consisting of combinations of two samples from each trace. Discarding repetitions and not allowing replacement, the length of the new traces is defined by the number of combinations $\binom{n}{r}$ where n is the number of samples in the trace and $r = 2$. In our case, we



(a) PRE_FRD 1st-order

(b) PRE_FRD 2nd-order

(c) PRE_FRK 1st-order

(d) PRE_FRK 2nd-order

(e) PREM_FRD 1st-order

(f) PREM_FRD 2nd-order

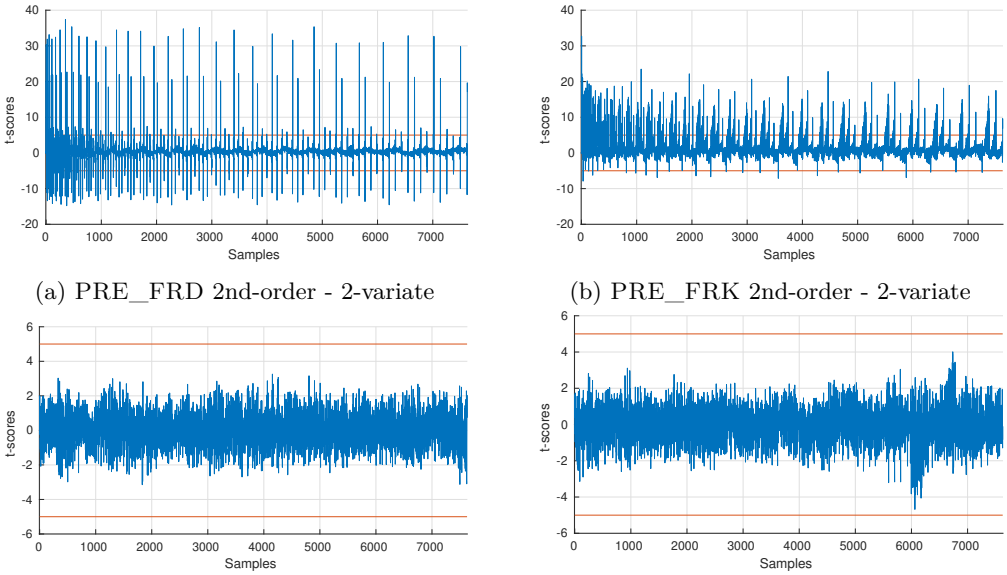
(g) PREM_FRK 1st-order

(h) PREM_FRK 2nd-order

Figure 6: Univariate t-test evaluation of the hardware implementations of PRESENT. Test results were derived using 10e6 traces. The vertical lines represent the start and end of the 31 PRESENT iterations. The horizontal lines represent the TVLA's detection threshold.

focus on the 31 rounds of PRESENT. Given a sampling ratio of 4, this yields a trace length of 124 samples. Thus the new traces have a length of over 7K data points. With the new datasets we then performed second order leakage assessment in the same way as for the univariate case. The results of this experiment are shown in Fig. 7. With this analysis we verify how the protected architectures show no multivariate leakages of second order with 10e6 traces.

We devised an additional experiment in order to demonstrate the effect of the refresh function on the security of the protected architecture. We disabled the PRNG functions and captured 1e6 traces with fixed/random plaintexts and keys. We then used these data



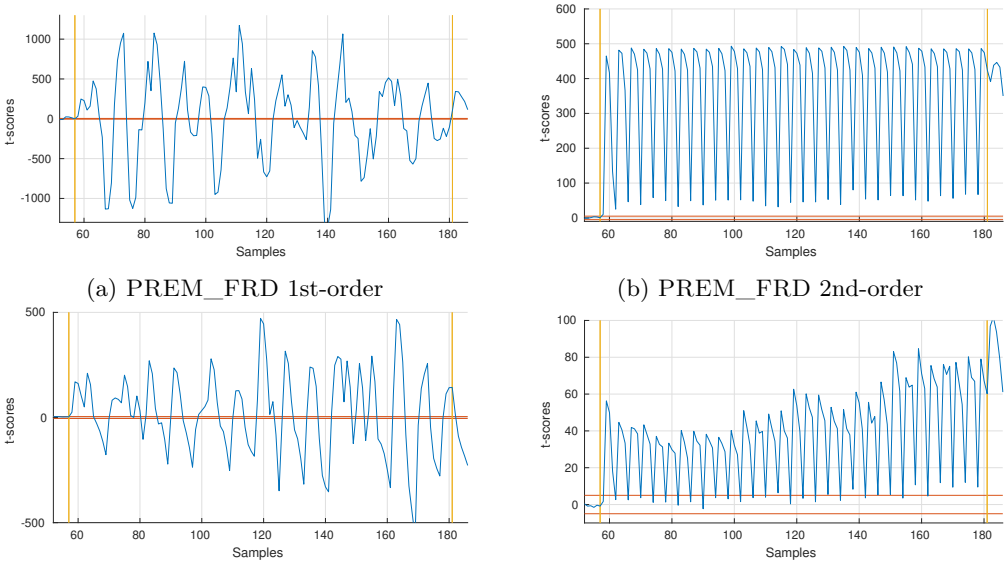
(a) PRE_FRD 2nd-order - 2-variate

(b) PRE_FRK 2nd-order - 2-variate

(c) PREM_FRD 2nd-order - 2-variate

(d) PREM_FRK 2nd-order - 2-variate

Figure 7: Bivariate t-test evaluation of the hardware implementations of PRESENT. These results were derived using 10e6 traces.



(a) PREM_FRD 1st-order

(b) PREM_FRD 2nd-order

(c) PREM_FRK 1st-order

(d) PREM_FRK 2nd-order

Figure 8: Univariate t-test results from the protected PRESENT architecture where the PRNG have been disabled.

to evaluate the different order univariate leakages as previously described. The results are presented in Fig. 8. This analysis demonstrates how without the refresh functions the leakage is significant even with a fraction of the traces on the regular datasets.

7.2 Test of the noiseless output of the shared S-Box

In the previous experiments we have shown that there are no leakages surpassing the detection threshold for the protected architecture of PRESENT. However, there remains the question of wether we have used enough power traces. Or wether the capture setup is sensitive enough. To alleviate these concerns we propose to study the output of the shared function in the noiseless model, through the generation of synthetic traces. To conduct this analysis we implemented the masked S-Box of PRESENT described in Section 6 and feed it with the XOR of the blinded plaintext and key. Following the standard TVLA methodology we alternated randomly between fixed (F) or random (R) plaintexts (D) and used a fixed key value (K). This is equivalent to the FRD test. We generated ten datasets of 1e6 traces each. This allowed us to repeat the experiment ten times to show some repeatability. The masked S-Box under study has 40 outputs which can be combined (by means of an AND function) to mount bivariate and trivariate attacks. However, 4 of these outputs are fixed to zero and thus are not relevant. Thus, our bivariate traces have $\binom{36}{2} = 630$ data points and the trivariate traces have $\binom{36}{3} = 7140$ data points.

In Fig. 9 we provide the univariate results for first, second and third order of protection combining up to three outputs of the masked S-Box. Recall that the masked S-Box offers second degree protection and thus some third order leakages are expected. Furthermore, note how there are also leakages for the trivariate analysis. This is also expected since a combination of three output shares may depend on all the shares of the input. Indeed, in Table 4 we summarize the number of combinations of the S-Box output showing leakages over ten experiments. Furthermore, for the trivariate case the leakages are concentrated for specific combinations of outputs. Some of these leakages are detailed in Table 5.

Table 4: Number of outputs of the masked S-Box (or combinations of them) surpassing the detection threshold (over ten experiments).

Attack	1-variate	2-variate	3-variate
1st-order	0	0	7
2nd-order	0	0	0
3rd-order	67	67	54

Table 5: The three "most frequent output combinations showing leakages" for the 3-variate TVLA. In all cases the 10 experiments surpass the detection threshold. Given $y(i)_j$, let i represent the output bit for the S-Box and let j represent the share index.

Attack	1st leakage	2nd leakage	3rd leakage
1st-order	$y(0)_0 \wedge y(0)_1 \wedge y(0)_2$	$y(0)_0 \wedge y(1)_1 \wedge y(1)_2$	$y(0)_1 \wedge y(0)_2 \wedge y(3)_0$
2nd-order	$y(0)_0 \wedge y(0)_1 \wedge y(0)_2$	$y(0)_0 \wedge y(1)_1 \wedge y(1)_2$	$y(0)_1 \wedge y(0)_2 \wedge y(3)_0$
3rd-order	$y(0)_0 \wedge y(0)_1 \wedge y(0)_2$	$y(0)_1 \wedge y(0)_2 \wedge y(3)_0$	$y(1)_0 \wedge y(1)_1 \wedge y(1)_2$

Observe that in all cases reported in Table 5, the leaky data-points combine shares zero, one, and two of the output bits. Recall that the non-complete set covering used in the masking of the S-Box is $S^{nc}_{(10,3,2)}$), whose structure is reported in Section 4. That is, the output shares are functions of different inputs $s_0 = f_0(x_0, x_1, x_2, x_3, x_6, x_8)$, $s_1 = f_1(x_0, x_1, x_2, x_4, x_7, x_9)$, and $s_2 = f_2(x_0, x_1, x_3, x_4, x_5, x_8)$. It follows that

$$s_0 \wedge s_1 \wedge s_2 = g(x_0, x_1, x_2, x_3, x_4, x_5, x_6, x_7, x_8, x_9)$$

which invalidates the masking at any degree of protection. □

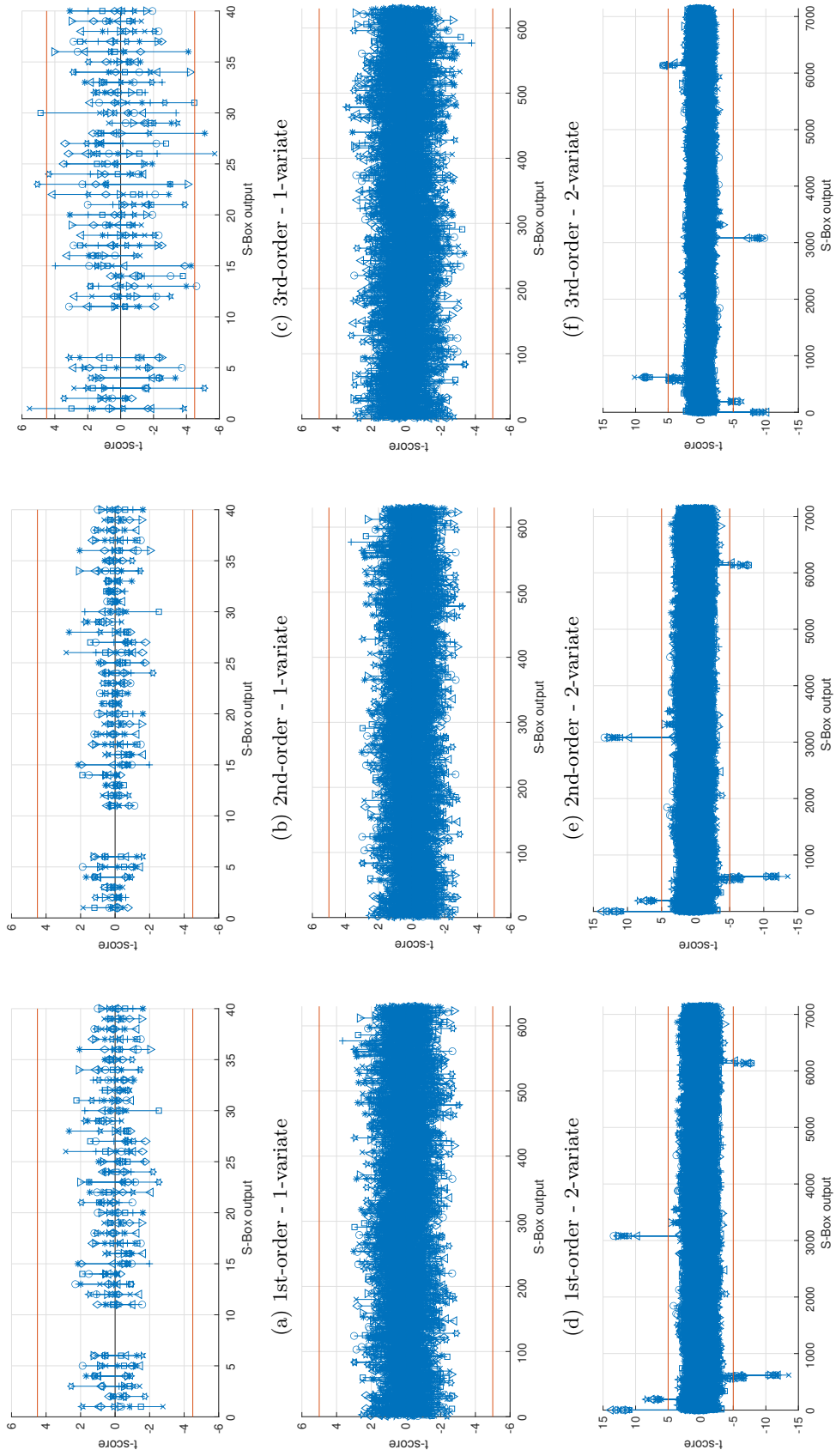
8 Conclusion

In this paper, we have improved the State of the Art on high-order threshold implementations by consolidating the idea of high-order masking from non-complete set coverings. We build on the idea first presented by [Pet19] to explore the feasibility of creating masked implementations with a large number of shares. By allowing more freedom on the allocation of resources we can derive higher degree masked functions which effectively reduce the latency of an implementation. Not only that, but these gadgets have low critical paths by virtue of additional synchronization layers leading to high performance implementations.

As case study, we built a second-order protected implementation of PRESENT and showed that it requires a small amount of resources when framed in the context of modern reconfigurable platforms ($\sim 3\%$ of the CLBs on an Alveo U200 Data Center Acceleration card). We also showed through TVLA of physical and simulated traces that the proposed masking approach is effective up to the d-th order.

9 Acknowledgments

The first author is supported by the French Agence Nationale de la Recherche through the grant ANR-22-CE39-0008 (Project PROPHY). The second author is supported by the project HERMES, funded by the European Union NextGenerationEU/PRTR via INCIBE.



(a) 1st-order - 1-variate (b) 2nd-order - 1-variate (c) 3rd-order - 1-variate

(d) 1st-order - 2-variate (e) 2nd-order - 2-variate (f) 3rd-order - 2-variate

(g) 1st-order - 3-variate (h) 2nd-order - 3-variate (i) 3rd-order - 3-variate

Figure 9: TVLA over combinations of the 40 outputs of the masked S-Box of PRESENT. Note that for the shares that always yield zero the result is null. In each plot the horizontal axis represents the output of the sub-functions (or their combinations). In each case 10 independent experiments are shown, identified by different markers. Horizontal lines represent the detection threshold.

References

[ABP+18] Victor Arribas, Begül Bilgin, George Petrides, Svetla Nikova, and Vincent Rijmen. Rhythmic Keccak: SCA security and low latency in HW. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2018(1):269–290, 2018.

[Alf96] Peter Alfke. Efficient shift registers, LFSR counters, and long pseudo-random sequence generators. Application Note XAPP 052 1.1, AMD-Xilinx, 1996.

[AZN21] Victor Arribas, Zhenda Zhang, and Svetla Nikova. LLTI: low-latency threshold implementations. *IEEE Trans. Inf. Forensics Secur.*, 16:5108–5123, 2021.

[Bar77] Jon Barwise. An introduction to first-order logic. In Jon Barwise, editor, *Handbook of Mathematical Logic*, volume 90 of *Studies in Logic and the Foundations of Mathematics*, pages 5–46. Elsevier, 1977.

[BBD+16] Gilles Barthe, Sonia Belaïd, François Dupressoir, Pierre-Alain Fouque, Benjamin Grégoire, Pierre-Yves Strub, and Rébecca Zucchini. Strong non-interference and type-directed higher-order masking. In Edgar R. Weippl, Stefan Katzenbeisser, Christopher Kruegel, Andrew C. Myers, and Shai Halevi, editors, *ACM CCS 2016: 23rd Conference on Computer and Communications Security*, pages 116–129, Vienna, Austria, October 24–28, 2016. ACM Press.

[BBK+13] Begül Bilgin, Andrey Bogdanov, Miroslav Knežević, Florian Mendel, and Qingju Wang. Fides: Lightweight authenticated cipher with side-channel resistance for constrained hardware. In Guido Bertoni and Jean-Sébastien Coron, editors, *Cryptographic Hardware and Embedded Systems – CHES 2013*, volume 8086 of *Lecture Notes in Computer Science*, pages 142–158, Santa Barbara, CA, USA, August 20–23, 2013. Springer, Heidelberg, Germany.

[BGG+17] Erik Boss, Vincent Grosso, Tim Güneysu, Gregor Leander, Amir Moradi, and Tobias Schneider. Strong 8-bit Sboxes with efficient masking in hardware: extended version. *Journal of Cryptographic Engineering*, 7(2):149–165, June 2017.

[BGN+14] Begül Bilgin, Benedikt Gierlichs, Svetla Nikova, Ventzislav Nikov, and Vincent Rijmen. Higher-order threshold implementations. In Palash Sarkar and Tetsu Iwata, editors, *Advances in Cryptology – ASIACRYPT 2014, Part II*, volume 8874 of *Lecture Notes in Computer Science*, pages 326–343, Kaoshiung, Taiwan, R.O.C., December 7–11, 2014. Springer, Heidelberg, Germany.

[CS21] Gaëtan Cassiers and François-Xavier Standaert. Provably secure hardware masking in the transition- and glitch-robust probing model: Better safe than sorry. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2021(2):136–158, 2021.

[DDF14] Alexandre Duc, Stefan Dziembowski, and Sebastian Faust. Unifying leakage models: From probing attacks to noisy leakage. In Phong Q. Nguyen and Elisabeth Oswald, editors, *Advances in Cryptology – EUROCRYPT 2014*, volume 8441 of *Lecture Notes in Computer Science*, pages 423–440, Copenhagen, Denmark, May 11–15, 2014. Springer, Heidelberg, Germany.

[FGP+18] Sebastian Faust, Vincent Grosso, Santos Merino Del Pozo, Clara Paglialonga, and François-Xavier Standaert. Composable masking schemes in the presence of physical defaults & the robust probing model. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2018(3):89–120, 2018.

[FPS17] Sebastian Faust, Clara Paglialonga, and Tobias Schneider. Amortizing randomness complexity in private circuits. In Tsuyoshi Takagi and Thomas Peyrin, editors, *Advances in Cryptology – ASIACRYPT 2017, Part I*, volume 10624 of *Lecture Notes in Computer Science*, pages 781–810, Hong Kong, China, December 3–7, 2017. Springer, Heidelberg, Germany.

[GMK16] Hannes Groß, Stefan Mangard, and Thomas Korak. Domain-oriented masking: Compact masked hardware implementations with arbitrary protection order. In *Proceedings of the 2016 ACM Workshop on Theory of Implementation Security (TIS@CCS)*, page 3, 2016.

[GP99] Louis Goubin and Jacques Patarin. DES and differential power analysis (the "duplication" method). In Çetin Kaya Koç and Christof Paar, editors, *Cryptographic Hardware and Embedded Systems – CHES'99*, volume 1717 of *Lecture Notes in Computer Science*, pages 158–172, Worcester, Massachusetts, USA, August 12–13, 1999. Springer, Heidelberg, Germany.

[ISW03] Yuval Ishai, Amit Sahai, and David Wagner. Private circuits: Securing hardware against probing attacks. In Dan Boneh, editor, *Advances in Cryptology – CRYPTO 2003*, volume 2729 of *Lecture Notes in Computer Science*, pages 463–481, Santa Barbara, CA, USA, August 17–21, 2003. Springer, Heidelberg, Germany.

[KJJ99] Paul C. Kocher, Joshua Jaffe, and Benjamin Jun. Differential power analysis. In Michael J. Wiener, editor, *Advances in Cryptology – CRYPTO'99*, volume 1666 of *Lecture Notes in Computer Science*, pages 388–397, Santa Barbara, CA, USA, August 15–19, 1999. Springer, Heidelberg, Germany.

[Koc96] Paul C. Kocher. Timing attacks on implementations of Diffie-Hellman, RSA, DSS, and other systems. In Neal Koblitz, editor, *Advances in Cryptology – CRYPTO'96*, volume 1109 of *Lecture Notes in Computer Science*, pages 104–113, Santa Barbara, CA, USA, August 18–22, 1996. Springer, Heidelberg, Germany.

[LNDPMS17] Carlos Andres Lara-Nino, Arturo Diaz-Perez, and Miguel Morales-Sandoval. Lightweight hardware architectures for the PRESENT cipher in FPGA. *IEEE Trans. Circuits Syst. I Regul. Pap.*, 64(9):2544–2555, 2017.

[MMSS19] Thorben Moos, Amir Moradi, Tobias Schneider, and François-Xavier Standaert. Glitch-resistant masking revisited. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2019(2):256–292, 2019.

[MPG05] Stefan Mangard, Thomas Popp, and Berndt M. Gammel. Side-channel leakage of masked CMOS gates. In Alfred Menezes, editor, *Topics in Cryptology – CT-RSA 2005*, volume 3376 of *Lecture Notes in Computer Science*, pages 351–365, San Francisco, CA, USA, February 14–18, 2005. Springer, Heidelberg, Germany.

[MPL+11] Amir Moradi, Axel Poschmann, San Ling, Christof Paar, and Huaxiong Wang. Pushing the limits: A very compact and a threshold implementation

of AES. In Kenneth G. Paterson, editor, *Advances in Cryptology – EUROCRYPT 2011*, volume 6632 of *Lecture Notes in Computer Science*, pages 69–88, Tallinn, Estonia, May 15–19, 2011. Springer, Heidelberg, Germany.

[MPO05] Stefan Mangard, Norbert Pramstaller, and Elisabeth Oswald. Successfully attacking masked AES hardware implementations. In Josyula R. Rao and Berk Sunar, editors, *Cryptographic Hardware and Embedded Systems – CHES 2005*, volume 3659 of *Lecture Notes in Computer Science*, pages 157–171, Edinburgh, UK, August 29 – September 1, 2005. Springer, Heidelberg, Germany.

[NGPM22] Rishub Nagpal, Barbara Gigerl, Robert Primas, and Stefan Mangard. Riding the waves towards generic single-cycle masking in hardware. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2022(4):693–717, 2022.

[NRR06] Svetla Nikova, Christian Rechberger, and Vincent Rijmen. Threshold implementations against side-channel attacks and glitches. In Peng Ning, Sihan Qing, and Ninghui Li, editors, *ICICS 06: 8th International Conference on Information and Communication Security*, volume 4307 of *Lecture Notes in Computer Science*, pages 529–545, Raleigh, NC, USA, December 4–7, 2006. Springer, Heidelberg, Germany.

[NRS09] Svetla Nikova, Vincent Rijmen, and Martin Schläffer. Secure hardware implementation of non-linear functions in the presence of glitches. In Pil Joong Lee and Jung Hee Cheon, editors, *ICISC 08: 11th International Conference on Information Security and Cryptology*, volume 5461 of *Lecture Notes in Computer Science*, pages 218–234, Seoul, Korea, December 3–5, 2009. Springer, Heidelberg, Germany.

[PAB+23] Enrico Piccione, Samuele Andreoli, Lilya Budaghyan, Claude Carlet, Siemen Dhooghe, Svetla Nikova, George Petrides, and Vincent Rijmen. An optimal universal construction for the threshold implementation of bijective S-boxes. *IEEE Trans. Inf. Theory*, 69(10):693–717, 2023.

[Pet19] George Petrides. On non-completeness in threshold implementations. In *Proceedings of the 2019 ACM Workshop on Theory of Implementation Security (TIS@CCS)*, pages 24–28, 2019.

[PMK+11] Axel Poschmann, Amir Moradi, Khoongming Khoo, Chu-Wee Lim, Huaxiong Wang, and San Ling. Side-channel resistant crypto for less than 2,300 GE. *J. Cryptology*, 24(2):322–345, April 2011.

[RBN+15] Oscar Reparaz, Begül Bilgin, Svetla Nikova, Benedikt Gierlichs, and Ingrid Verbauwhede. Consolidating masking schemes. In Rosario Gennaro and Matthew J. B. Robshaw, editors, *Advances in Cryptology – CRYPTO 2015, Part I*, volume 9215 of *Lecture Notes in Computer Science*, pages 764–783, Santa Barbara, CA, USA, August 16–20, 2015. Springer, Heidelberg, Germany.

[Rep15] Oscar Reparaz. A note on the security of higher-order threshold implementations. Cryptology ePrint Archive, Report 2015/001, 2015. https://eprint.iacr.org/2015/001.

[SBB+22] Mateus Simoes, Lilian Bossuet, Nicolas Bruneau, Vincent Grosso, Patrick Haddad, and Thomas Sarno. Self-timed masking: Implementing masked S-Boxes without registers. In *Proceeding of the 21st International Conference*

on Smart Card Research and Advanced Applications (CARDIS), pages 146–164, 2022.

[SBHM20] Pascal Sasdrich, Begül Bilgin, Michael Hutter, and Mark E. Marson. Low-latency hardware masking with application to AES. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2020(2):300–326, 2020.

[SBM18] Pascal Sasdrich, René Bock, and Amir Moradi. Threshold implementation in software - case study of PRESENT. In *Proceedings of the 9th International Workshop on Constructive Side-Channel Analysis and Secure Design (COSADE)*, pages 227–244, 2018.

[Sin05] Carsten Sinz. Towards an optimal CNF encoding of boolean cardinality constraints. In *Proceedings of the 11th International Conference on Principles and Practice of Constraint Programming (CP)*, pages 827–831, 2005.

[SM16] Tobias Schneider and Amir Moradi. Leakage assessment methodology. *J. Cryptogr. Eng.*, 6:85–99, 2016.

[Tse83] Griigori S. Tseitin. *On the Complexity of Derivation in Propositional Calculus*, pages 466–483. Springer, Berlin, Heidelberg, 1983.

[WDM20] Felix Wegener, Lauren De Meyer, and Amir Moradi. Spin me right round rotational symmetry for FPGA-specific AES: Extended version. *Journal of Cryptology*, 33(3):1114–1155, July 2020.

[WM19] Felix Wegener and Amir Moradi. Yet another size record for AES: A first-order SCA secure AES S-box based on GF(2^8) multiplication. In *Proceeding of the 21st International Conference on Smart Card Research and Advanced Applications (CARDIS)*, pages 111–124, 2019.