Optimized Threshold Implementations: Minimizing the Latency of Secure Cryptographic Accelerators

Dušan Božilov[1,2(✉)], Miroslav Knežević[1], and Ventzislav Nikov[1]

[1] NXP Semiconductors, Leuven, Belgium
[2] COSIC KU Leuven and imec, Leuven, Belgium
dusan.bozilov@esat.kuleuven.be

Abstract. Threshold implementations have emerged as one of the most popular masking countermeasures for hardware implementations of cryptographic primitives. In this work, we first provide a generic construction for $d+1$ TI sharing which achieves the minimal number of output shares for any n-input Boolean function of degree $t = n - 1$ and for any d. Secondly, we demonstrate the applicability of our results on a first-order and second-order $d + 1$ low-latency PRINCE implementation.

Keywords: Threshold implementations · PRINCE · SCA · Masking

1 Introduction

Historically, the field of lightweight cryptography focused on algorithm designs occupying smallest possible silicon area. Small area results in low power consumption, another equally important optimization target. However, hitting these two targets degrades performance of lightweight cryptographic primitives, and for most online applications, they frequently do not meet the requirements. Only a handful of designs consider latency among their main design goals. PRINCE [6] and Midori [1] are two prominent examples.

Vulnerability to physical attacks, e.g. side-channel analysis (SCA) is a threat faced by the field of (lightweight) cryptography since its creation, with significant effort being invested in SCA resistant implementation design. To resist an adversary that has access up to d wires in a circuit [11] the secret value has to be shared into at least $d + 1$ random shares using a masking technique, such as Boolean masking.

Circumventing a masked implementation requires attackers to recover the secret information from several shares, i.e. they need to employ a d-th order higher-order attack at least. These attacks are harder to mount due to their susceptibility to measurement noise. Higher-order SCA protection incurs penalties in silicon area, execution time, power consumption and the amount of random bits required for secure execution. Additional cost comes from the increasing number of shares required. The number of output shares grows exponentially

S. Belaïd and T. Güneysu (Eds.): CARDIS 2019, LNCS 11833, pp. 20–39, 2020.
https://doi.org/10.1007/978-3-030-42068-0_2

with the algebraic degree of the function, the number of nonlinear terms the function has, and the security order that needs to be achieved.

Secure cryptographic circuit design becomes significantly harder once the requirements have to be met for latency, energy consumption, silicon area or power. In the context of this paper, and as stated in [12], we consider latency as the total time needed to execute a single cryptographic operation. Minimizing latency can be achieved by increasing the frequency the circuit can operate on or by reducing clock cycle count of the operation. Hence, one design outperforms another with regards to latency if the product of the number of clock cycles and the minimal clock period is smaller in that design.

In [14], the authors provide the first example in the literature where latency and SCA protection are considered as the main design goal. Their results indicate that this is a significantly more difficult problem than designing a countermeasure by optimizing area or the amount of randomness, which are the typical design criteria addressed by the scientific community. Therefore, designing side-channel countermeasures for low-latency or low-energy implementations is considered to be an important open problem. The authors of [8] introduced a generalized concept for low-latency masking that is supposed to be applicable to any implementation and protection order, however they have applied their concept to designs which are not low-latency and therefore it is difficult to compare their approach. We have to stress that the goal to achieve minimal latency is not equivalent to get only execution within less cycles, since at the same time the complexity of the circuit grows resulting in longer critical path. In other words one gets a design which can be executed in less cycles but also with lower max frequency. It has been pointed out in [13] that the generalized concept has to use another re-sharing technique, since the original one has a flaw for $d > 2$.

Threshold Implementations (TI) [15] is a provably secure masking scheme specifically designed to counter side-channel leakage caused by the presence of glitches in hardware. Later the approach of TI was extended to counter higher-order (uni-variate) attacks [3]. The theory suggests the usage of at least $td + 1$ number of input shares in order to make a Boolean function with algebraic degree t secure against a d-th order side-channel attack. That is the reason why these TI schemes are often referred to as a $td + 1$ TI. Consolidated Masking Scheme (CMS) [17] reduced the required number of input shares needed to resist a d-th order attack to $d + 1$, regardless of the algebraic degree of the shared function. Recall that this is theoretically the lowest bound on the number of input shares with respect to the order of security d. After that, many schemes using $d + 1$ shares such as Domain Oriented Masking (DOM) and Unified Masking Approach (UMA) emerged [9,10], where the essential differences among them is in the way the refreshing of the output shares is performed. Since the security against glitches of all these schemes (CMS, DOM, UMA, etc.) relies on the TI principles, these are also referred as $d + 1$ TI.

While the established theory of TI guarantees that the number of input shares linearly grows with the order of protection d, it does not provide efficient means to keep the exponential explosion of the number of output shares under control.

The state-of-the-art is a lower bound of $(d+1)^t$ given in [17], while in [3] the authors described a method to obtain a TI-sharing with $\binom{td+1}{t}$ output shares. The latter work also notes that the number of output shares can sometimes be reduced by using more than $td+1$ input shares. Aside from a formula for the lower bound in [17], there was not much other work of applying $d+1$ TI to functions with higher degree than 2. The only exception is the AES implementations by [19, 20] where $d+1$ TI is applied to the inversion of $GF(2^4)$, which is a function of algebraic degree 3. However, even for this particular case, the first attempt [20] resulted in sharing with minimal number of output shares but it did not satisfy the non-completeness property of TI. Only in the follow-up publication [19] the sharing was correct and minimal. It has to be noted that for the particular case of cubic function, it is fairly easy to find the minimal first-order sharing of 8 output shares by exhaustive trial and error approach.

Our Contribution: In this paper we first introduce a method for optimizing Threshold Implementations. In particular, we provide a constructive solution for $d+1$ TI that achieves the optimal number of output shares for any n-input Boolean function of degree $t = n-1$ for any security order d. Using this construction we demonstrate how to reduce the latency to achieve faster TI-protected implementation of PRINCE. Third, we also show the most energy efficient round-based first-order secure implementation of PRINCE using $d+1$ TI sharing.

Finally, we would like to point out that the method of minimizing the number of output shares is of general interest since it can equally well be applied to any cryptographic implementation and any design optimisation criteria.

2 Preliminaries

The elements of the finite field $\mathcal{F}_2^n$ are represented with small letters. Subscripts are used to specify each bit of an element or each coordinate function of a vectorial Boolean function, for example $x = (x_1, \cdots, x_n)$, where $x_i \in \mathcal{F}_2$. Subscripts are used to represent shares of one-bit variables. The reader should be able to distinguish from the context if the text is referring to specific bits of unshared variable or specific shares of a variable. Next we denote Hamming weight, concatenation, cyclic right shift, right shift, composition, multiplication and addition with $wt(.), ||, \ggg, \gg, \circ, .$ and $+$ respectively. We will use Algebraic Normal Form representations of Boolean functions and will refer to the algebraic degree of such Boolean function.

Two permutations S and S' are affine equivalent if and only if there exists two affine permutations C and D satisfying $S' = C \circ S \circ D$. We refer to C as the output and D as the input transformation. Last the TI sharing which is designed to protect against the d-th order attack we will simply refer to as the d-th order TI.

2.1 Threshold Implementations

The most important property that ensures security of TI even in the presence of glitches is *non-completeness*. The d-th order non-completeness property requires any combination of up to d component functions to be independent of at least one input share. When cascading multiple nonlinear functions, the 1-st order sharing must also satisfy the *uniformity*: namely a sharing is uniform if and only if the sharing of the output preserves the distribution of the unshared output. In other words, for a given unmasked value, all possible combinations of output shares representing that value are equally likely to happen. For higher-order sharing and to achieve uniformity one can always apply refreshing of the output shares.

Given the shares $x_1, \ldots, x_n$ a (first- and second-order) refreshing can be realized by mapping $(x_1, \ldots, x_n)$ to $(y_1, \ldots, y_n)$ using n random values $r_1, \ldots, r_n$ as follows:

$$y_1 = x_1 + r_1 + r_n \qquad y_i = x_i + r_{i-1} + r_i, \qquad i \in \{2, \ldots, n\} \tag{1}$$

This refreshing scheme is called *ring re-masking*. An improvement regarding the number of random bits used when multiplication gate is shared has been achieved in [10] where the amount of randomness required is halved compared to CMS. In [9], the authors have shown that the amount of randomness for sharing a multiplication gate can be further reduced to one third, although this comes at the significant performance cost. Since our goal is to build low-latency side-channel secure implementations, we do not take the approach of UMA. Instead, we choose CMS/DOM for d + 1 TI designs In this paper we will interchangeably use terms mask refreshing and re-masking.

In order to prevent glitch propagation when cascading nonlinear functions, TI requires register(s) to be placed between the nonlinear operations. Otherwise, the non-completeness property may be violated and the leakage of the secret internal state is likely to be manifested.

When sharing a nonlinear function the number of output shares is typically larger than the number of input shares. This is likely to occur when applying $td + 1$ TI and it always occurs when applying $d + 1$ TI. In order to minimize the number of output shares we need to refresh and recombine (compress) some shares by adding several of them together. To prevent glitches from revealing unmasked values, decreasing the number of shares can only be done after storing these output shares into a register. The output shares that are going to be recombined together still need to be carefully chosen such that they do not reveal any unmasked value.

While using $d + 1$ TI the relation between the input shares needs to obey a stronger requirement, namely shared input variables need to be *independent* [17]. This can be achieved in various ways - for example by refreshing some of the inputs or by using a technique proposed in [10].

2.2 Minimizing Implementation Overheads Using S-box Decomposition

Similar to other side channel countermeasures, the area overhead of applying TI increases polynomially with respect to the security order and exponentially with respect to the algebraic degree of the function we are trying to protect. To keep the large overheads caused by exponential dependency under control, designers often use decomposition of the higher degree functions into several lower degree functions. This approach has originally been demonstrated in [16] where the authors implemented a TI-protected PRESENT block cipher [5] by decomposing its cubic S-box into two simpler quadratic S-boxes. Finally, decomposition of all cubic 4-bit S-boxes into chains of smaller quadratic S-boxes was given in [4], which eventually enables compact, side-channel secure implementations.

Although a decomposition of nonlinear functions into several simpler functions of smaller algebraic degree is the proper approach to use for area reduction of the TI-protected implementations, its side-effect is the increased latency of the S-box evaluation and hence the entire implementation. Recall that the TI requires registers to be placed between the nonlinear operations in order to prevent the glitch propagation, which in turn increases the latency. We will not use this approach since our goal is to achieve low-latency.

2.3 A Note on Latency and Energy Efficiency

As already mentioned, most of the effort the scientific community has spent on designing secure implementations has been focused on reducing area overheads. Another important metric that had been given lots of attention is the amount of randomness used in protected implementations. While both of these metrics are important, performance and energy consumption of secure implementations have been unjustly treated as less significant. It has been widely accepted that performance is the metric to sacrifice in order to achieve the lowest possible gate count. Contrary to this view, most of the practical applications nowadays require (very) fast execution and it is often latency of the actual implementation that matters rather than the throughput. Energy consumption is another equally important metric and, unlike power consumption, it cannot be well controlled by keeping the area low while sacrificing performance. Optimizing for energy consumption is in fact one of the most difficult optimization problems in (secure) circuit design since the perfect balance between the circuit power consumption and its execution speed needs to be hit.

The absolute latency is directly proportional to the number of clock cycles a certain operation takes to execute. At the same time, the absolute latency is inversely proportional to the clock frequency the system is running at. While the clock frequency is determined by taking into account multiple factors from the whole system, most important of which is the overall power/energy consumption, the number of clock cycles a certain algorithm takes to execute is under full control of the designer. Especially when considering embedded devices, the tendency is to keep the clock frequency as low as we can while still meeting the

performance requirements. That is the reason why minimizing the number of clock cycles of a certain algorithm is the most important strategy when it comes to minimizing the overall latency of that algorithm.

Although the majority of results available in public literature deal with area-efficient hardware architectures, there are still a few notable examples where the latency reduction has been the main target. In [14], the authors particularly explore the extreme case of a single clock cycle side-channel secure implementations of PRINCE and Midori. Moreover, they conclude that designing a low-latency side-channel secure implementation of cryptographic primitives remains an open problem.

3 Finding an Efficient Sharing

To find a $d+1$ sharing for a quadratic vectorial Boolean function is straightforward and especially easy for the functions that have a simple ANF e. g., a quadratic function with a single second degree term. However, finding an efficient sharing for a vectorial Boolean function of higher algebraic degree with several high degree terms may not be evident, requiring increasingly more effort to find the minimal number of the output shares.

Minimizing the number of output shares becomes even harder the higher the security order d is. In this section we propose methods to deal with this complexity and we describe an optimal solution for the $d+1$ sharing for any security order d.

To achieve d-th order security using $d+1$ sharing for a single term of degree t, i.e. a product of t variables, one gets exactly $(d+1)^t$ shares for the product [17].

For *non-completeness*, in the $d+1$ TI sharing each output share should contain only one share per input variable. In other words if in an output share there are two shares of an input variable then the d-th order non-completeness will be violated. We can see this in the $d+1$ sharing of Eq. (3), the first output share only has one input share of x, y and z: x_0, y_0 and z_0, respectively. All other output shares in Eq. (3) adhere to this rule as well.

Therefore, to ensure *non-completeness* it is enough to have only one share of each input variable present in any given output share. We will assume that the independence of input shares is always satisfied.

Correctness is achieved by verifying that each monomial of a shared term (product) in the unshared function f must be present in one of the output shares.

Consider again the function $xy+z$. One possible first-order $d+1$ sharing of it is given in Eq. (2).

$$\begin{array}{ll} (x, y, z) & \\ (0, 0, 0) & o_1 = x_0 y_0 + z_0 \\ (0, 1, *) & o_2 = x_0 y_1 \\ (1, 0, *) & o_3 = x_1 y_0 \\ (1, 1, 1) & o_4 = x_1 y_1 + z_1 \end{array} \tag{2}$$

Shorter representation of the sharing is shown within the brackets in Eq. (2). Each output share is a row of a table, and each column represents the shares of different input variable. Entry in row i and column j is the allowed input share of j-th input variable for i-th output share.

Columns are representing the variables x, y and z respectively. The asterisk values indicate that we do not care about what input share of z is there, since the sharing of linear term z is ensured by combining rows 1 and 4 of the table. This also shows that the table representation of the sharing does not uniquely determine the exact formula for each output share, and there is certain freedom in determining where we can insert the input shares.

For example, we can use the table of Eq. (2) to share function $x + y + xy + z$. There are two options for terms x_0 and x_1, rows 1 and 2, and rows 3 and 4, respectively. Similarly, y_0 can be in either shares 1 or 3, y_1 can be in share 2 or 4.

Non-completeness and *correctness* can be easily argued from the table representation. Since for every table row, each column entry in the table can represent only one input share of that column's variable, *non-completeness* is automatically satisfied. For row 3 of the table in Eq. (2) by fixing the entries representing x to 1 and y to 0 we ensure that only x_1 and y_0 can occur in that output sharing. Hence, there is no way that x_0 or y_1 can be a part of that particular output share, which is the only way to violate *non-completeness* in $d+1$ sharing. *Correctness* of the table can be verified by checking correctness for every monomial in unshared function f individually. If the combined columns representing variables of the monomial contain all possible combinations of share indexes, sharing is correct, since all terms of shared product for each monomial can be present in the output sharing. Following example from Eq. (2), for monomial xy we see that all four combinations $\{(0,0),(0,1),(1,0),(1,1)\}$ are present in two columns representing variables x and y, allowing all the terms of shared product $xy = (x_0 + x_1)(y_0 + y_1) = x_0 y_0 + x_0 y_1 + x_1 y_0 + x_1 y_1$ to be present in at least one output share. The same holds for $z = z_0 + z_1$ as both combination $\{(0),(1)\}$ are present in output table of Eq. (2). Also, the number of rows in correct sharing table is lower-bounded by the $(d+1)^t$, when the degree of the function is t.

Now, consider a function $xy + xz + yz$. One possible first-order $d+1$ sharing and its table is given in Eq. (3). Columns represent x, y, and z, respectively.

$$
\begin{array}{ll}
(0,0,0) & o_1 = x_0y_0 + x_0z_0 + y_0z_0 \\
(0,1,1) & o_2 = x_0y_1 + x_0z_1 + y_1z_1 \\
(1,0,0) & o_3 = x_1y_0 + x_1z_0 \\
(1,1,1) & o_4 = x_1y_1 + x_1z_1 \\
(*,0,1) & o_5 = \qquad\qquad y_0z_1 \\
(*,1,0) & o_6 = \qquad\qquad y_1z_0
\end{array} \tag{3}
$$

The table has 6 rows representing different output shares, which is larger than theoretically minimal 4 shares. Sharing given by Eq. (3) is also very easily obtained when we try to derive it by hand. Naive approach is to start by sharing xy into four shares. Next, we try to incorporate xz into these four shares by setting all indexes of z to be equal to y. The problem arises when we try to add sharing of yz. In the existing four output shares we have z and y have same indexes, thus we are required to add two more shares for terms y_0z_1 and y_1z_0.

Further on, we will show that for any function with n input variables of degree $t = n - 1$ it is possible to have a $d + 1$ sharing with minimal $(d + 1)^t$ shares.

Definition 1. *Table with n columns representing output sharing of a function of degree t with n input variables is referred to as a D^n-table. The number of rows of the table is the number of output shares for a given sharing. If the output sharing is correct then D^n-table is t-degree correct D^n-table. t-degree correct D^n-table with minimal numbers of rows is called an* optimal *D^n-table. Optimal D^n-table that has $(d+1)^t$ rows is called* ideal *D^n-table, denoted D^n_t-table.*

For $t = n$ ideal D^n_n-table is just a table that contain all different $(d + 1)^t$ indexes of input variables in the terms of shared product that occur when sharing a function of degree t. We can also consider each row of a D^n-table as an ordered tuple of size n. i-th value in a such tuple represents the i-th input variable, and it's value is the allowed input share of that variable in the output share represented by the tuple. All tuple entries have values from the set $\{0, \ldots, d\}$.

Definition 2. *D^t-table D_1 is t-subtable of D^n-table D_2 for given t columns if D_2 reduced to these t columns is equal to D_1.*

We have shown with the sharing in Eq. (2) how one can check the correctness of the table. Now we generalize this by showing how to check if a given D^n-table can be used for sharing of any function of degree t. It is sufficient to check correctness only for the terms of degree t, since if a product of t variables can be shared with a number of output shares, any product of a subset of these t variables can also be shared using the same output shares.

It is easy to see that a D^n-table D can be used to share any function of degree t if and only if for any combination of t columns, D^t-table formed by chosen t columns contains all possible $(d+1)^t$ ordered tuples of size t. In, other words, t-subtable of D for any t columns is t-degree correct D^t-table. This comes from the fact that D^t-table that contains all possible $(d+1)^t$ ordered t-tuples represents

Table 1. D^3-table and its 3 2-subtables.

xyz	xy	xz	yz
000	00	00	00
011	01	01	11
100	10	10	00
111	11	11	11
001	00	01	01
110	11	10	10

a correct sharing for functions of degree t. If this is true for any combination of t columns of D we can correctly share any combination of products of size t from n input variables.

An example is given in Table 1 where D^3-table on the left can be used for first-order sharing of any function of degree 2 since all 3 D^2-tables obtained from it have all 4 possible ordered 2-tuples $(0,0)$, $(0,1)$, $(1,0)$ and $(1,1)$ as at least one of its rows. Next we show how one can construct ideal D^n-table for any function for given n, d and $t = n-1$. To recap, we first build a $(d+1)^t \times n$ table D, where every row is a tuple of indexes (in a single row no variable index is allowed to be missing and, naturally, no variable index is duplicated) and t-subtable of D for any t columns is a t-degree correct D^t-table. Since $t = n-1$ we can consider t-subtable generation as one column removal from D. Such a D^n-table D is then equivalent to a sharing which fulfills the *correctness* and the *non-completeness* properties of TI. Constructing an ideal D^n_n-table is trivial by enumerating all ordered index n-tuples. The number of rows in it is $(d+1)^n$.

Showing that a particular D^n-table with $(d+1)^{n-1}$ rows is a D^n_{n-1}-table becomes equivalent to proving that removal of any single column (restriction to $n-1$ columns or, equivalently, variables) from the D^n-table yields a D^{n-1}_{n-1}-table. Alternatively, any $(n-1)$-subtable of D^n_{n-1}-table is a D^{n-1}_{n-1}-table.

Here we will show how to build the D^n_t-table for the case when $t = n-1$. For any given D^n_{n-1}-table and security order d we will prove the existence of other d D^n_{n-1}-tables such that no n-tuple exists in more than one table. In other words, no two tables contain rows that are equal. We call such $d+1$ D^n_{n-1}-tables *conjugate tables*, and the sharings produced from them *conjugate sharings*. Having all rows different implies that these $d+1$ D^n_{n-1}-tables cover $(d+1)(d+1)^{n-1} = (d+1)^n$ index n-tuples, i.e. all possible index n-tuples. Therefore, these $d+1$ D^n_{n-1}-tables together form a D^n_n-table.

We build the $d+1$ conjugate D^n_{n-1}-tables inductively. For a given d we build $d+1$ conjugate D^2_1-tables, then assuming $d+1$ conjugate D^n_{n-1}-tables exist we construct $d+1$ conjugate D^{n+1}_n-tables.

The *initial step* is simple: D^2_1 has two columns (for the variables x and y) and in each row i (enumerated from 0 to d) of each conjugate table j (enumerated from 0 to d) we set the value in the first column to be i, and the value of the second column to be $(i+j) \mod (d+1)$, hence obtaining the $(d+1)$ conjugate

For $0 \leq i \leq d$ we construct the i-th D_n^{n+1}-table as follows:

1. Start with empty D-table P of $n+1$ variables.
2. For $0 \leq j \leq d$:
   (a) Take the j-th D_{n-1}^n-table and append one more column as last column.
   (b) Fill up the last column with the value $(i+j) \mod (d+1)$.
   (c) The obtained extended table is added to the D-table P.

Fig. 1. Algorithm for optimal $d+1$ sharing

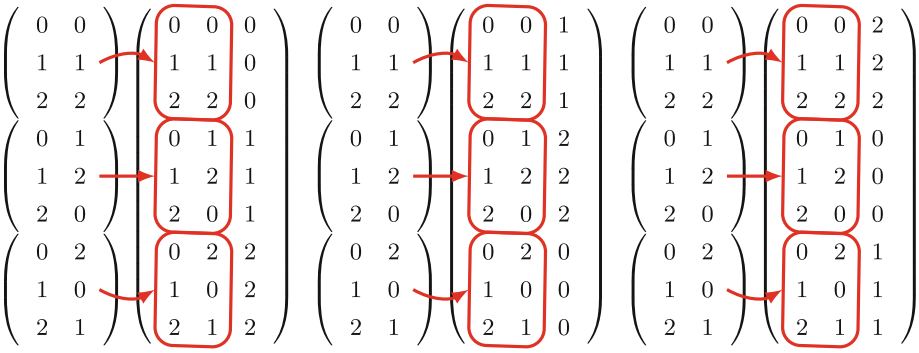

Fig. 2. Generating conjugate D_2^3-tables from D_1^2-tables.

D-tables with $d+1$ rows. Indeed, both columns of any of the constructed D_1^2-tables contain all values between 0 and d, so by removing either column we always obtain a correct D_1^1-table. Also, this construction ensures that second column never has the same index value in one row for different tables, therefore no two rows for different tables are the same, ensuring that formed tables are indeed conjugate.

Induction step - assume we have $d+1$ conjugate D_{n-1}^n-tables. Using them we are now going to build $d+1$ conjugate D_n^{n+1}-tables as described in Fig. 1.

The example of the iterative step from Algorithm 1 is given in Fig. 2.

Lemma 1. *Given $d+1$ conjugate D_{n-1}^n-tables the algorithm described in Fig. 1 constructs $d+1$ conjugate D_n^{n+1}-tables.*

Proof. First, let us show that the constructed $d+1$ D_n^{n+1}-tables are conjugate, i.e. there is no $(n+1)$-tuple which belongs to more than one of them. Let us assume there exists an $(n+1)$-tuple which belongs to two D_n^{n+1}-tables. This implies the existence of an n-tuple which belongs to two of the initial $d+1$ D_{n-1}^n-tables, contradicting the fact that these initial tables are conjugate.

Finally, any restriction to a particular set of columns has to have all the combinations of index n-tuples, i.e. the *correctness* property. In fact, it is sufficient to prove that any set of n columns in any of the new conjugate tables contains all possible n-tuples. Indeed, if we remove the last column in any of the

so constructed tables we get the union of the original $d+1$ D^n_{n-1}-tables forming one D^n_n-table. By definition D^n_n-table satisfies this property. Lastly, we are left with the other case of removing one of the first n columns, which results in a table of dimensions $(d+1)^n \times n$. If we prove there are no duplicates among the $(d+1)^n$ tuples within this table, all combinations will be the table, making it again a D^n_n-table. Consider two n-tuples. If they are equal their last indexes are also equal. By Algorithm 1 design, equality of the last indexes (these are in the $(n+1)$-st column) implies that the two $(n-1)$-tuples belong to one of the starting conjugate D^n_{n-1}-tables, i.e. they can't be in different conjugate D^n_{n-1}-tables. However, for the $(n-1)$-tuples which belong to one of the starting D^n_{n-1}-tables by assumption is known that there are no duplications and hence the considered two $(n-1)$-tuples cannot be equal. □

Theorem 1. *Any of the constructed conjugate D^n_{n-1}-tables by algorithm in Fig. 1 provides optimal sharing for given n, d and $t = n-1$.*

Proof. The algorithm is applied inductively for the number of variables from 2 till n. Since one D^n_{n-1}-table contains exactly $(d+1)^{n-1}$ rows, we conclude it is optimal because this is the theoretical lower bound for the number of output shares for the case $t = n-1$. □

Recall that aside from a formula for the lower bound in [17], there was not much other work of applying $d+1$ TI to functions with higher degree than 2 with the only exception: the AES implementations by [19,20] where $d+1$ TI was applied to the inversion of $GF(2^4)$, which function has algebraic degree 3. When we tried to obtain by hand $d+1$ TI for PRINCE S-box of algebraic degree 3 we only managed to find output sharing for the most significant bit of the S-box with 12 and 44 output shares, for the first-order and the second-order $d+1$ TI prior to the discovery of the Algorithm 1. Optimal solution is 8 and 27 output shares for these two cases, respectively, which is easily found using approach described here.

Another benefit of using algorithmic solution is it can easily be automated using a computer, removing the possibility of human error that is likely to occur, the more complex the ANF becomes.

It is well known that a balanced Boolean function of n variables has a degree at most $n-1$. Therefore all $n \times n$ S-boxes which are permutations have a degree of at most $n-1$. Indeed nearly all bijective S-boxes used in symmetric ciphers are chosen to have a maximum degree of $n-1$. In particular, inversion in the field is always has maximum degree of $n-1$, most notable example of its usage being AES S-box. In the particular case of AES inversion, applying the algorithm shown here will produce the minimal number of shares, which is 128. This is however too large for any practical application.

Most notable exception where low-degree function is used is Keccak's [2] χ-function which is a 5×5 S-box of degree 2. A sharing with 8 shares can be easily found for χ by hand while a conjugate D^5-table will have 16 entries which corresponds to the optimal sharing for degree 4. Hence, the method presented in this section is not optimal when the degree of the function is lower than $n-1$.

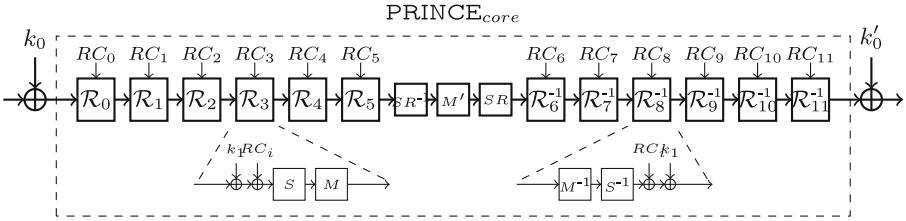

Fig. 3. *PRINCE cipher.*

Therefore, finding the optimal sharing for functions with a degree lower than $n-1$ remains an *open problem.*

4 Hardware Implementation

As a proof of concept we apply the optimal $d+1$ TI to PRINCE [6], a block cipher designed for low-latency hardware implementations. PRINCE block size is 64 bits, with a 128-bit key, used to derived 3 64-bit internally used keys k_0, k'_0 and k_1. Figure 3 shows the internal structure of the cipher consisting of 12 rounds.

PRINCE round consists of 4-bit S-box operation, linear layer realized as matrix multiplication, and round constant addition. The S-box look-up table is $S(x) = [B, F, 3, 2, A, C, 9, 1, 6, 7, 8, 0, E, 5, D, 4]$. The algebraic degree of the S-box is 3, and S-box is affine equivalent to its inverse $S^{-1} = A_{io} \circ S \circ A_{io}$. The A_{io} look-up table is $A_{io}(x) = [5, 7, 6, 4, F, D, C, E, 1, 3, 2, 0, B, 9, 8, A]$.

To implement the first-order secure masking of PRINCE S-box, with $d = 1$, we use the algorithm described in Sect. 3 to obtain a conjugate D_3^4-table. This table represents an optimal solution for 2 input shares with 8 output shares for each input/output bit of the S-box. Recall that the PRINCE S-box is a 4×4-bit S-box and that it has a degree 3.

The optimal sharing is given below in Eq. (4) as conjugate D_3^4-table. The exact sharing for four bits of PRINCE S-box is given with Eqs. (5), (6), (7) and (8), respectively.

$$\begin{array}{ccc} (x,y,z,w) & (1,0,1,0) & (0,1,0,1) \\ (1,1,0,0) & (0,0,1,1) & (1,0,0,1) \\ (0,1,1,0) & (1,1,1,1) & (0,0,0,0) \end{array} \tag{4}$$

As an example consider the first coordinate functions of PRINCE. For the first bit we have $o^1 = 1 + zw + y + yz + wzy + x + xw + xy$ with optimal sharing:

$$
\begin{aligned}
o_1^1 &= 1 + z_0w_0 + y_0 + y_0z_0 + w_0z_0y_0 + x_0 + x_0w_0 + x_0y_0\\
o_2^1 &= \qquad\qquad\qquad\qquad w_0z_0y_1 \qquad\qquad\qquad + x_1y_1\\
o_3^1 &= \quad z_1w_0 \quad + y_1z_1 + w_0z_1y_1\\
o_4^1 &= \qquad\qquad\qquad\qquad w_0z_1y_0 \quad + x_1w_0\\
o_5^1 &= \quad z_1w_1 \quad + y_0z_1 + w_1z_1y_0 \quad + x_0w_1\\
o_6^1 &= \qquad\qquad\qquad\qquad w_1z_1y_1\\
o_7^1 &= \quad z_0w_1 + y_1 + y_1z_0 + w_1z_0y_1 \qquad\qquad + x_0y_1\\
o_8^1 &= \qquad\qquad\qquad\qquad w_1z_0y_0 + x_1 + x_1w_1 + x_1y_0
\end{aligned} \tag{5}
$$

Continuing for the second bit's algebraic function $o^2 = 1+yw+yz+xz+yzw+xyz$ optimal sharing is:

$$
\begin{aligned}
o_1^2 &= 1 + y_0w_0 + y_0z_0 + x_0z_0 + y_0z_0w_0 + x_0y_0z_0\\
o_2^2 &= \qquad\qquad\qquad\qquad y_1z_0w_0 + x_1y_1z_0\\
o_3^2 &= \quad y_1w_0 + y_1z_1 \qquad\quad + y_1z_1w_0 + x_0y_1z_1\\
o_4^2 &= \qquad\qquad\qquad x_1z_1 + y_0z_1w_0 + x_1y_0z_1\\
o_5^2 &= \quad y_0w_1 + y_0z_1 + x_0z_1 + y_0z_1w_1 + x_0y_0z_1\\
o_6^2 &= \qquad\qquad\qquad\qquad y_1z_1w_1 + x_1y_1z_1\\
o_7^2 &= \quad y_1w_1 + y_1z_0 \qquad\quad + y_1z_0w_1 + x_0y_1z_0\\
o_8^2 &= \qquad\qquad\qquad x_1z_0 + y_0z_0w_1 + x_1y_0z_0
\end{aligned} \tag{6}
$$

Optimal sharing for the third bit with algebraic function $o^3 = w + x + zw + xw + xz + xzw + xyz$ is:

$$
\begin{aligned}
o_1^3 &= w_0 + x_0 + z_0w_0 + x_0w_0 + x_0z_0 + x_0z_0w_0 + x_0y_0z_0\\
o_2^3 &= \qquad\qquad\qquad\qquad x_1z_0w_0 + x_1y_1z_0\\
o_3^3 &= \qquad\quad z_1w_0 \qquad\qquad + x_0z_1w_0 + x_0y_1z_1\\
o_4^3 &= \qquad\qquad\qquad x_1w_0 + x_1z_1 + x_1z_1w_0 + x_1y_0z_1\\
o_5^3 &= w_1 \quad + z_1w_1 + x_0w_1 + x_0z_1 + x_0z_1w_1 + x_0y_0z_1\\
o_6^3 &= \qquad\qquad\qquad\qquad x_1z_1w_1 + x_1y_1z_1\\
o_7^3 &= \qquad\quad z_0w_1 \qquad\qquad + x_0z_0w_1 + x_0y_1z_0\\
o_8^3 &= \quad x_1 \qquad + x_1w_1 + x_1z_1 + x_1z_0w_1 + x_1y_0z_0
\end{aligned} \tag{7}
$$

Finally, for the fourth bit of PRINCE S-box and its function $o^4 = 1 + z + x + yz + xy + yzw + xzw + xyw$ optimal sharing is given with:

$$
\begin{aligned}
o_1^4 &= 1 + z_0 + x_0 + y_0z_0 + x_0y_0 + y_0z_0w_0 + x_0z_0w_0 + x_0y_0w_0 \\
o_2^4 &= x_1y_1 + y_1z_0w_0 + x_1z_0w_0 + x_1y_1w_0 \\
o_3^4 &= y_1z_1 + y_1z_1w_0 + x_0z_1w_0 + x_0y_1w_0 \\
o_4^4 &= y_0z_1w_0 + x_1z_1w_0 + x_1y_0w_0 \\
o_5^4 &= z_1 + y_0z_1 + y_0z_1w_1 + x_0z_1w_1 + x_0y_0w_1 \\
o_6^4 &= y_1z_1w_1 + x_1z_1w_1 + x_1y_1w_1 \\
o_7^4 &= y_1z_0 + x_0y_1 + y_1z_0w_1 + x_0z_0w_1 + x_0y_1w_1 \\
o_8^4 &= x_1 + x_1y_0 + y_0z_0w_1 + x_1z_0w_1 + x_1y_0w_1
\end{aligned} \tag{8}
$$

The sharing of the cubic terms is unique while multiple options exist for the sharings of the lower degree terms and that is why one needs to avoid repetitions.

The resharing of the first-order secure implementation is performed according to the DOM [10] rules, in which complementary domains are remasked using the same randomness, with no remasking for output shares containing only one domain. It can be noticed from Eq. 4 that output shares o_1, o_2, o_3, o_4 have complementary domains of shares o_6, o_5, o_8, o_7, respectively. If we consider 8 output shares of 4-bit length, remasking is given with Eq. 9, where o_i, ro_i are S-box outputs output before and after remasking, and r_i are random 4-bit values, requiring 12 random bits. Recombination is achieved by adding shares ro_1, ro_2, ro_3, ro_4 into one, and ro_5, ro_6, ro_7, ro_8 into another recombined share.

$$
\begin{aligned}
ro_1 &= o_1 & ro_2 &= o_2 + r_1 & ro_3 &= o_3 + r_2 & ro_4 &= o_4 + r_3 \\
ro_5 &= o_5 + r_1 & ro_6 &= o_6 & ro_7 &= o_7 + r_3 & ro_8 &= o_8 + r_2
\end{aligned} \tag{9}
$$

If we inspect the PRINCE round structure we can further reduce the first-order randomness requirement. The mixing layer consists of matrices M, M' or M^{-1}, while M can be derived from M' using nibble shuffling SR, i.e. $M = SR \circ M'$. The 64×64 involution matrix M' independently affects 16-bit parts of its input, and can be viewed as 4 independent 16×16 matrices (M_0, M_1, M_1, M_0). PRINCE state composed of 16 nibbles enumerated from 0 to 15 can be separated into 4 groups: $(0, 1, 2, 3)$, $(4, 5, 6, 7)$, $(8, 9, 10, 11)$ and $(12, 13, 14, 15)$. Randomness for the S-boxes can be reused between groups, as the nibble shuffling

$$SR : (0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15) \rightarrow (0, 5, 10, 15, 4, 9, 14, 3, 8, 13, 2, 7, 12, 1, 6, 11)$$
$$SR^{-1} : (0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15) \rightarrow (0, 13, 10, 7, 4, 1, 14, 11, 8, 5, 2, 15, 12, 9, 6, 3)$$

together with M' operation does not cause mixing of the S-box outputs obtained using the same randomness. Additionally, assuming probing model case, the first-order attacker can observe one share out of two at a given cycle, disallowing him to exploit the reuse of randomness. Hence, this structure in round-based implementation reduces the amount of randomness by a factor of four.

The second-order implementation of the PRINCE S-box is again obtained using algorithm explained in Sect. 3. It provides a sharing with 3 input shares

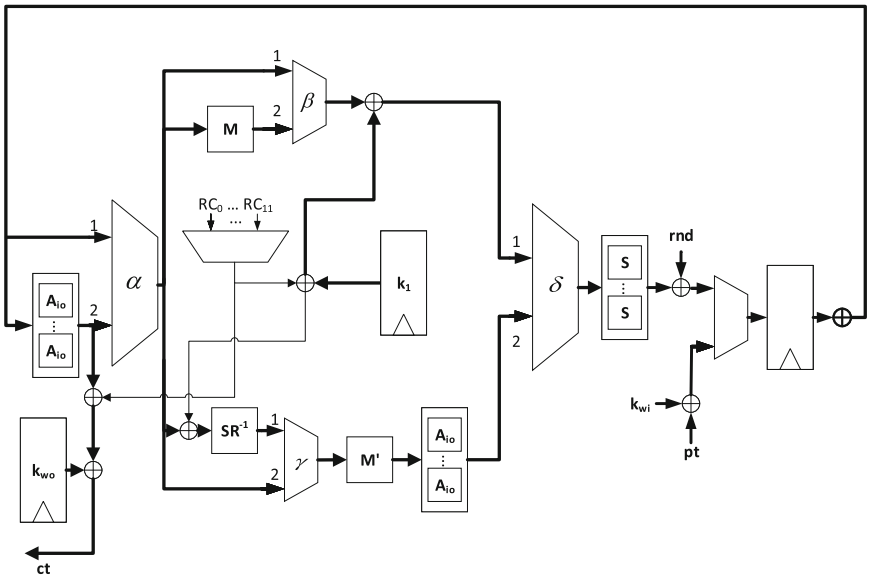

Fig. 4. Protected PRINCE round based architecture with one cycle per round execution.

and 27 output shares. The second order D_3^4-table is given in Eq. 10. Due to space requirements we omit the exact sharing, but a correct sharing can be derived from Eq. 10. For the second-order implementation ring-resharing technique is used, requiring 27 random bits per S-box output bit, or 108 random bits per S-box.

$$
\begin{array}{lllll}
(x,y,z,w) & (0,0,1,1) & (0,0,2,2) & (2,0,2,1) & (2,0,0,2) \\
(1,1,0,0) & (1,1,1,1) & (1,1,2,2) & (0,2,0,1) & (0,2,1,2) \\
(2,2,0,0) & (2,2,1,1) & (2,2,2,2) & (1,0,0,1) & (1,0,1,2) \\
(0,1,1,0) & (0,1,2,1) & (0,1,0,2) & (2,1,0,1) & (2,1,1,2) \\
(1,2,1,0) & (1,2,2,1) & (1,2,0,2) & (2,0,1,0) & (0,2,2,0) \\
(1,0,2,0) & (2,1,2,0) & (0,0,0,0) & &
\end{array}
\tag{10}
$$

Hardware architecture of two $d+1$ TI PRINCE implementations without S-box decomposition is shown in Fig. 4. Control for the two implementations is exactly the same, while datapath only differs in the numbers of shares that are used. First-order implementation has 2 shares throughout, except for the S-box output, that has 8 shares, recombined back to two after the register stage. Second-order implementation has 3 shares, with S-box output having 27 shares.

Table 2. Area/power/energy/randomness/latency/max frequency comparison

PRINCE	Area @10 MHz (GE)	Power @10 MHz (uW)	Energy @10 MHz (pJ)	Rand/Cycle (bits)	Clock # (cycle)	f_{max} (MHz)	Latency @ f_{max} (ns)
Unprotected	3589	59	71	0	12	393	30.5
[14] 1^{st} $(td+1)$ with S-box decomp.	9484	66	264	0	40	432	92.6
1^{st} $(d+1)$ w/o S-box decomp.	11596	100	241	48	24	376	63.8
2^{nd} $(d+1)$ w/o S-box decomp.	32444	374	898	1728	24	385	62.4

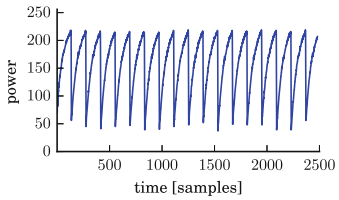

Fig. 5. Example power trace waveform used to perform the t-test on first-order PRINCE.

4.1 Synthesis Results and Side-Channel Evaluation

We have synthesized our designs as well as the previously existing TI PRINCE implementation [14] using TSMC 90 nm library using the typical case of +25 °C. Synthesis tool is Cadence Encounter RTL Compiler version 14.20-s034. Producing the smallest possible implementation was achieved by setting the frequency well below the critical path, at 10 MHz. The power consumption at 10 MHz is averaged from 100 random inputs simulations of a back-annotated post-synthesis netlist, obtained using Cadence Incisive Enterprise Simulator version 15.10.006. Energy is given for one encryption operation, assuming average power consumption. Table 2 shows area, power and energy consumption, the number of random bits required per clock cycle and maximum frequency for 3 hardware implementations, one given by Moradi [14], and two that newly proposed ones. The authors of [14] provided us with their implementations, allowing for a fair comparison of three designs using the same compiler and library, as the synthesis results for design presented in [14] differ from the original paper.

At the maximum frequency, our first-order design surpasses previous state of the art by reducing latency by almost a third. The energy consumption of our first-order at the frequency of 10 MHz is lower by almost 10%. On the other hand, the implementation from [14] beats our version with respect to area, power consumption, maximal running frequency and randomness required. Potentially, it also can achieve higher throughput, with small modifications to the finite state machine, so it processes three messages at once. Given that our goal was to minimize implementation latency and energy, these results are not surprising.

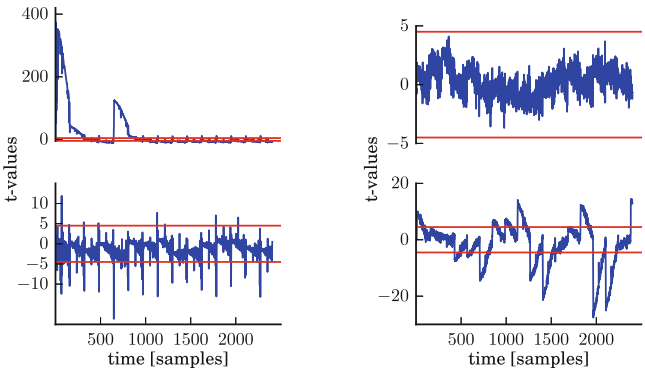

Fig. 6. Leakage detection test results on first-order PRINCE. PRNG off (left) and PRNG on (right). First- (top) and second- (bottom) order t-test results.

We first provide evaluation of the first-order PRINCE without S-box decomposition using optimal $d + 1$ sharing which design was programmed onto a Xilinx Spartan-6 FPGA. The platform used is a Sakura-G board. The design is separated into two FPGAs to minimize the noise: one performs the PRINCE encryption and second FPGA handles the I/O and the start signal. Our core runs at 3.072 MHz while the sampling rate is 500 million samples per second. The power waveform is given in Fig. 5.

We apply a non-specific leakage detection test [7] on the input plaintext following the standard methodology [18], and resulting t-test graphs are shown in the Fig. 6. First, we turn PRNG off to verify validity of the setup and leakage is detected with 1 million traces. The left hand side in Fig. 6 demonstrates a strong first-order leakage during the loading of the plaintext and the key. This can be attributed to one share of both the key and the plaintext being equal to the unshared value, while the other share is zero. Another strong peak is during the first S-box execution as there is still high correlation to the input. Leakage is present in later rounds as well due to lack of additional randomness, although it becomes smaller. Second-order leakage can also be observed when the masks are off. When PRNG is on no first-order leakage is detected after 100 million traces, while second-order leakage is observed as expected.

Due to size and randomness needed, the second-order design did not fit onto the same FPGA board. Instead, the design is tested against simulated power traces. We measured the estimated power consumption by running a post-synthesis simulation with back-annotated netlist. Input-to-output timing delays and current consumption of every gate in the netlist were taken into account and modeled as specified by the technology liberty timing file. In our simulations, one clock cycle is represented with 50 sample points and we cover first seven rounds of the execution. One million traces have been obtained with PRNG switched on, and two thousand traces with PRNG off. Simulated traces are perfectly aligned, they do not contain any measurement noise, and numerical

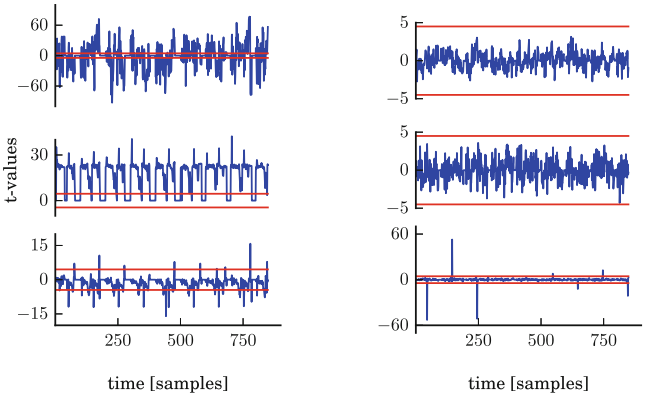

Fig. 7. Leakage detection test results on second-order PRINCE. PRNG off (left) and PRNG on (right). First, second and third-order (top - middle - down) t-test results.

noise of the samples is minimized by having a precision of 32-bit floating point representation compared to 8-bit obtained from the FPGA setup.

The second-order implementation t-test results are shown in Fig. 7. We notice that with PRNG off, leakage occurs in all orders with only two thousand traces. With PRNG on, the design is leakage free in first and second-order, while several points leak in the third order. More precisely, third order leakage occurs during writing of the S-Box output to the register every other cycles.

5 Conclusion and Outlook

In this paper we provided an algorithm which produces a $d + 1$ TI sharing with the optimal (minimum) number of output shares for any n-input Boolean function of degree $t = n - 1$ and for any security order d. We highlight that this contribution is of general interest since the method of minimizing the number of output shares can be applied to any cryptographic design.

Second, we reported, evaluated and compared hardware figures for our proposed TI-protected round-based version of PRINCE cipher, with the previous state of the art. The comparison showed that our designs have more than 30 % lower latency compared to the architecture presented in [14] while the energy consumption is lower by about 10 %. It should, however, be noted that the design presented in [14] still has the highest power efficiency reported in the literature.

We would like to summarize that the generic algorithm for achieving minimal number of output shares is necessary, but not sufficient condition when designing for low-latency and low-energy applications. Applying TI on higher degree functions reduces the total clock count, in effect reducing latency and energy consumed during one operation. However, due to increased circuit complexity it increases the area and the critical path of the design, which have negative impact on energy consumption and latency, respectively. A circuit designer should take

all these parameters into consideration, since the optimal design choice heavily depends on the algorithm in question, alongside the constraints imposed upon the design. In the case of PRINCE block cipher, our work shows that for achieving low-latency it is more efficient not to perform S-box decomposition.

As discussed in [14], designing a low-latency side-channel protection in general, and for PRINCE block cipher in particular, has been identified as an open problem. In this work we have shown the fastest and the most energy efficient round based first-order secure implementation of PRINCE using $d+1$ TI sharing.

Acknowledgements. We would like to thank Amir Moradi and Tobias Schneider for providing us with HDL code of PRINCE TI presented in [14]. Also we would like to thank the reviewers for helping us to improve the paper.

References

1. Banik, S., et al.: Midori: a block cipher for low energy. In: Iwata, T., Cheon, J.H. (eds.) ASIACRYPT 2015. LNCS, vol. 9453, pp. 411–436. Springer, Heidelberg (2015). https://doi.org/10.1007/978-3-662-48800-3_17
2. Bertoni, G., Daemen, J., Peeters, M., Assche, G.V.: The Keccak reference, January 2011. http://keccak.noekeon.org/
3. Bilgin, B., Gierlichs, B., Nikova, S., Nikov, V., Rijmen, V.: Higher-order threshold implementations. In: Sarkar, P., Iwata, T. (eds.) ASIACRYPT 2014. LNCS, vol. 8874, pp. 326–343. Springer, Heidelberg (2014). https://doi.org/10.1007/978-3-662-45608-8_18
4. Bilgin, B., Nikova, S., Nikov, V., Rijmen, V., Stütz, G.: Threshold implementations of all 3×3 and 4×4 S-boxes. In: Prouff, E., Schaumont, P. (eds.) CHES 2012. LNCS, vol. 7428, pp. 76–91. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-33027-8_5
5. Bogdanov, A., et al.: PRESENT: an ultra-lightweight block cipher. In: Paillier, P., Verbauwhede, I. (eds.) CHES 2007. LNCS, vol. 4727, pp. 450–466. Springer, Heidelberg (2007). https://doi.org/10.1007/978-3-540-74735-2_31
6. Borghoff, J., et al.: PRINCE – a low-latency block cipher for pervasive computing applications. In: Wang, X., Sako, K. (eds.) ASIACRYPT 2012. LNCS, vol. 7658, pp. 208–225. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-34961-4_14
7. Cooper, J., DeMulder, E., Goodwill, G., Jaffe, J., Kenworthy, G., Rohatgi, P.: Test Vector Leakage Assessment (TVLA) methodology in practice. In: International Cryptographic Module Conference (2013)
8. Groß, H., Iusupov, R., Bloem, R.: Generic low-latency masking in hardware. IACR Trans. Cryptogr. Hardw. Embed. Syst.-TCHES **2**, 1–21 (2018)
9. Gross, H., Mangard, S.: Reconciling $d + 1$ masking in hardware and software. In: Fischer, W., Homma, N. (eds.) CHES 2017. LNCS, vol. 10529, pp. 115–136. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-66787-4_6
10. Groß, H., Mangard, S., Korak, T.: Domain-oriented masking: compact masked hardware implementations with arbitrary protection order. In: Proceedings of the ACM Workshop on Theory of Implementation Security, TIS@CCS 2016, Vienna, Austria, p. 3, October 2016

11. Ishai, Y., Sahai, A., Wagner, D.: Private circuits: securing hardware against probing attacks. In: Boneh, D. (ed.) CRYPTO 2003. LNCS, vol. 2729, pp. 463–481. Springer, Heidelberg (2003). https://doi.org/10.1007/978-3-540-45146-4_27
12. Knežević, M., Nikov, V., Rombouts, P.: Low-latency encryption – is "Lightweight = Light + Wait"? In: Prouff, E., Schaumont, P. (eds.) CHES 2012. LNCS, vol. 7428, pp. 426–446. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-33027-8_25
13. Moos, T., Moradi, A., Schneider, T., Standaert, F.X.: Glitch-resistant masking revisited - or why proofs in the robust probing model are needed. IACR Trans. Cryptogr. Hardw. Embed. Syst.-TCHES **2**, 256–292 (2019)
14. Moradi, A., Schneider, T.: Side-channel analysis protection and low-latency in action. In: Cheon, J.H., Takagi, T. (eds.) ASIACRYPT 2016. LNCS, vol. 10031, pp. 517–547. Springer, Heidelberg (2016). https://doi.org/10.1007/978-3-662-53887-6_19
15. Nikova, S., Rechberger, C., Rijmen, V.: Threshold Implementations against side-channel attacks and glitches. In: Ning, P., Qing, S., Li, N. (eds.) ICICS 2006. LNCS, vol. 4307, pp. 529–545. Springer, Heidelberg (2006). https://doi.org/10.1007/11935308_38
16. Poschmann, A., Moradi, A., Khoo, K., Lim, C.W., Wang, H., Ling, S.: Side-channel resistant crypto for less than 2,300 GE. J. Cryptol. **24**(2), 322–345 (2011). https://doi.org/10.1007/s00145-010-9086-6
17. Reparaz, O., Bilgin, B., Nikova, S., Gierlichs, B., Verbauwhede, I.: Consolidating masking schemes. In: Gennaro, R., Robshaw, M. (eds.) CRYPTO 2015. LNCS, vol. 9215, pp. 764–783. Springer, Heidelberg (2015). https://doi.org/10.1007/978-3-662-47989-6_37
18. Reparaz, O., Gierlichs, B., Verbauwhede, I.: Fast leakage assessment. In: Fischer, W., Homma, N. (eds.) CHES 2017. LNCS, vol. 10529, pp. 387–399. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-66787-4_19
19. Ueno, R., Homma, N., Aoki, T.: A systematic design of tamper-resistant Galois-field arithmetic circuits based on threshold implementation with (d + 1) input shares. In: 2017 IEEE 47th International Symposium on Multiple-Valued Logic (ISMVL), pp. 136–141 (2017)
20. Ueno, R., Homma, N., Aoki, T.: Toward more efficient DPA-resistant AES hardware architecture based on threshold implementation. In: Guilley, S. (ed.) COSADE 2017. LNCS, vol. 10348, pp. 50–64. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-64647-3_4