Optimized Threshold Implementations: Securing Cryptographic Accelerators for Low-Energy and Low-Latency Applications

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

[1] NXP Semiconductors, Leuven, Belgium
[2] COSIC, KU Leuven and imec, 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 the original version of TI, the number of input shares was dependent on both security order d and algebraic degree of a function t, namely $td+1$. At CRYPTO 2015, a new method was presented yielding to a d-th order secure implementation using $d+1$ input shares. In this work, we first provide a construction for $d+1$ TI sharing which achieves the minimal number of output shares for any n-input Boolean function of degree $t = n-1$. Furthermore, we present a heuristic for minimizing the number of output shares for higher order $td+1$ TI. Finally, we demonstrate the applicability of our results on $d+1$ and $td+1$ TI versions, for first- and second-order secure, low-latency and low-energy implementations of the PRINCE block cipher.

Keywords: Threshold Implementations · PRINCE · SCA · Masking

1 Introduction

Historically, the field of lightweight cryptography has been focused on designing algorithms that leave as small footprint as possible when manufactured in silicon. Small area implicitly results in low power consumption, which is another, equally important, optimization target.

Hitting these two targets comes at a price since the performance and energy consumption of lightweight cryptographic primitives are far from being competitive and, for most online applications, they are often not meeting the requirements. There are only a handful of designs that consider latency and energy consumption among their main design goals. PRINCE [BCG+12] and Midori [BBI+15] are two prominent examples.

The inherent vulnerability to physical attacks is a serious threat which the field of lightweight cryptography has been exposed to since its creation. Side-channel analysis is one of the most powerful examples of such attack. A technique introduced by Kocher et al. [KJJ99] extracts secret information from physical devices using power consumption or electromagnetic radiation during the execution of cryptographic algorithm. A significant effort has ever since been invested in trying to design hardware and software implementations resistant to these types of attacks. To resist an adversary that has access up to d wires inside the circuit [ISW03] the secret value has to be shared into at least $d+1$ random shares using a masking technique. An example of such scheme is Boolean masking where the secret is shared into shares using Boolean addition and the shares are then independently processed in a way that prevents revealing the secret information.

In order to circumvent a masked implementation, attackers need to extract and combine the secret information from several shares, i.e. they need to employ a higher-order attack of degree d at least. These attacks are harder to mount because they are susceptible to the amount of noise collected during the trace acquisition but they still represent a serious threat in practice. Securing against higher-order attacks incurs penalties in silicon area, execution time, power consumption and the amount of random bits required for secure execution. Increased cost comes from the number of additional shares that are required to guarantee security. When protecting a nonlinear function the number of output shares grows exponentially and depends on the number of input shares, the algebraic degree of the function, the number of nonlinear terms the function has, and the security order that needs to be achieved. The challenge of designing secure cryptographic circuits becomes significantly harder once the strict requirements have to be met for at least one of the following metrics: silicon area, latency, power or energy consumption. In [MS16], the authors provide the only example in the literature where latency together with side-channel 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. Threshold Implementations (TI) [NRR06] is a provably secure masking scheme specifically designed to counter side-channel leakage caused by the presence of glitches in hardware. The countermeasure removes the dependency between the number of nonlinear terms and the number of input shares, which is a big advantage over classical masking schemes.

In [BGN$^+$14], the authors extended the approach of TI to counter higher-order attacks. 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 the TI scheme introduced in [BGN$^+$14] is often referred to as a $td + 1$ TI. In 2015, the authors of [RBN$^+$15] proposed a Consolidated Masking Scheme (CMS), and 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. Recently, more schemes using $d + 1$ shares such as Domain Oriented Masking (DOM) and Unified Masking Approach (UMA) emerged [GMK16, GM17], where the essential difference with CMS is in the way the refreshing of the output shares is performed. Since the security of CMS, DOM, and UMA relies on the TI principles, in this paper we refer to all these schemes 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 [RBN$^+$15], while in [BGN$^+$14] 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 [RBN$^+$15], 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 [UHA17b, UHA17a] 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 [UHA17b] 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 [UHA17a] the sharing was correct and minimal. Also, 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 introduce a method for optimizing Threshold Implementations, making the low-latency implementations of side-channel secure designs practical. 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 $td + 1$ TI, we present a heuristic for higher-order protection, that yields a number of output shares significantly lower than $\binom{td+1}{t}$. Additionally, we investigate the energy consumption of different approaches, an important design factor, yet often overlooked in the literature.

To demonstrate the feasibility of our method, we apply the optimized $d + 1$ TI and $td + 1$ TI on the hardware implementation of PRINCE. We demonstrate how to reduce the latency to achieve the fastest known TI protected implementation of PRINCE (round-based TI) and, at the same time, minimize the area penalty of adding side-channel countermeasures. By reducing the area overhead we implicitly decrease the power consumption of our design and by reducing the latency we ensure the implementation becomes energy-efficient.

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

2 Preliminaries

We use small letters to represent elements of the finite field $\mathcal{F}_2^n$. Subscripts are used to specify each bit of an element or each coordinate function of a vectorial Boolean function, i.e. $x = (x_1, \cdots, x_n)$, where $x_i \in \mathcal{F}_2$ and $S(x) = (S_1(x), \cdots, S_m(x))$ where S is defined from $\mathcal{F}_2^n$ to $\mathcal{F}_2^m$ and S_i's are defined from $\mathcal{F}_2^n$ to $\mathcal{F}_2$. We omit subscripts if $n = 1$ or $m = 1$. We use subscripts also to represent shares of one-bit variables. The reader should be able to distinguish from the context if we are referring to specific bits of unshared variable or specific shares of a variable. We denote Hamming weight, concatenation, cyclic right shift, right shift, composition, multiplication and addition with $wt(.)$, $||$, $\ggg$, $\gg$, $\circ$, $.$ and $+$ respectively.

Every Boolean function S can be represented uniquely by its *Algebraic Normal Form* (ANF):

$$S(x) = \sum_{i=(i_1,\ldots,i_n)\in\mathcal{F}_2^n} a_i x_1^{i_1} x_2^{i_2} \cdots x_n^{i_n}. \tag{1}$$

Then, the *algebraic degree* of a Boolean function S is

$$deg(S) = max\{wt(i) : i \in \mathcal{F}_2^n, a_i \neq 0\}. \tag{2}$$

The algebraic degree of a vectorial Boolean function S is equal to the highest algebraic degree of its coordinate functions S_j.

Two permutations S and S' are affine equivalent if and only if there exists 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.

The TI sharing 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 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. Finding a uniform sharing of the given vectorial Boolean function is still an open problem, although several heuristics exist. Fortunately, uniformity can still be achieved by refreshing the output shares when no uniform sharing is available.

Given the shares $x_1, \ldots, x_n$ a 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:

$$y_1 = x_1 + r_1 + r_n \qquad y_i = x_i + r_{i-1} + r_i, \qquad i \in \{2, \ldots, n\} \tag{3}$$

This refreshing scheme is called *ring re-masking* [RBN+15] and is mostly used for the higher-order TI implementations. A simpler refreshing using $n-1$ random values exists especially for the first-order secure implementations as we can re-mask the shares $x_1, \ldots, x_n$ in the following way:

$$y_i = x_i + r_i, \qquad i \in \{1, \ldots, n-1\}, \qquad y_n = x_n + r_1 + \ldots + r_{n-1} \tag{4}$$

An improvement regarding the number of random bits used when multiplication gate is shared has been achieved in [GMK16] where the amount of randomness required is halved compared to CMS.

In [GM17], 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 for d+1 TI designs although a small reduction in area and the amount of randomness used for refreshing might be possible if one uses DOM instead. 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* [RBN+15]. This can be achieved in various ways - for example by refreshing some of the inputs or by using a technique proposed in [GMK16].

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 [PMK+11] where the authors implemented a TI-protected PRESENT block cipher [BKL+07] by decomposing its cubic S-box into two simpler quadratic S-boxes. Finally, decomposition of the cubic 4-bit S-boxes into chains of smaller quadratic S-boxes was given in [BNN+12], which eventually enables compact, side-channel secure, implementations of all 4-bit S-boxes.

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.

2.3 A Note on Latency and Energy Efficiency

As mentioned in the introduction, 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 online 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.

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 [MS16], 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 $td+1$ or $d+1$ sharing for a quadratic vectorial Boolean function is fairly straightforward in general and especially easy for the functions that have a simple ANF e.g., a quadratic function with a single higher degree term. However, to find an efficient sharing for a vectorial Boolean function of algebraic degree three or higher and with several higher degree terms may not be evident and the work required to find the minimal number of the output shares becomes increasingly difficult. In this section we propose methods to deal with this complexity and we explicitly describe an optimal solution for the $d+1$ sharing.

3.1 How to find efficient $td+1$ sharing

To obtain a $td+1$ TI implementation, where t is the algebraic degree of the function and d is the security order, one needs to go through the following two computational phases:

a) The expansion phase in which the shared function f uses $s_{in} \geq td+1$ input shares and results in s_{out} output shares. Output share functions f_i are referred to as component functions.

b) The compression phase in which re-masked s_{out} output shares stored in a register are combined again to s_{in} shares.

This process takes 2 clock cycles except when $s_{in} = s_{out}$. For example, for the first-order TI only the first phase suffices. Additionally, if the TI sharing is uniform, refreshing step can be removed.

The d-th order TI (more specifically, its *non-completeness* property) requires that any combination of up to d component functions f_i is missing at least one share for each of the input variables. The method presented in [BGN+14] demonstrates how to find a sharing with the minimum number of input shares, i.e. $s_{in} = td+1$, which results in $s_{out} = \binom{s_{in}}{t}$ output shares. However, this approach does not guarantee that s_{out} is indeed the theoretical minimum. Even more, there are examples which show that by increasing s_{in} it is possible to decrease s_{out}.

We use s_{out} as a figure of merit since the amount of registers required to store the output shares and the amount of random bits required for refreshing increases with the number of output shares. Further on, we will describe a way to find a $td+1$ sharing with small s_{out}.

For every output share there is a subset of input shares of all variables that are permitted to be part of that share's equation. If we enumerate all s_{in} input shares as values from $\{0, \ldots s_{in}-1\}$, for each output share we can define a subset of allowed input shares that can appear in that output share. We call these sets output share sets, or output sets. Set representation determines maximum degree of the output share component function and is equal to the number of elements in it. Further on, we will refer to a set with k elements as k-set. It should be noted that output sets impose no special restrictions for any particular input variable. Each input variable can have any of the allowed input shares present in the output share.

Equation (5) shows an example of how second-order secure sharing of function $xy+z$ can be obtained using 6 input shares and 7 output shares. An illustration of this sharing is additionally represented by their output share sets on the left.

$$\begin{aligned}
&\{0,1,2\} && o_1 = x_0y_1 + x_1y_0 + x_0y_2 + x_1y_2 + x_2y_1\\
&\{0,3,4\} && o_2 = z_4 + x_4y_4 + x_0y_3 + x_0y_4 + x_3y_4 + x_4y_3\\
&\{1,3,5\} && o_3 = z_3 + x_3y_3 + x_1y_3 + x_3y_1 + x_1y_5 + x_3y_5 + x_5y_3\\
&\{2,4,5\} && o_4 = z_5 + x_5y_5 + x_2y_4 + x_4y_2 + x_2y_5 + x_5y_2 + x_4y_5 + x_5y_4\\
&\{0,1,4\} && o_5 = z_0 + x_0y_0 + x_4y_0 + x_1y_4 + x_4y_1\\
&\{0,1,5\} && o_6 = z_1 + x_1y_1 + x_0y_5 + x_5y_0 + x_5y_1\\
&\{0,2,3\} && o_7 = z_2 + x_2y_2 + x_2y_0 + x_3y_0 + x_2y_3 + x_3y_2
\end{aligned} \tag{5}$$

As it can be seen the output share sets dictate which indexes of variables are allowed in the corresponding output share. For example, for o_1 only input shares (i.e. indexes) 0, 1 and 2 are allowed. That requirement is indeed fulfilled by the formula describing o_1. Note that these sets do not uniquely define the sharing. We could, for example, remove the term x_0y_1 from o_1 and assign it to o_5 or o_6, as $\{0,1\}$ is a subset of both $\{0,1,4\}$ and $\{0,1,5\}$. Output share sets would still be the same and the sharing would still be correct second-order $td+1$ sharing. Good heuristic for assigning monomials to output shares is that all output shares should have about the same amount of terms. That way the implementation becomes balanced and the critical path is roughly equal among all output shares.

We can reason about the properties of a given $td+1$ sharing by inspecting its output sets. Consider a sharing of a function of degree t and the set S_t that contains all $\binom{s_{in}}{t}$ different t-sets of input shares. *Correctness* is satisfied if and only if the set that contains all different t-sets that are subsets of at least one output set is equal to the S_t. Indeed, if that is not the case, there would exist a cross-product of degree t that could not be part of any output share, making correctness of the sharing violated. For Equation (5), given that the unshared function is of degree 2, we can check and confirm that all of the $\binom{6}{2}$ 2-sets are contained in output sets of the sharing. For *non-completeness* property, which ensures d-th order security, no union of d output sets is equal to a set covering all input shares $\{0,\ldots,s_{in}-1\}$. Equivalently, any combination of d output shares does not contain all input shares. In Equation (5) this is true for $d=2$ as no union of two output sets gives the set of all input shares $\{0,1,2,3,4,5\}$.

Interesting question when searching for minimal number of shares is the size of output sets. Since any subset of output shares that contains all possible t-sets also contains all possible sets of smaller length, and smaller output sets do not contribute in generation of correct sharing of degree t function, we can conclude sets of length smaller than t are redundant and can be omitted. On the other hand, the size of each output share set cannot be larger than $s_{in} - (t(d-1)+1)$. Let us assume otherwise, i.e. there is an output set of size at least $s_{in} - t(d-1)$. To cover set of all input shares we need to add missing $t(d-1)$ input shares to it from other output sets. Since the length of output set is at least t, we can do it with at most $(d-1)$ other output sets. But now the *non-completeness* would be violated since there are d output shares which cover all input shares.

Now, we are ready to describe the greedy algorithm (see Figure 1) which finds the sharing with small number of output shares.

Step *2b* in algorithm described in Figure 1 involves randomly choosing k-set o among all possible k-set with the desired property. This nondeterministic behavior leads to different output sharings with different cardinalities. Hence, we iterate the greedy algorithm multiple times and choose the output sharing S_o with the smallest cardinality among all executions. After 100 iterations we have observed that the number of output shares remains the same, regardless of how many times we run Algorithm 1, while the execution time increases. Thus, for the results we present here we have run greedy algorithm 100 times and chosen the smallest sharing among all executions.

The example of a single pass of the greedy algorithm is given in Table 1, for $S_{in}=6$, $d=2$ and $t=2$, making set O containing all sets of size $k = s_{in} - (t(d-1)+1) = 3$. In each step of the greedy algorithm, these sets are scored according to the step *2b*. In bold we have marked the chosen set to be added to the output sharing, and in light gray we highlight the sets that must be removed, because if they remain in the next steps of the algorithm, they could violate

1. For a given number of input shares s_{in}, security order d and algebraic degree t, we work with two sets:
   (a) A set O containing initially all k-sets with $k = s_{in} - (t(d-1)+1)$, with elements from $\{0, \ldots, s_{in}-1\}$.
   (b) An empty solution set S_o, i.e. initially there are no output shares.
2. Then we iterate as follows until the combination of t-sets covered by its output shares S_o contains all different t-sets:
   (a) Remove all k-sets from O that would cause violation of non-completeness when combined with existing k-sets in S_o.
   (b) From set O we pick set o that covers the highest number of t-sets that are not subsets of any output set in S_o. If there are multiple k-sets with this property we randomly choose one of them as o.
   (c) We add o to S_o and remove it from O.
3. The set S_o is the found output sharing s_{out}.

Figure 1: Greedy Algorithm for efficient $td + 1$ sharing

non-completeness of the constructed sharing. Left column in each table is the score of a given k-set, or the amount of t-sets it contains, that are not present in S_o. Right column is all remaining k-sets that do not violate non-completeness if added to S_o. In the same column above the horizontal line is partially constructed S_o represented by k-sets that are added to it. If we take the fourth table, k-set $\{0, 1, 4\}$ has a score of 1 as only $\{1, 4\}$ is the new t-set it would add, given $\{0, 1\}$ and $\{0, 4\}$ are already subsets of output shares $\{0, 1, 2\}$ and $\{0, 3, 4\}$. On the other hand, k-set $\{2, 4, 5\}$ has a score of 3 since none of the t-sets $\{2, 4\}$, $\{2, 5\}$ and $\{4, 5\}$ are present in any of the output shares. For this particular order of the k-sets, we end up with output sharing that contains 7 shares.

3.2 How to find optimal $d + 1$ sharing

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 [RBN+15]. Alternatively, for $s_{in} = d + 1$ input shares and a product of t variables one gets $s_{out} = (d + 1)^t$ output shares.

Main difference with $td + 1$ sharing is how the *non-completeness* property is interpreted. Unlike with $td + 1$ TI sharing 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. Recall that for the $td + 1$ TI using more shares per input variable is possible since the number of input shares is bigger. We observe this in Equations (5) and (7). In the first output share of Equation (5), we see 3 input shares of x: x_0, x_1 and x_2. In $d + 1$ sharing of Equation (7), the first output share only has one input share of x: x_0.

Therefore, for $d + 1$ case 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 for the $d + 1$ case.

Correctness of the sharing in $d + 1$ case 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

Table 1: Example execution of the greedy algorithm for the case $s_{in} = 6$, $d = 2$, $t = 2$. Each table shows a single step of algorithm execution. Left column is the amount of t-sets not covered in S_o by the set on the right. Sets above the horizontal line are partially constructed sharing S_o.

	$\{\mathbf{0},\mathbf{1},\mathbf{2}\}$
3	$\{\mathbf{0},\mathbf{1},\mathbf{2}\}$
3	$\{0,1,3\}$
3	$\{0,1,4\}$
3	$\{0,1,5\}$
3	$\{0,2,3\}$
3	$\{0,2,4\}$
3	$\{0,2,5\}$
3	$\{0,3,4\}$
3	$\{0,3,5\}$
3	$\{0,4,5\}$
3	$\{1,2,3\}$
3	$\{1,2,4\}$
3	$\{1,2,5\}$
3	$\{1,3,4\}$
3	$\{1,3,5\}$
3	$\{1,4,5\}$
3	$\{2,3,4\}$
3	$\{2,3,5\}$
3	$\{2,4,5\}$
3	$\{3,4,5\}$

	$\{0,1,2\}$
	$\{\mathbf{0},\mathbf{3},\mathbf{4}\}$
2	$\{0,1,3\}$
2	$\{0,1,4\}$
2	$\{0,1,5\}$
2	$\{0,2,3\}$
2	$\{0,2,4\}$
2	$\{0,2,5\}$
3	$\{\mathbf{0},\mathbf{3},\mathbf{4}\}$
3	$\{0,3,5\}$
3	$\{0,4,5\}$
2	$\{1,2,3\}$
2	$\{1,2,4\}$
2	$\{1,2,5\}$
3	$\{1,3,4\}$
3	$\{1,3,5\}$
3	$\{1,4,5\}$
3	$\{2,3,4\}$
3	$\{2,3,5\}$
3	$\{2,4,5\}$

	$\{0,1,2\}$
	$\{0,3,4\}$
	$\{\mathbf{1},\mathbf{3},\mathbf{5}\}$
1	$\{0,1,3\}$
1	$\{0,1,4\}$
2	$\{0,1,5\}$
1	$\{0,2,3\}$
1	$\{0,2,4\}$
2	$\{0,2,5\}$
2	$\{0,3,5\}$
2	$\{0,4,5\}$
2	$\{1,2,3\}$
2	$\{1,2,4\}$
2	$\{1,3,4\}$
3	$\{\mathbf{1},\mathbf{3},\mathbf{5}\}$
3	$\{1,4,5\}$
2	$\{2,3,4\}$
3	$\{2,3,5\}$
3	$\{2,4,5\}$

	$\{0,1,2\}$
	$\{0,3,4\}$
	$\{1,3,5\}$
	$\{\mathbf{2},\mathbf{4},\mathbf{5}\}$
0	$\{0,1,3\}$
1	$\{0,1,4\}$
1	$\{0,1,5\}$
1	$\{0,2,3\}$
2	$\{0,2,5\}$
1	$\{0,3,5\}$
2	$\{0,4,5\}$
1	$\{1,2,3\}$
2	$\{1,2,4\}$
1	$\{1,3,4\}$
2	$\{1,4,5\}$
2	$\{2,3,4\}$
2	$\{2,3,5\}$
3	$\{\mathbf{2},\mathbf{4},\mathbf{5}\}$

	$\{0,1,2\}$
	$\{0,3,4\}$
	$\{1,3,5\}$
	$\{2,4,5\}$
	$\{\mathbf{0},\mathbf{1},\mathbf{4}\}$
1	$\{\mathbf{0},\mathbf{1},\mathbf{4}\}$
1	$\{0,1,5\}$
1	$\{0,2,3\}$
1	$\{0,2,5\}$
1	$\{0,3,5\}$
1	$\{0,4,5\}$
1	$\{1,2,3\}$
1	$\{1,2,4\}$
1	$\{1,3,4\}$
1	$\{1,4,5\}$
1	$\{2,3,4\}$
1	$\{2,3,5\}$

	$\{0,1,2\}$
	$\{0,3,4\}$
	$\{1,3,5\}$
	$\{2,4,5\}$
	$\{0,1,4\}$
	$\{\mathbf{0},\mathbf{1},\mathbf{5}\}$
1	$\{\mathbf{0},\mathbf{1},\mathbf{5}\}$
1	$\{0,2,3\}$
1	$\{0,2,5\}$
1	$\{0,3,5\}$
1	$\{0,4,5\}$
1	$\{1,2,3\}$
0	$\{1,2,4\}$
0	$\{1,3,4\}$
0	$\{1,4,5\}$
1	$\{2,3,4\}$

	$\{0,1,2\}$
	$\{0,3,4\}$
	$\{1,3,5\}$
	$\{2,4,5\}$
	$\{0,1,4\}$
	$\{0,1,5\}$
	$\{\mathbf{0},\mathbf{2},\mathbf{3}\}$
1	$\{\mathbf{0},\mathbf{2},\mathbf{3}\}$
0	$\{0,2,5\}$
0	$\{0,3,5\}$
0	$\{0,4,5\}$
1	$\{1,2,3\}$
0	$\{1,2,4\}$
0	$\{1,3,4\}$
0	$\{1,4,5\}$

Equation (6).

$$
\begin{array}{ll}
(x,y,z) & \\
(0,0,0) & o_1 = x_0y_0 + z_0 \\
(0,1,*) & o_2 = x_0y_1 \\
(1,0,*) & o_3 = x_1y_0 \\
(1,1,1) & o_4 = x_1y_1 + z_1
\end{array}
\tag{6}
$$

The sharing can also be represented with a table as shown on the left side of Equation (6). 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. Compared to $td + 1$ set representation, table representation restricts input shares per each variable separately, while output sets impose restriction that was the same for all input variables.

The asterisk values indicate that we do not care about what input share of z is there, because 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 Equation (6) 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. The same holds for terms y_0 and y_1, y_0 can be either in output share 1 or 3, and y_1 can be in output share 2 or 4.

Properties of *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, first-order *non-completeness* is automatically satisfied. For row 3 of the table in Equation (6) 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. Indeed, if this is the case, all terms of shared product for each monomial can be present in the output sharing. Following example from Equation (6), 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. Hence, all of the terms of shared product $xy = (x_0 + x_1)(y_0 + y_1) = x_0y_0 + x_0y_1 + x_1y_0 + x_1y_1$ can 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 Equation (6). Also, it is easy to see that 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 Equation (7) with table entries on the left. Columns represent x, y, and z, respectively.

$$\begin{aligned}
&(0,0,0) \qquad o_1 = x_0y_0{+}x_0z_0{+}y_0z_0 \\
&(0,1,1) \qquad o_2 = x_0y_1{+}x_0z_1{+}y_1z_1 \\
&(1,0,0) \qquad o_3 = x_1y_0{+}x_1z_0 \\
&(1,1,1) \qquad o_4 = x_1y_1{+}x_1z_1 \\
&(*,0,1) \qquad o_5 = \qquad\qquad\quad y_0z_1 \\
&(*,1,0) \qquad o_6 = \qquad\qquad\quad y_1z_0
\end{aligned} \tag{7}$$

Obviously, the table now has 6 rows representing different output shares, which is larger than theoretically minimal 4 shares. Sharing given by Equation (7) 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 now 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

Obviously, 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 can have values from the set $\{0, \dots, d\}$.

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 Equation (6) 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 turns out that it is sufficient to check correctness only for the terms of degree t, since if we are able to share a product of t variables with a number of output shares, we can also always share any product of a subset of these t variables 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 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 2 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.

Table 2: 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

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 column removal from D, since there is exactly one column from it that is missing in the generated t-subtable. As already explained, 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, since all we need to do is to enumerate 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, since D^{n-1}_{n-1}-table contain all possible combinations of $(n-1)$-tuples with no repetitions. 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 easy: 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 D-tables with $d+1$ rows. Indeed, both columns of any of the constructed D^2_1-tables contain all values between 0 and d, so by removing either column we always obtain a correct D^1_1-table, meaning we can share correctly both x and y variables. 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_{n-1}-tables. Using them we are now going to build $d+1$ conjugate D^{n+1}_n-tables in a following manner: The example of the iterative step from Algorithm 2 is given in the Appendix A.

Lemma 1. *Given $d+1$ conjugate D^n_{n-1}-tables the algorithm described in Figure 2 constructs $d+1$ conjugate D^{n+1}_n-tables.*

Proof. First, let us show that the constructed $d+1$ D^{n+1}_n-tables are conjugate, i.e. there is no $(n+1)$-tuple which belongs to more than one of them. Let us then assume on the contrary, that there exists an $(n+1)$-tuple which belongs to two D^{n+1}_n-tables. This, in turn, will imply the existence of an n-tuple which belongs to two of the initial $d+1$ D^n_{n-1}-tables. This now leads to a contradiction of the fact that these initial tables are conjugate.

```
For $0 \le i \le d$ 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 $0 \le j \le d$:
      (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 $P$.
```

Figure 2: Algorithm for optimal $d+1$ sharing

The only remaining property which needs to be proven is that 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-1}^n-tables, that form 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 $(d+1)^n \times n$. To show that we have all the combinations of tuples it is sufficient to point out that, since this table contains $(d+1)^n$ tuples and there are no duplications, all combinations are in the table, i.e. this is again a D_n^n-table. Consider two n-tuples and for each of them split on $(n-1)$-tuple of the first $n-1$ indexes and the last index as a separate value. The two n-tuples are now equal if the two $(n-1)$-tuples are equal and the equality of the last indexes is ensured. By Algorithm 2 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-1}^n-tables, i.e. they can't be in different conjugate D_{n-1}^n-tables. However, for the $(n-1)$-tuples which belong to one of the starting D_{n-1}^n-tables by assumption is known that there are no duplications and hence the considered two $(n-1)$-tuples cannot be equal. The latter finishes the proof. □

Theorem 1. *Any of the constructed conjugate D_{n-1}^n-tables by algorithm in Figure 2 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-1}^n-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 [RBN+15], 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 [UHA17b, UHA17a] 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 2. 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 $n-1$. Most notable exception is Keccak's [BDPA11] 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 $n-1$.

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

4 PRINCE

As a proof of concept we apply different flavors of TI to PRINCE [BCG+12], a low-latency block cipher designed to be very efficient in hardware when implemented in an unrolled manner. Its α-reflection property allows a reuse of the same circuitry for both encryption and decryption.

Although not designed to be efficient in software, a bit-sliced software implementation of PRINCE is also fast and can even be executed in fewer clock cycles than other lightweight block ciphers such as PRESENT and KATAN, for example [Pap14].

Here we give a brief overview of PRINCE and for more detailed explanation of the cipher we refer the reader to the original paper [BCG+12]. The block size is 64 bits and a key has a length of 128 bits. The key is split into two 64-bit parts $k_0||k_1$ and expanded to $k_0||k_0'||k_1$ as shown below.

$$(k_0||k_0'||k_1) = k_0||((k_0 \ggg 1) + (k_0 \gg 63))||k_1)$$

As depicted in Figure 3, k_0 and k_0' are used as whitening keys at the start and at the end of the cipher; k_1 is used as round key in PRINCE$_{core}$ which consists of 12 rounds. More precisely, 6 rounds followed by the middle involution layer which is then followed by the 6 inverse rounds.

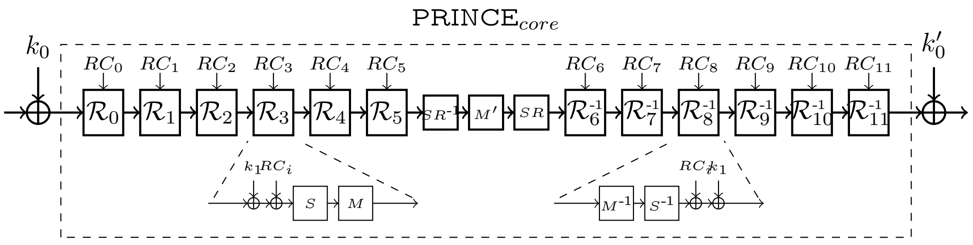
PRINCE$_{core}$
k_0
k_0'
RC_0 RC_1 RC_2 RC_3 RC_4 RC_5
RC_6 RC_7 RC_8 RC_9 RC_{10} RC_{11}
$\mathcal{R}_0$ $\mathcal{R}_1$ $\mathcal{R}_2$ $\mathcal{R}_3$ $\mathcal{R}_4$ $\mathcal{R}_5$
SR^{-1} M' SR
$\mathcal{R}_6^{-1}$ $\mathcal{R}_7^{-1}$ $\mathcal{R}_8^{-1}$ $\mathcal{R}_9^{-1}$ $\mathcal{R}_{10}^{-1}$ $\mathcal{R}_{11}^{-1}$
k_1 RC_i
S M
M^{-1} S^{-1}
RC_i k_1

Figure 3: *PRINCE cipher.*

S-box layer. The S-box is a 4-bit permutation of algebraic degree 3 and its look-up table is $S(x) = [B, F, 3, 2, A, C, 9, 1, 6, 7, 8, 0, E, 5, D, 4]$. The S-box inverse is in the same affine equivalence class as the S-box itself. Moreover, input and output transformations are the same:

$$S^{-1} = A_{io} \circ S \circ A_{io} \tag{8}$$

The affine transformation A_{io} is given as $A_{io}(x) = [5, 7, 6, 4, F, D, C, E, 1, 3, 2, 0, B, 9, 8, A]$.

Linear layer. Matrices M and M' define the diffusion layer of PRINCE. M' is an involution and M matrix can be obtained from M' by adding a shift-rows operation SR so that $M = SR \circ M'$. Recall that SR is a linear operation that permutes the nibbles of the PRINCE state.

RC_i **addition** is a 64-bit round constant addition. The round constants $RC_0 \ldots RC_{11}$ are chosen such that $RC_i + RC_{11-i} = \alpha$ where α is a 64-bit constant. This property, called α-reflection property, together with the construction of PRINCE$_{core}$ rounds makes the decryption of PRINCE$_{core}$ same as the encryption with $k_1 + \alpha$ key.

4.1 S-box decomposition

PRINCE S-box has an algebraic degree three and belongs to class $\mathcal{C}_{131}$ [BNN+12]. According to [BNN+12] and the tables given in [Nik12] there are several hundreds of decompositions into three quadratic S-boxes and four affine transformations.

We choose a decomposition where all 3 quadratic S-boxes are the same, belonging to class Q_{294}, for its small area footprint. Decomposition leads to lower area and randomness requirement as they depend on the algebraic degree of the function when applying TI. On the other hand, performance is penalized. PRINCE S-box ANF $(o_1, o_2, o_3, o_4) = F(x, y, z, w)$ is given by:

$$\begin{aligned} o_1 &= 1 + wz + y + zy + wzy + x + wx + yx \\ o_2 &= 1 + wy + zy + wzy + zx + zyx \\ o_3 &= w + wz + x + wx + zx + wzx + zyx \\ o_4 &= 1 + z + zy + wzy + x + wzx + yx + wyx \end{aligned} \tag{9}$$

S-box and its inverse decompositions used in our implementation are given in Equation (10).

$$\begin{aligned} S &= A_1 \circ Q_{294} \circ A_2 \circ Q_{294} \circ A_3 \circ Q_{294} \circ A_4 \\ S^{-1} &= A_5 \circ Q_{294} \circ A_2 \circ Q_{294} \circ A_3 \circ Q_{294} \circ A_6 \end{aligned} \tag{10}$$

Here A_1 to A_6 are affine transformations and their respective look-up tables are:
$A_1(x) = [C, E, 7, 5, 8, A, 3, 1, 4, 6, F, D, 0, 2, B, 9]$, $A_2(x) = [6, D, 9, 2, 5, E, A, 1, B, 0, 4, F, 8, 3, 7, C]$, $A_3(x) = [0, 8, 4, C, 2, A, 6, E, 1, 9, 5, D, 3, B, 7, F]$, $A_4(x) = [A, 1, 0, B, 2, 9, 8, 3, 4, F, E, 5, C, 7, 6, D]$, $A_5(x) = [B, 8, E, D, 1, 2, 4, 7, F, C, A, 9, 5, 6, 0, 3]$, $A_6(x) = [9, 3, 8, 2, D, 7, C, 6, 1, B, 0, A, 5, F, 4, E]$.

The ANF of the Q_{294} function, $(x, y, z, w) = F(a, b, c, d)$, is given in Equation (11)

$$\begin{aligned} x &= a \\ y &= b \\ z &= ab + c \\ w &= ac + d \end{aligned} \tag{11}$$

We recall that, for a secure implementation with this decomposition method, nonlinear operations need to be separated by registers, making evaluation of a single S-box take 3 clock cycles.

As discussed in Section 2.2, we explore the implementation of the decomposed S-box in order to address the issue of low-area and low-power applications but, in what follows, we also explore the sharing of the non-decomposed S-box to address the issue of low latency and low energy.

5 Implementations

5.1 Unprotected implementation

Figure 4 represents the architecture of unprotected round-based PRINCE. Here we evaluate the S-box in one cycle. One encryption is performed in 12 clock cycles. We utilize the fact that the S-box and its inverse are in the same equivalence class to reuse the same circuitry for both first and last rounds, with the exception that affine transformation circuits A_{io} are used during the evaluation of S^{-1}. By adding an extra multiplexer we can perform decryption as well. To minimize the overhead of adding decryption, we use the round counter design as explained in [MS16].

Following Figure 4, when evaluating the S-box, the data travels through multiplexers $\alpha_1 - \beta_2 - \delta_1$, except in the first round where the path is $\alpha_1 - \beta_1 - \delta_1$. Similarly, when evaluating the inverse S-box, active path is $\alpha_2 - \gamma_1 - \delta_2$, except in the last round where we use $\alpha_1 - \gamma_2 - \delta_2$.

Next, in Sections 5.2-5.6 we first present the TIs of the S-box and its building block Q_{294} and then in Section 5.7 we present the actual secure implementations of PRINCE.

5.2 TIs of Q_{294}

We have implemented $td + 1$ and $d + 1$ variants of TI for both the first- and the second-order Q_{294} implementations . We use the first-order $td + 1$ direct TI sharing [BNN+12], the second-order $td + 1$ sharing with 5 input shares and 10 output shares, as explained in [BGN+14]. For the $d + 1$ first- and second-order implementations, we used sharing given in [RBN+15]. Compression is applied to the second-order $td + 1$ and both $d + 1$ versions. Detailed description of the hardware architecture can be found in the Appendix B.

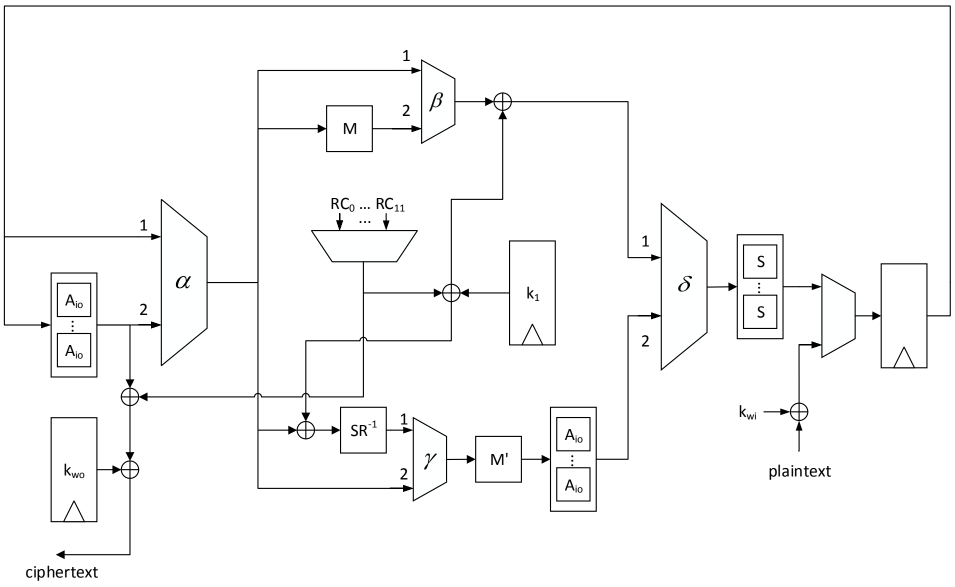

Figure 4: Unprotected PRINCE round based architecture.

5.3 First-order secure $td + 1$ TI of PRINCE S-box

For the first-order $td + 1$ design, we generated sharing with 4 input and output shares following [BNN+12].

Output bits are refreshed using Equation (4), requiring 12 random bits per S-box. The exact sharing does not fit within the margins of this paper and will be provided as a Verilog code online.

5.4 Second-order secure $td + 1$ TI of PRINCE S-box

To create a second-order secure masking for the PRINCE S-box with $d = 2$ we have used the iterated greedy algorithm described in Section 3.1. This algorithm provides a solution that has 17 output shares and 8 input shares. Compared to the solution given in [BGN+14], which had 35 output shares and 7 input shares, we have reduced the total number of shares by almost a half. All output bits are refreshed using the ring re-masking from Equation (3), requiring 68 random bits per S-box. Since the rest of the PRINCE core is using three shares (see Section 5.7), we generate five extra shares before the S-box input which consumes 20 random bits extra. Therefore, the whole S-box evaluation uses 88 random bits. Again, due to the limited space in this paper, the exact sharing will be provided as an online Verilog code.

5.5 First-order secure $d + 1$ TI of PRINCE S-box

To implement the first-order secure masking of PRINCE S-box, with $d = 1$, we use the algorithm described in Section 3.2 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.

All output bits are refreshed using the mask refreshing as given in Equation (4), requiring 7 bits of randomness per output bit, or 28 bits per S-box in total. The optimal sharing is given below in Equation (12) as conjugate D_3^4-table. The exact sharing will again be provided as an online

Verilog code.

$$\begin{array}{c}(x,y,z,w)\\(0,0,0,0)\\(1,1,0,0)\\(0,1,1,0)\\(1,0,1,0)\\(0,0,1,1)\\(1,1,1,1)\\(0,1,0,1)\\(1,0,0,1)\end{array} \tag{12}$$

As an example consider the first coordinate function of PRINCE as given in Equation (9): $o = 1 \oplus wz \oplus y \oplus zy \oplus wzy \oplus x \oplus wx \oplus yx$. Then the optimal sharing is obtained as follows:

$$\begin{aligned}
o_1 &= 1 + w_0z_0 + y_0 + z_0y_0 + w_0z_0y_0 + x_0 + w_0x_0 + y_0x_0\\
o_2 &= y_1 + z_0y_1 + w_0z_0y_1 + x_1 + w_0x_1 + y_1x_1\\
o_3 &= w_0z_1 + z_1y_1 + w_0z_1y_1 + y_1x_0\\
o_4 &= z_1y_0 + w_0z_1y_0 + y_0x_1\\
o_5 &= w_1z_1 + w_1z_1y_0 + w_1x_0\\
o_6 &= w_1z_1y_1 + w_1x_1\\
o_7 &= w_1z_0 + w_1z_0y_1\\
o_8 &= w_1z_0y_0
\end{aligned} \tag{13}$$

Note that the sharing of the cubic terms is unique while there are more options for the sharings of the lower degree terms and that is why one needs to avoid repetitions.

5.6 Second-order secure $d + 1$ TI of PRINCE S-box

For the second-order secure masking of PRINCE S-box, with $d = 2$, we again use the algorithm described in Section 3.2 to obtain conjugate D_3^4-table. This table represents an optimal solution with 3 input shares and 27 output shares for each input/output bit of the S-box. All output bits are refreshed using the ring re-masking as given in Equation (3), requiring 108 random bits for the whole S-box. The optimal sharing is given below in Equation (14) as a conjugate D_3^4-table, where the same rules are used as in the previous section for $d = 1$ case. The exact sharing will be provided as an online Verilog code.

$$\begin{array}{ccc}(x,y,z,w) & & \\(0,0,0,0) & (0,0,1,1) & (0,0,2,2)\\(1,1,0,0) & (1,1,1,1) & (1,1,2,2)\\(2,2,0,0) & (2,2,1,1) & (2,2,2,2)\\(0,1,1,0) & (0,1,2,1) & (0,1,0,2)\\(1,2,1,0) & (1,2,2,1) & (1,2,0,2)\\(2,0,1,0) & (2,0,2,1) & (2,0,0,2)\\(0,2,2,0) & (0,2,0,1) & (0,2,1,2)\\(1,0,2,0) & (1,0,0,1) & (1,0,1,2)\\(2,1,2,0) & (2,1,0,1) & (2,1,1,2)\end{array} \tag{14}$$

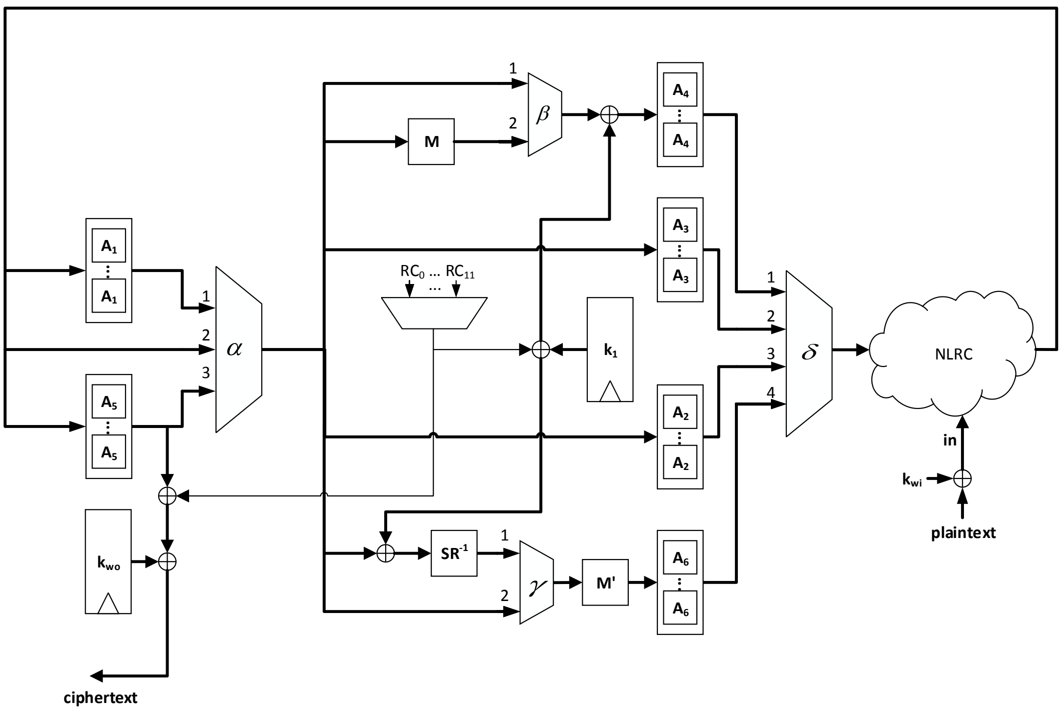

Figure 5: TI PRINCE round based architecture with decomposition.

5.7 Protected implementations of the whole PRINCE cipher

Figure 5 depicts the data path of hardware implementation for the four protected round-based implementations of PRINCE which use the S-box decomposition. All the data lines have the width of $64 \times s$ bits, where s is number of input shares. The only exception is the RC constant output, which has the width of 64 bits. The sharing of the nonlinear layer, followed by the linear layer, re-masking and compression layer is denoted with the NLRC block in Figure 5. Hardware implementations of NLRC layers of Q_{294} are discussed in Appendix B.

In order to support both encryption and decryption, input and output whitening keys, k_{wi} and k_{wo} are either k_0 or k_0', depending on what is being executed. We only require one extra multiplexer to implement this feature. When evaluating the S-box, the data path of the multiplexers is $\alpha_1 - \beta_2 - \delta_1$ in the first, $\alpha_2 - \delta_2$ in the second, and $\alpha_2 - \delta_3$ in the third clock cycle, except in the first round where the third cycle path is $\alpha_1 - \beta_1 - \delta_1$. Similarly, when evaluating the inverse S-box, the active inputs of multiplexers are $\alpha_3 - \gamma_1 - \delta_4$ in the first, $\alpha_2 - \delta_2$ in the second, and $\alpha_2 - \delta_3$ in the third clock cycle, except in the last round where the path during the third cycle is $\alpha_2 - \gamma_2 - \delta_4$.

For $td + 1$ implementations we use 3 and 5 shares respectively for the affine operations in order to reduce the amount of randomness required for the execution. This incurs additional penalty in area occupied by the implementation. Recall that the output of the S-box component functions for $td+1$ TI is shared with 3 and 10 shares respectively for the first- and the second-order secure implementations. Re-masking and compression are done only for the second-order $td + 1$ TI. The $d + 1$ implementations use 2 and 3 shares for the first- and the second-order secure implementation, respectively. The output of the S-box component functions is shared with 4 and 9 shares respectively for the first- and the second-order secure implementations. Re-masking and compression are required in both cases.

The round constant is added to only one of the shares. The key is shared with the same number of shares as the plaintext. Unlike most of the secure implementations available in the literature, in this paper we focus on the round based implementation instead of the serialized one. This greatly reduces the execution time, at the expense of increased area and the required amount of randomness

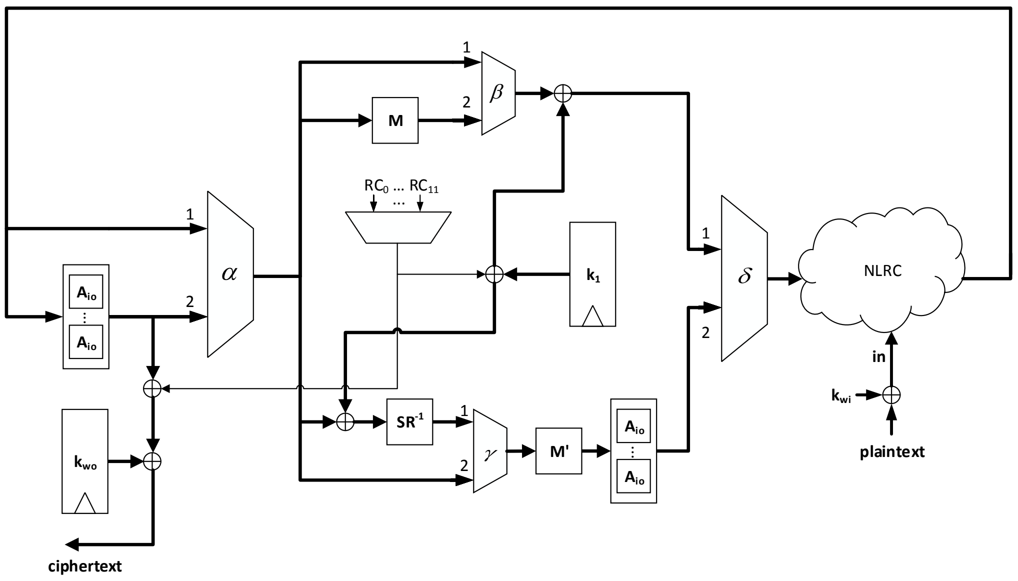

Figure 6: TI PRINCE round based architecture without decomposition.

per clock cycle. In order to decrease area, we employ multiplexers to avoid instantiating additional registers for the 3 stages of the S-box evaluation. Since PRINCE has 12 rounds and each with S-box evaluations has 3 (for $d = 1$ and $td + 1$ TI) or $2 * 3$ stages (for $d + 1$ TI and $td + 1$ TI but $d > 1$) so the total execution takes 36 or 72 clock cycles.

Figure 6 represents the architecture for the four protected round-based implementations of PRINCE without S-box decomposition. The architecture is almost the same as for the unprotected design. The implementations of NLRC layers of PRINCE S-box are discussed in Sections 5.3-5.6.

When evaluating the S-box, the data path of the multiplexers is $\alpha_1 - \beta_2 - \delta_1$ except in the first round where the path in the first clock cycle is $\alpha_1 - \beta_1 - \delta_1$. Similarly, when evaluating the inverse S-box, the active inputs of multiplexers are $\alpha_2 - \gamma_1 - \delta_2$, except in the first cycle of the last round where the path is $\alpha_1 - \gamma_2 - \delta_2$. Unlike in the unprotected version the S-box evaluation takes two cycles (S-box layer and linear layer are separated into two cycles), hence it takes 24 cycles for one encryption/decryption operation. The exception is the first-order $td + 1$ implementation where S-box evaluation takes one cycle, making encryption/decryption latency 12 cycles.

For $td + 1$ implementations we use 4 and 3 shares respectively for the affine operations. Recall that the output of the S-box component functions for $td + 1$ TI is shared with 4 and 17 shares respectively for the first- and the second-order secure implementations. Compression is required only for the second-order $td + 1$ implementation, while re-masking is applied for both of them. The $d + 1$ implementations use 2 and 3 shares for the first- and the second-order secure implementation, respectively. Recall that the output of the S-box component functions is shared with 8 and 27 shares respectively. Re-masking and compression are required in both cases.

5.8 Randomness reduction

Let us take a closer look at the PRINCE round structure. As explained in Section 4 the mixing layer consists of matrices M, M' or M^{-1}. Recall that M can be obtained from M' using nibble shuffling operation SR, i.e. $M = SR \circ M'$. The involution 64×64 matrix M' is constructed as block diagonal matrix with entires (M_0, M_1, M_1, M_0) where M_0 and M_1 are 16×16 matrices.

This structure implies that 16-bit chunks of the state are processed independently. Therefore, we can use the same randomness for all four 16-bit blocks for the attacker case of $d = 1$ and $d = 2$.

Namely, assuming the PRINCE state is composed of 16 nibbles enumerated from 0 to 15 then the following 4 groups can be formed $(0, 1, 2, 3)$, $(4, 5, 6, 7)$, $(8, 9, 10, 11)$ and $(12, 13, 14, 15)$. When evaluating the S-boxes in a given group the randomness required can be reused for the evaluation of the S-boxes in the other groups. It can also be observed that the nibble shuffling

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

does not cause mixing of the S-boxes outputs obtained with the same randomness. Hence, using this structure in round-based implementation reduces the amount randomness by a factor of four. Although the probing model does not accurately reflect the HW specifics like glitches and the parallelism of the implementations, note the first increases the attacker capabilities while the second diminishes it, below we will still use this attacker model in the argumentations.

When we consider first-order attacker he can probe one share out of two at a given cycle, thus the reuse of randomness is not exploitable. For the case of second-order attacker he is able to get either a) 2 shares out of 3 of one nibble or b) 1 share of the 2 nibbles using the same randomness at a given cycle. In case of a) again the attacker cannot exploit the reuse of randomness since he does not know anything about this second value which can be combined with his own knowledge for the first value. In the case of b) the attacker is unable to mount a bivariate attack using points from different rounds (and hence cycles) due to the re-masking after each operation and the key addition (all of them done in the same cycle), and since the nibble shuffling does not cause mixing of the S-boxes outputs in the same round.

6 Results

To demonstrate our results, we use the 90 nm CMOS library provided by TSMC and consider the worst PVT corner case (the temperature of +125° C and the supply voltage of 1.0 V). Please note that the worst corner case is used in almost all industrial applications. In most of the scientific publications, however, typical corner case is usually reported, giving an optimistic estimate of what will eventually be manufactured in silicon. For synthesis, we use Cadence Encounter RTL Compiler version 14.20-s034 to evaluate the proposed architectures. The designs are synthesized using the operating frequency of 10 MHz and the power consumption is estimated by simulating a back-annotated post-synthesis netlist with 100 random test vectors, using Cadence Incisive Enterprise Simulator version 15.10.006. Energy is calculated for one complete encryption/decryption operation. Table 3 shows area, power and energy consumption, the number of random bits required per clock cycle and maximum frequency for all the hardware implementations. All the designs have their unconstrained critical paths well below 100 ns and, hence, collecting area figures and power/energy consumption at the frequency of 10 MHz guarantees fair comparison.

It should be noted that the area, power and energy consumption of PRNG is not included in Table 3, thus making the obtained results favoring solutions with more randomness. In practice, one must take the impact of PRNG into account and it is expected that higher throughput PRNGs consume more area, power and energy. However, in most security applications, PRNG is a component shared between multiple resources, making its impact on the overall area, power and energy consumption limited. Finally, the most energy efficient design, which is a first-order protected $td + 1$ implementation with S-box decomposition from [MS16], consumes only 264 pJ and requires no additional randomness.

As expected, the first-order $d + 1$ TI design with S-box decomposition occupies the smallest area, compared to other secure implementations. Compared to the first-order $td+1$ TI architecture with S-box decomposition, this comes at the cost of extra randomness required.

We report an interesting observation when comparing energy consumption of different architectures. The smallest energy consumption of 264 pJ has been achieved for the first-order secure $td + 1$ TI architecture with S-box decomposition from [MS16]. This is closely followed by our design with the same security order and S-box structure that consumes 270 pJ. We attribute this to the absence of randomness needed for resharing in this specific design, despite the area of both versions of first-order $td + 1$ TI architectures with S-box decomposition being larger compared to

Table 3: 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
[MS16] 1st $(td+1)$ with S-box decomp.	9484	66	264	0	40	328	122
1st $(d+1)$ with S-box decomp.	8701	97	698	24	72	260	277
1st $(td+1)$ with S-box decomp.	14153	75	270	0	36	268	134
1st $(d+1)$ w/o S-box decomp.[1]	12220	115	276	112	24	289	83.0
1st $(td+1)$ w/o S-box decomp.[1]	31116	576	691	48	12	204	58.8
2nd $(d+1)$ with S-box decomp.	13421	161	1159	72	72	250	288
2nd $(td+1)$ with S-box decomp.	18767	232	1670	40	72	243	296
2nd $(d+1)$ w/o S-box decomp.[1]	32444	374	898	432	24	292	82.2
2nd $(td+1)$ w/o S-box decomp.[1]	177647	1533	3679	352	24	282	85.1

[1] Designs introduced in this paper

several other designs in the Table 3. Absence of randomness greatly reduces switching activity of the circuit lowering the power consumption considerably. Still, we observe that the first-order $d+1$ TI architecture without S-box decomposition consumes only 5 % more energy, while being 33 % faster. Another interesting observation is that the first-order secure designs consume considerably less energy compared to second-order designs.

For second-order designs, the designs without decomposition lead to large area overheads (particularly in $td+1$ scenario) and high number of random bits compared to simpler designs. We conclude that the $d+1$ designs are still interesting implementation choices if enough randomness can be provided. Second-order $td+1$, on the other hand, is quite unpractical due to its large area overheads and large power and energy consumption.

One can see that all protected designs except first-order $td+1$ without S-box decomposition have their maximal frequency within 20 % of each other. The reason for the first-order $td+1$ without S-box decomposition smaller maximum frequency is the absence of register prior to the S-box operation. Also, the implementation from [MS16] has smaller critical path compared to our designs. Critical path for all implementations goes from the round counter to the S-box input register. In first-order $td+1$ without S-box decomposition case we do not have the S-box input register, thus making the critical part longer. Still, even with this limitation, the $td+1$ first-order version has smaller total latency compared to other designs.

Compared to our designs, the design described in [MS16] stores the key in plain, requiring less area for key storage. In addition, the authors of [MS16] proposed different affine transformation of decomposed S-boxes and their architecture has simplified interface and control logic. That is the reason why their design is considerably smaller and has lower power consumption than the first-order $td+1$ version of our proposal with S-box decomposition. When the energy consumption is compared, however, the two designs perform similarly with the design of [MS16] being 2.3 % more efficient. This is obviously due to the fact that our first-order $td+1$ design with S-box decomposition is 10 % faster in terms of the required number of clock cycles.

Another interesting observation is that our first-order $td+1$ design with S-box decomposition has lower power consumption than four other designs form Table 3, while having larger area. As discussed previously, this is due to no additional randomness being required during the encryption/decryption process. We have investigated the impact of mask refreshing further and came to the conclusion that adding (or removing) the mask refreshing might change the power consumption up to a factor of two. Example for this is the first-order $d+1$ TI without S-box decomposition, where the power/energy consumption drops by 40 % if the random inputs are set to zero. Hence when achieving lower power/energy is a main requirement using uniform sharing is the best approach.

Table 3 also clearly shows the difference between power and energy consumption. The most extreme example is comparison between first-order $td+1$ design without S-box decomposition and $d+1$ design with S-box decomposition. Although $td+1$ design without S-box decomposition has almost 6 times the power consumption, it has slightly smaller energy consumption, as it takes 6 times less clock cycles to complete.

As can be seen by the reported figures, adding side-channel countermeasures increases the size of the unprotected PRINCE by at least a factor of 2.5. It should be noted that one has the penalty of extra clock cycles as well in all the cases except the first-order $td+1$ without S-box decomposition version.

The fastest unprotected PRINCE takes 30.5 ns, followed by first-order $td+1$ TI without decomposition which takes 58.8 ns, i.e. 93% latency increase; next is the second-order $d+1$ TI without decomposition which takes 82.2 ns, i.e. additional 41% latency increase. Also, all designs without S-box decomposition have significantly smaller latency compared to the implementation presented in [MS16], ranging from 1.4 to 2 times performance increase.

7 Side-channel evaluation

We have chosen for evaluation 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 specially designed for side-channel evaluation. The design is separated into two FPGAs in order to minimize the noise: one that performs the PRINCE encryption and second FPGA that 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. Since one trace consists of 2500 points are able to cover the first seven rounds of the execution. The power waveform is given in Figure 7.

We apply a non-specific leakage detection test [CDG$^+$13] on the input plaintext following the standard methodology [RGV17]. First, we test the design with randomness off to verify validity of the setup and we are able to detect leakage with 1 million traces. As it can be seen on the left hand side in Figure 8, there is a strong first-order leakage during the loading of the plaintext

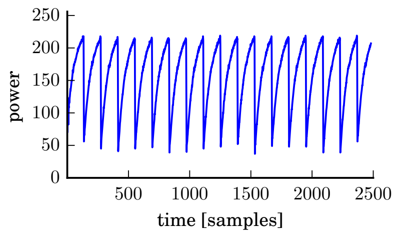

Figure 7: Example power trace waveform used to perform the t-test.

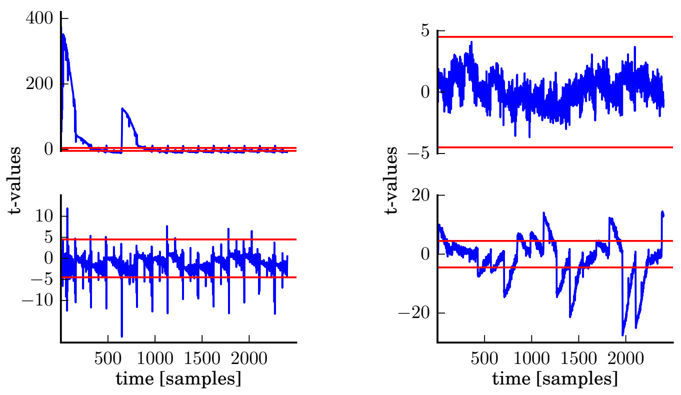

Figure 8: Leakage detection test results. PRNG off (left column) and PRNG on (right column). First- (top row) and second- (bottom row) order t-test results.

and the key. We can attribute this to the fact that one share of both the key and the plaintext is equal to the unshared value, while the other share is zero. Another strong peak is during the first S-box execution since the state has not diffused yet and there is still high correlation to the input. Nonetheless, since there is no randomness added, leakage is evident in later rounds as well, although it becomes smaller. Second-order leakage can also be observed when the masks are off.

Then we switch the randomness on and corroborate that the design does not leak using 100 million traces. Namely it can be seen that there is no first-order leakage detected for the threshold of ± 4.5 when the PRNG is on. Also as expected there is a second-order leakage since we have tested the first-order $d+1$ implementation. The resulting t-test graphs are shown in the Figure 8.

8 Conclusion and Outlook

In this paper we introduced a method for optimizing Threshold Implementations, which makes the low-latency of side-channel secure designs practical. We have investigated several different trade-offs that occur in side-channel secure designs. Particularly, we discuss the energy consumption of given implementations, an important factor in several applications, such as battery powered devices.

First, 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 $t = n-1$. Second, we presented a heuristic for minimizing the number of output shares for higher-order $td+1$ TI. In addition to that, we would like to point out that this contribution is of general interest since the method of minimizing the number of output shares can equally well be applied to any cryptographic design.

Finally, we reported, evaluated and compared hardware figures for eight different TI-protected round-based versions of PRINCE cipher, namely $d+1$ and $td+1$ TI versions, first- and second-order secure, with or without the S-box decomposition. The $td+1$ TI versions tend to consume less randomness. The $d+1$ TI versions with decomposition achieve lower area and power consumption. The first-order designs without decomposition have favorable energy consumption. In comparison with the previous state of the art, we show that our design without S-box decomposition are 1.4 to 2 times faster than architecture of [MS16]. It should be noted that [MS16] design still has the most power efficient design and also the most energy efficient design, albeit by a very small margin in the latter category. From the area viewpoint, [MS16] design is the second smallest one.

As discussed in [MS16] 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 round based first and second-order secure implementations of PRINCE using $td+1$ and $d+1$ TI sharing correspondingly.

Acknowledgements

We would like to thank Amir Moradi for providing us with HDL code of PRINCE TI presented in [MS16].

Dušan Božilov is supported by the European Union's Horizon 2020 Marie Skłodowska-Curie research fellowship under grant agreement $No.$ 643161 (ECRYPT-NET).

References

[BBI$^+$15] Subhadeep Banik, Andrey Bogdanov, Takanori Isobe, Kyoji Shibutani, Harunaga Hiwatari, Toru Akishita, and Francesco Regazzoni. Midori: A block cipher for low energy. In *ASIACRYPT 2015*, pages 411–436. Springer-Verlag New York, Inc., 2015.

[BCG$^+$12] Julia Borghoff, Anne Canteaut, Tim Güneysu, Elif Bilge Kavun, Miroslav Knezevic, Lars R. Knudsen, Gregor Leander, Ventzislav Nikov, Christof Paar, Christian Rechberger, Peter Rombouts, Søren S. Thomsen, and Tolga Yalçın. Prince – a low-latency block cipher for pervasive computing applications. In *ASIACRYPT 2012*, pages 208–225. Springer LNCS, 2012.

[BDPA11] Guido Bertoni, Joan Daemen, Michaël Peeters, and Gilles Van Assche. The Keccak reference, January 2011. http://keccak.noekeon.org/.

[BGN$^+$14] Begül Bilgin, Benedikt Gierlichs, Svetla Nikova, Ventzislav Nikov, and Vincent Rijmen. Higher-order threshold implementations. In *ASIACRYPT 2014*, pages 326–343. Springer LNCS, 2014.

[BKL$^+$07] Andrey Bogdanov, Lars R. Knudsen, Gregor Leander, Christof Paar, Axel Poschmann, Matthew J. B. Robshaw, Yannick Seurin, and Charlotte Vikkelsoe. PRESENT: An ultra-lightweight block cipher. In Pascal Paillier and Ingrid Verbauwhede, editors, *Cryptographic Hardware and Embedded Systems - CHES 2007*, pages 450–466. Springer Berlin Heidelberg, 2007.

[BNN$^+$12] Begül Bilgin, Svetla Nikova, Ventzislav Nikov, Vincent Rijmen, and Georg Stütz. Threshold implementations of all 3 $\times$3 and 4 $\times$4 s-boxes. In *CHES 2012*, pages 76–91. Springer LNCS, 2012.

[CDG$^+$13] Jeremy Cooper, Elke DeMulder, Gilbert Goodwill, Joshua Jaffe, Gary Kenworthy, and Pankaj Rohatgi. Test vector leakage assessment (TVLA) methodology in practice. In *International Cryptographic Module Conference*, 2013.

[GM17] Hannes Gross and Stefan Mangard. Reconciling $d+1$ masking in hardware and software. In *Cryptographic Hardware and Embedded Systems – CHES*. Springer International Publishing, 2017.

[GMK16] Hannes Groß, Stefan Mangard, and Thomas Korak. 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, October, 2016*, page 3, 2016.

[ISW03] Yuval Ishai, Amit Sahai, and David Wagner. Private circuits: Securing hardware against probing attacks. In *CRYPTO 2003*, pages 463–481. Springer Berlin Heidelberg, 2003.

[KJJ99] Paul C. Kocher, Joshua Jaffe, and Benjamin Jun. Differential power analysis. In *CRYPTO 1999*, pages 388–397. Springer LNCS, 1999.

[MS16] Amir Moradi and Tobias Schneider. Side-channel analysis protection and low-latency in action - case study of PRINCE and Midori. In *ASIACRYPT 2016*. Springer LNCS, 2016.

[Nik12] Svetla Nikova. TI tools for the 3x3 and 4x4 S-boxes, 2012. http://homes.esat.kuleuven.be/~snikova/ti_tools.html.

[NRR06] Svetla Nikova, Christian Rechberger, and Vincent Rijmen. Threshold implementations against side-channel attacks and glitches. In *ICICS 2006*, pages 529–545. Springer LNCS, 2006.

[Pap14] Kostas Papapagiannopoulos. High throughput in slices: The case of PRESENT, PRINCE and KATAN64 ciphers. In *RFIDSec 2014*, pages 137–155. Springer LNCS, 2014.

[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. *Journal of Cryptology*, 24(2):322–345, 2011.

[RBN$^+$15] Oscar Reparaz, Begül Bilgin, Svetla Nikova, Benedikt Gierlichs, and Ingrid Verbauwhede. Consolidating masking schemes. In *CRYPTO 2015*, pages 764–783. Springer LNCS, 2015.

[RGV17] Oscar Reparaz, Benedikt Gierlichs, and Ingrid Verbauwhede. Fast leakage assessment. In *Cryptographic Hardware and Embedded Systems - CHES 2017*, pages 387–399, 2017.

[UHA17a] Rei Ueno, Naofumi Homma, and Takafumi Aoki. 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)*, pages 136–141, 2017.

[UHA17b] Rei Ueno, Naofumi Homma, and Takafumi Aoki. Toward more efficient DPA-resistant AES hardware architecture based on threshold implementation. In *Constructive Side-Channel Analysis and Secure Design - COSADE 2017*, pages 50–64, 2017.

Appendix A

Example of the iterative step from algorithm 2 is presented below, where for security order $d = 2$ we have created 3 conjugate D_2^3-tables from 3 conjugate D_1^2-tables.

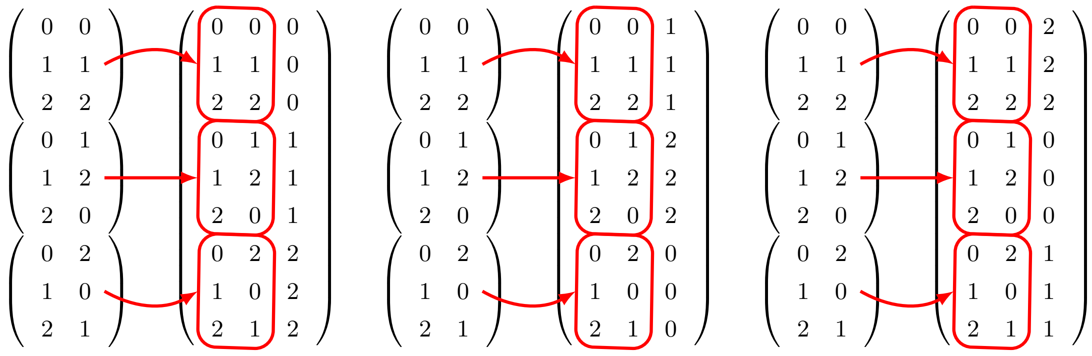

Figure 9: Generating conjugate D_2^3-tables from D_1^2-tables.

Appendix B

8.1 First-order secure $td + 1$ TI of Q_{294}

We use first-order $td + 1$ direct TI sharing [BNN+12] with 3 shares. Here, we recall that $d = 1$ and $t = 2$. The actual sharing is given in Equation (15).

$$\begin{aligned}
x_1 &= a_1 & z_1 &= a_1b_1 \oplus a_1b_2 \oplus a_2b_1 \oplus c_1\\
x_2 &= a_2 & z_2 &= a_2b_2 \oplus a_2b_3 \oplus a_3b_2 \oplus c_2\\
x_3 &= a_3 & z_3 &= a_3b_3 \oplus a_3b_1 \oplus a_1b_3 \oplus c_3\\
\\
y_1 &= b_1 & w_1 &= a_1c_1 \oplus a_1c_2 \oplus a_2c_1 \oplus d_1\\
y_2 &= b_2 & w_2 &= a_2c_2 \oplus a_2c_3 \oplus a_3c_2 \oplus d_2\\
y_3 &= b_3 & w_3 &= a_3c_3 \oplus a_3c_1 \oplus a_1c_3 \oplus d_3
\end{aligned} \tag{15}$$

Figure 10 depicts the hardware implementation of the $td + 1$ version of Q_{294}.

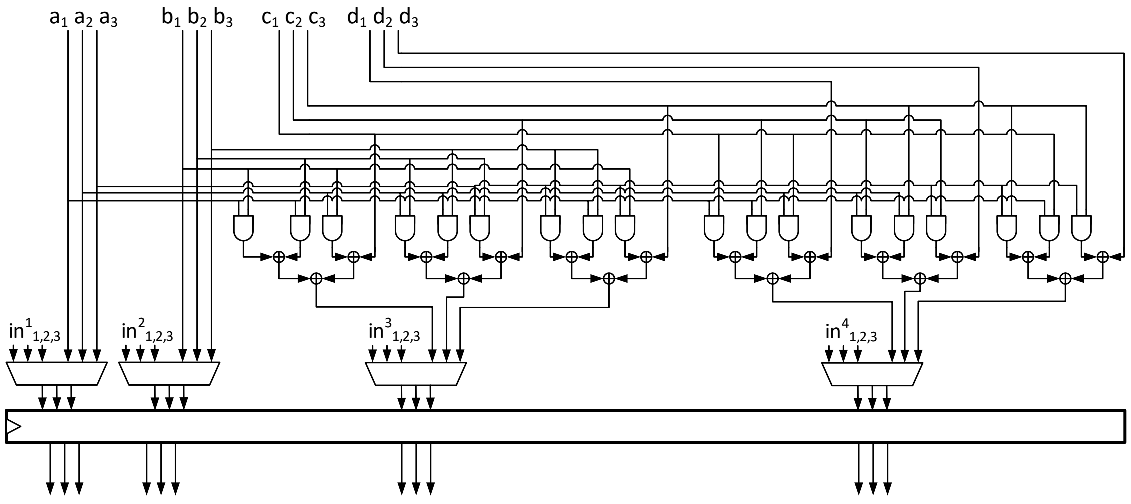

Figure 10: First-order secure sharing of Q_{294} with $td + 1$ TI.

8.2 Second-order secure $td + 1$ TI of Q_{294}

We use the second-order $td + 1$ TI sharing of Q_{294} with 5 input shares and 10 output shares as shown in Equation (16). In this case we have $d = 2$ and $t = 2$. The shares are first processed and thus expanded, then refreshed and stored into a register. Next, they are compressed into 5 shares using the method explained in [BGN+14]. Values in Equation (16) denoted with the overline represent the output after the compression step.

$$
\begin{aligned}
&x_1 = a_1 && && y_1 = b_1\\
&x_2 = a_2 && && y_2 = b_2\\
&x_3 = a_3 && && y_3 = b_3\\
&x_4 = a_4 && && y_4 = b_4\\
&x_5 = a_5 && && y_5 = b_5\\[1em]
&z_1 = a_1b_3 \oplus a_3b_1 && z_6 = a_1b_1 \oplus a_1b_2 \oplus a_2b_1 \oplus c_1 && \bar{z}_1 = z_1 \oplus z_6\\
&z_2 = a_2b_4 \oplus a_4b_2 && z_7 = a_2b_2 \oplus a_2b_3 \oplus a_3b_2 \oplus c_2 && \bar{z}_2 = z_2 \oplus z_7\\
&z_3 = a_3b_5 \oplus a_5b_3 && z_8 = a_3b_3 \oplus a_3b_4 \oplus a_4b_3 \oplus c_3 && \bar{z}_3 = z_3 \oplus z_8\\
&z_4 = a_4b_1 \oplus a_1b_4 && z_9 = a_4b_4 \oplus a_4b_5 \oplus a_5b_4 \oplus c_4 && \bar{z}_4 = z_4 \oplus z_9\\
&z_5 = a_5b_2 \oplus a_2b_5 && z_{10} = a_5b_5 \oplus a_5b_1 \oplus a_1b_5 \oplus c_5 && \bar{z}_5 = z_5 \oplus z_{10}\\[1em]
&w_1 = a_1c_3 \oplus a_3c_1 && w_6 = a_1c_1 \oplus a_1c_2 \oplus a_2c_1 \oplus d_1 && \bar{w}_1 = w_1 \oplus w_6\\
&w_2 = a_2c_4 \oplus a_4c_2 && w_7 = a_2c_2 \oplus a_2c_3 \oplus a_3c_2 \oplus d_2 && \bar{w}_2 = w_2 \oplus w_7\\
&w_3 = a_3c_5 \oplus a_5c_3 && w_8 = a_3c_3 \oplus a_3c_4 \oplus a_4c_3 \oplus d_3 && \bar{w}_3 = w_3 \oplus w_8\\
&w_4 = a_4c_1 \oplus a_1c_4 && w_9 = a_4c_4 \oplus a_4c_5 \oplus a_5c_4 \oplus d_4 && \bar{w}_4 = w_4 \oplus w_9\\
&w_5 = a_5c_2 \oplus a_2c_5 && w_{10} = a_5c_5 \oplus a_5c_1 \oplus a_1c_5 \oplus d_5 && \bar{w}_5 = w_5 \oplus w_{10}
\end{aligned}
\tag{16}
$$

Please note that in order to avoid multivariate attacks, where the attacker probes values from different time samples, only nonlinear parts need to be refreshed, namely $z_1, \ldots, z_5$ and $w_1, \ldots, w_5$. Therefore, we need 10 random bits for each shared Q_{294} function.

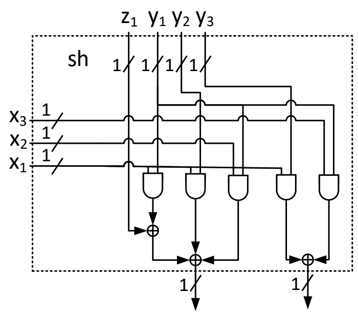

Figure 11: Generating two outputs bits for partial evaluation of $xy + z$.

The sub-circuit used to generate two output bits of a partial evaluation of shared nonlinear function $xy + z$ is shown in Figure 11.

8.3 First-order secure $d + 1$ TI of Q_{294}

We use the first-order sharing given in [RBN+15] and shown in Equation (17). In this case it holds $d = 1$. Unlike $td + 1$ TI, the first-order secure sharing here has four output shares for the nonlinear component functions. For the linear parts, however, we need only two shares instead of three. Compression and mask refreshing are needed to reduce the number of output shares and make the output uniform, respectively.

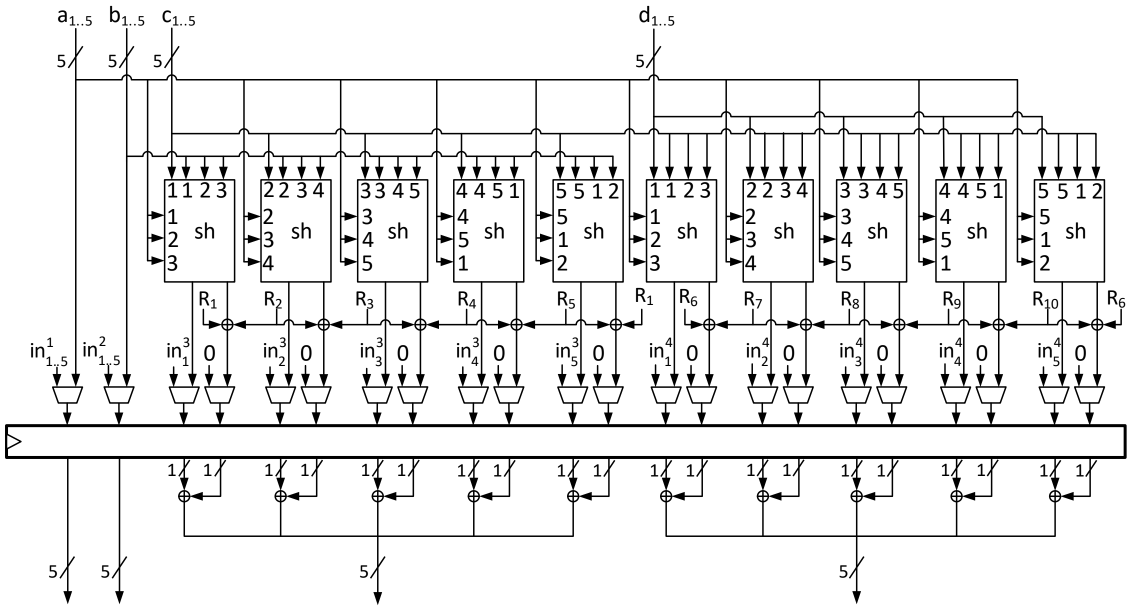

Figure 12: Second-order secure sharing of Q_{294} with $td + 1$ TI.

$$
\begin{aligned}
x_1 &= a_1 & y_1 &= b_1 \\
x_2 &= a_2 & y_2 &= b_2 \\
\\
z_1 &= a_1 b_1 \oplus c_1 & w_1 &= a_1 c_1 \oplus d_1 \\
z_2 &= a_1 b_2 & w_2 &= a_1 c_2 \\
z_3 &= a_2 b_2 \oplus c_2 & w_3 &= a_2 c_2 \oplus d_2 \\
z_4 &= a_2 b_1 & w_4 &= a_2 c_1 \\
\\
\bar{z}_1 &= z_1 \oplus z_2 & \bar{w}_1 &= w_1 \oplus w_2 \\
\bar{z}_2 &= z_3 \oplus z_4 & \bar{w}_2 &= w_3 \oplus w_4
\end{aligned}
\tag{17}
$$

Shares that contain quadratic terms are refreshed as given in Equation (4) before storing into a register. We have two shared output component functions with four shares, for which we need 6 random bits. As in the second-order secure $td + 1$ version we set appropriate register bits to 0 during initial loading to ensure correctness of the execution. A detailed hardware implementation of the $d + 1$ TI sharing of Q_{294} is depicted in Figure 13.

8.4 Second-order secure $d + 1$ TI of Q_{294}

Next, we create a second-order secure masking of Q_{294} following the work of [RBN+15]. In this case $d = 2$. Three input shares are needed for all the operations. However, sharing a nonlinear operation $xy + z$ produces 9 output shares that need to be first refreshed, then stored into a register, and

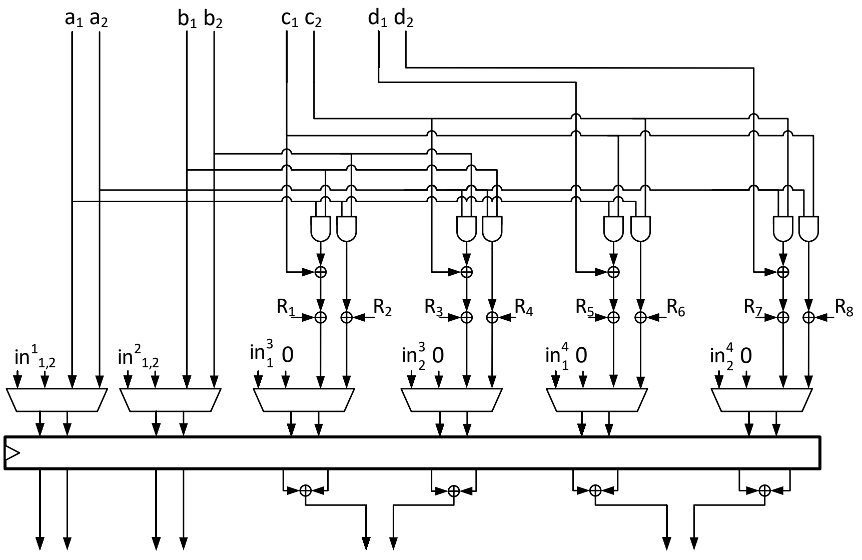

Figure 13: First-order secure sharing of Q_{294} with $d + 1$ TI.

finally compressed. We give the formula for $d + 1$ second-order secure sharing in Equation (18).

$$
\begin{aligned}
x_1 &= a_1 & y_1 &= b_1\\
x_2 &= a_2 & y_2 &= b_2\\
x_3 &= a_3 & y_3 &= b_3\\
\\
z_1 &= a_1b_1 \oplus c_1 & w_1 &= a_1c_1 \oplus d_1\\
z_2 &= a_1b_2 & w_2 &= a_1c_2\\
z_3 &= a_1b_3 & w_3 &= a_1b_3\\
z_4 &= a_2b_1 & w_4 &= a_2c_1\\
z_5 &= a_2b_2 \oplus c_2 & w_5 &= a_2c_2 \oplus d_2\\
z_6 &= a_2b_3 & w_6 &= a_2c_3\\
z_7 &= a_3b_1 & w_7 &= a_3c_1\\
z_8 &= a_3b_2 & w_8 &= a_3c_2\\
z_9 &= a_3b_3 \oplus c_3 & w_9 &= a_3c_3 \oplus d_3\\
\\
\bar{z}_1 &= z_1 \oplus z_2 \oplus z_3 & \bar{w}_1 &= w_1 \oplus w_2 \oplus w_3\\
\bar{w}_2 &= w_4 \oplus w_5 \oplus w_6 & \bar{w}_2 &= w_4 \oplus w_5 \oplus w_6\\
\bar{w}_3 &= w_7 \oplus w_8 \oplus w_9 & \bar{w}_3 &= w_7 \oplus w_8 \oplus w_9
\end{aligned}
\tag{18}
$$

A hardware diagram of this sharing is depicted in Figure 14.

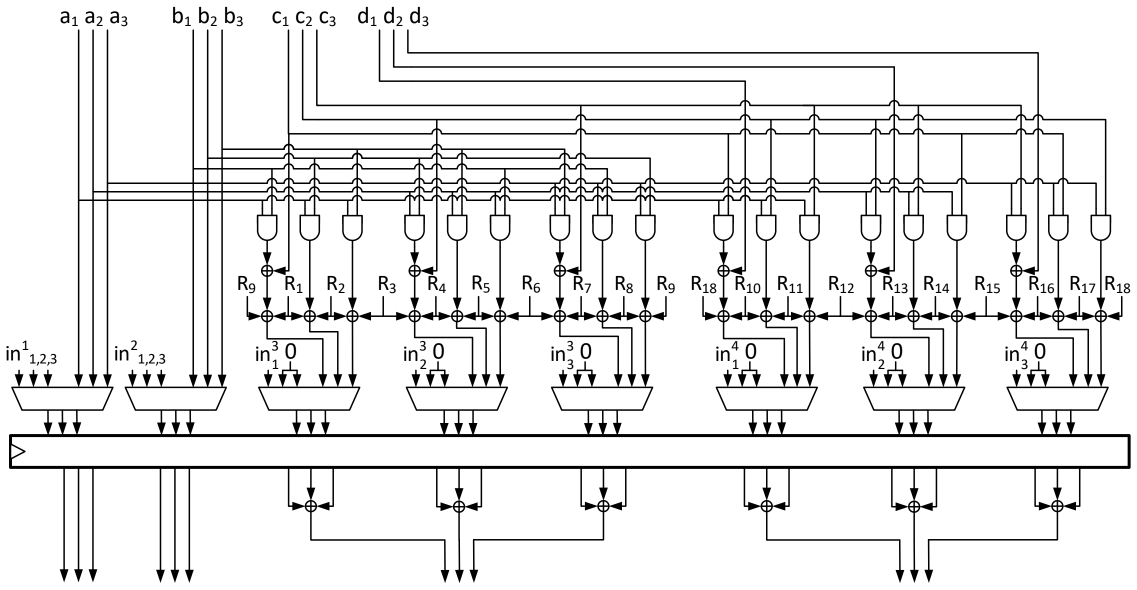

Figure 14: Second-order secure sharing of Q_{294} with $d+1$ TI.