

Rhythmic Keccak: SCA Security and Low Latency in HW

Victor Arribas¹, Begül Bilgin¹, George Petrides²,
Svetla Nikova¹ and Vincent Rijmen¹

¹ KU Leuven, imec-COSIC, Belgium, name.surname@esat.kuleuven.be

² Vrije Universiteit Brussel, Belgium, george.petrides@vub.be

Abstract. Glitches entail a great issue when securing a cryptographic implementation in hardware. Several masking schemes have been proposed in the literature that provide security even in the presence of glitches. The key property that allows this protection was introduced in threshold implementations as *non-completeness*. We address crucial points to ensure the right compliance of this property especially for low-latency implementations. Specifically, we first discuss the existence of a flaw in DSD 2017 implementation of KECCAK by Gross *et al.* in violation of the non-completeness property and propose a solution. We perform a side-channel evaluation on the first-order and second-order implementations of the proposed design where no leakage is detected with up to 55 million traces. Then, we present a method to ensure a non-complete scheme of an unrolled implementation applicable to any order of security or algebraic degree of the shared function. By using this method we design a two-rounds unrolled first-order KECCAK- f [200] implementation that completes an encryption in 20.61ns, the fastest implementation in the literature to this date.

Keywords: Glitch · non-completeness · threshold implementation · consolidated masking scheme · domain-oriented masking

1 Introduction

Physical attacks are a serious threat to cryptographic implementations, capable of retrieving important information such as the secret key. In particular, Side-Channel Attacks (SCA), which are based on observing the behavior of the device without making any changes on it or its working conditions, are used frequently due to their relatively low cost and difficulty to be detected. In this paper, we use Differential Power Analysis (DPA) in particular, which exploits the relation between the intermediate values produced internally during the calculations and the instantaneous power consumption of the cryptographic device [KJJ99]. However, our observations can be generalized to other forms of SCA, such as the ones exploiting the electromagnetic emanation of the device [GMO01].

Different countermeasures have been proposed that aim to make the power consumption of a cryptographic device independent of the intermediate values. Here, we focus on Threshold Implementation (TI) [NRR06, NRS08, NRS11] and Domain Oriented Masking (DOM) [GMK16], which are based on secret sharing schemes and techniques from Multi-Party Computation (MPC). Moreover, they have the advantage of providing theoretical security on hardware if implemented according to the non-completeness property defined in [NRR06, BGN⁺14a], if fed with enough entropy and if the device works under the independent leakage assumption as described in [DFS15]. There are several papers in the literature applying these countermeasures on the KECCAK permutations: Bertoni *et al.* [BDPA10a] and Bilgin *et al.* [BDN⁺14] with first-order resistant implementations and more recent work by Gross *et al.* [GSM17] that proposes higher-order countermeasures.

Our contribution In this paper, we first briefly summarize KECCAK sponge function, and TI and DOM schemes (Sect. 2). Then, we analyze the recently published higher-order DOM KECCAK implementations [GSM17] and point out a flaw that can possibly lead to successful attacks. We describe how careful tracing of the non-completeness property can be done to fix this flaw for any order. We show the different behavior between the original design and our proposal for first- and second-order implementations using TVLA with 55 million of traces (Sect. 3). Finally, we discuss how TI can be used for unrolled implementations without breaking the non-completeness property for the first time in literature. We present a low latency KECCAK implementation that performs an encryption in 20.61ns making it the fastest implementation published to this date (Sect. 4).

2 Preliminaries

2.1 Keccak Permutations

KECCAK is a family of sponge functions using the permutations KECCAK- $f[b]$ where $b \in \{25, 50, 100, 200, 400, 800, 1600\}$ defines different state sizes [BDPA10b]. KECCAK [BDPA10b] is the NIST SHA-3 standard [Rom14]. Among other properties, KECCAK stands out due to its high performance when implemented in hardware and its great area/speed trade-offs. The security claims of these variants follow the *Matryoshka principle*, i.e. analysis on of small sizes can easily be linked to bigger sizes. Benefiting from this property, we work on KECCAK- $f[200]$, which is suitable for lightweight architectures and note that the concepts presented in this paper can naturally be extended to other choices of b . Moreover, since we simply work on KECCAK- f this work also covers KETJE [BDP⁺16a] and KEYAK [BDP⁺16b] authenticated encryptions.

KECCAK- f operates on a three-dimensional state S , where the bit in the coordinate (x, y, z) is denoted by $S[x, y, z]$. A round consists of five steps:

$$R = \iota \circ \chi \circ \pi \circ \rho \circ \theta, \text{ with} \quad (1)$$

$$\begin{aligned} \theta: \quad S[x, y, z] &\leftarrow S[x, y, z] \oplus \bigoplus_{y'=0}^4 S[x-1, y', z] \oplus \bigoplus_{y'=0}^4 S[x+1, y', z-1], \\ \rho: \quad S[x, y, z] &\leftarrow S[x, y, z - (t+1)(t+2)/2], \\ &\text{with } t \text{ satisfying } 0 \leq t < 24 \text{ and } \begin{pmatrix} 0 & 1 \\ 2 & 3 \end{pmatrix}^t \begin{pmatrix} 1 \\ 0 \end{pmatrix} = \begin{pmatrix} x \\ y \end{pmatrix} \text{ in } \text{GF}(5)^{2 \times 2}, \\ &\text{or } t = -1 \text{ if } x = y = 0, \\ \pi: \quad S[x, y] &\leftarrow S[x', y'], \text{ with } \begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} 0 & 1 \\ 2 & 3 \end{pmatrix} \begin{pmatrix} x' \\ y' \end{pmatrix}, \\ \chi: \quad S[x] &\leftarrow S[x] \oplus (S[x+1] \oplus 1)S[x+2], \\ \iota: \quad S &\leftarrow S \oplus \text{RC}[i_r], \text{ where } \text{RC}[i_r] \text{ is the round counter in cycle } i_r. \end{aligned}$$

KECCAK- $f[200]$ computes 18 rounds. Notice that we abuse the notation to focus on specific coordinates. Moreover, addition and subtraction in the coordinates are modular. For convenience, we provide visualizations of these steps in App. A.1 (Fig. 8) for KECCAK- $f[200]$.

2.2 Masking Schemes

Different forms of masking schemes have been shown to provide provable security against side-channel analysis attacks given independent leakage and uniform input assumption. They are based on secret sharing where each sensitive, i.e. key dependent, data x is divided into s pieces ($\mathbf{x} = (x_1, \dots, x_s)$) such that $x = x_1 \perp \dots \perp x_s$. Throughout this paper, we consider Boolean masking where \perp is field addition denoted by \oplus and full threshold sharing where all s shares need to be combined to derive x . The exact value of s is chosen depending on the desired security level which is specified further for different flavors of masking below.

Each function $f(x) = y$ of the cryptographic algorithm is calculated in a shared manner where $f_i(\mathbf{x}) = y_i$. Masking is *correct* if $\bigoplus y_i = y$.

d -Probing Model. It has been shown in [DFS15] that if a masked circuit achieves security under the d -probing model [ISW03] where each calculation is treated separately, independent of the timing of the circuit, then it also provides security against d^{th} -order side-channel analysis attacks under independent leakage assumption. Similar to [RBN⁺15, FGP⁺17], we assume that an adversary probing a wire has information about all the intermediate values starting from the registers to the probed point of calculation. Note that this probing model is used specifically on hardware where glitches might occur in the circuit. As a result a d -probing adversary would acquire the knowledge of all the intermediate values used in all d probed wires.

2.2.1 Threshold Implementation (TI)

This masking method, which is introduced by Nikova *et al.* [NRR06] for first-order security and extended by Bilgin *et al.* [BGN⁺14a] for higher-order security, is used widely on hardware since, unlike many masking schemes, it provides security under the non-ideal gate assumption. That is, the gates can glitch depending on prior inputs of that cycle before stabilizing without giving an advantage to an attacker. Below we repeat the non-completeness definition for completeness.

Definition 1 (Non-completeness [BGN⁺14a]). The shared circuit \mathbf{f} of f is d^{th} -order non-complete if any combination of up to d component functions f_i is independent of at least one input share.

The lower bounds of number of input and output shares required to calculate a function of algebraic degree t such that it satisfies the above non-completeness property are given in [BGN⁺14a] as follows:

$$\begin{aligned} s_{in} &\geq td + 1, \\ s_{out} &\geq \binom{s_{in_{min}}}{t}. \end{aligned} \tag{2}$$

This bound has then been used for a variety of algorithms and security orders [BNN⁺12, BGN⁺14b, BNN⁺14, BGN⁺15, BDN⁺14, CBR⁺15].

We provide the sharings of an AND/XOR gate ($z = a \oplus bc$) where $(d, s_{in}, s_{out}) = (1, 3, 3)$ and $(d, s_{in}, s_{out}) = (2, 5, 10)$ in Eqns (3) and (9) (in App. A.2) respectively together with a graphical representation in Fig. 9 (in App. A.2). We refer to a sharing with $s_{in} = s_{out} = s$ as an s -sharing.

$$\begin{aligned} z_1 &= a_1 \oplus b_1 c_1 \oplus b_1 c_2 \oplus b_2 c_1, \\ z_2 &= a_2 \oplus b_2 c_2 \oplus b_2 c_3 \oplus b_3 c_2, \\ z_3 &= a_3 \oplus b_3 c_3 \oplus b_1 c_3 \oplus b_3 c_1. \end{aligned} \tag{3}$$

In order to satisfy the non-completeness property, the *compression* from s_{out} to s_{in} shares must follow a synchronization layer such as registers. The first-order case becomes advantageous since $s_{out} = s_{in}$ and this intermediate synchronization layer is not necessary enabling a one cycle implementation. If a sharing dissatisfies the non-completeness property, we call it a *complete* sharing.

Another property introduced by TI is a uniform sharing. Since we only use permutations in this paper, we provide a simplified definition below.

Definition 2 (Uniform Sharing [BGN⁺14a]). The s -sharing of an n -bit permutation $f(x) = y$ is uniform if its sharing $\mathbf{f}(\mathbf{x}) = \mathbf{y}$ (after compression) is an ns -bit permutation.

If this property is satisfied for each step of a cryptographic algorithm, the algorithm provides univariate security without the need of additional randomness. It has been shown in [RBN⁺15] that uniformity condition is not enough for multi-variate security in the higher-order case. The authors show that it is possible to achieve higher-order multi-variate security for TI by inserting fresh randomness to each s_{out} output shares just before the compression. This *refreshing* naturally provides a uniform sharing.

2.2.2 Domain Oriented Masking (DOM)

In [GMK16], Gross *et al.* introduced d^{th} -order secure DOM-indep multiplier which uses $s_{in} = d + 1$ input shares and $d(d + 1)/2$ units of randomness. DOM multiplier assumes s_{in} domains.

The sharing structure of DOM-indep multiplier uses $d + 1$ input shares for hardware and is a follow up of Reparaz *et al.* [RBN⁺15] which provides security given only the independence of the shared input variables. The difference between DOM-indep as opposed to [RBN⁺15] multiplier is the significant randomness optimization, from $(d + 1)^2$ to $d(d + 1)/2$. This optimized shared multiplier has been used in [GSM17] to provide very small implementations of KECCAK with higher-order security claim. We provide a DOM-indep AND gate ($z = ab$) where $(d, s_{in}, s_{out}) = (1, 2, 4)$ and $(d, s_{in}, s_{out}) = (2, 3, 9)$ in Eqn. 4 below and in Eqn. 10 in appendix. The parenthesis $[\cdot]$ and (\cdot) represent the mandatory and optional synchronizations respectively and r refers to the randomness used for refreshing.

$$\begin{aligned} z_1 &= (a_1 b_1) \oplus [a_1 b_2 \oplus r], \\ z_2 &= [a_2 b_1 \oplus r] \oplus (a_2 b_2). \end{aligned} \tag{4}$$

In [GMK16] a second multiplier called DOM-dep multiplier is also introduced. DOM-dep has the advantage of not relying on the independently shared inputs assumption of other $d + 1$ share schemes DOM-indep and [RBN⁺15] while using less randomness. For details of DOM-dep multiplier and randomness reductions of DOM, we refer to the original paper. Here, we only focus on DOM-indep multiplier which was used in [GSM17] and the non-completeness property. Therefore, we refer to DOM-indep as DOM for brevity.

3 Round-Based Implementations

KECCAK is implemented for a variety of platforms and constraints. The smallest round-based implementation claiming security against side-channel analysis attacks is the work from Gross *et al.* [GSM17] which uses DOM. In this section, we first show an exploitable weakness of that round based implementation based on the failure of non-completeness property. We then verify the observability of our claims on an FPGA using t-test based leakage detection. Finally, we discuss how a sharing using $s_{in} \geq d + 1$ shares should be

implemented such that it satisfies the non-completeness property and the restrictions of such an implementation. Note that our analysis covers round-based implementations of any order but does not cover the serialized architectures presented in [GSM17].

3.1 Analysis of DOM-Keccak

A traditional round-based implementation of KECCAK receives the input of θ from a register; θ , ρ , π , χ and ι operations are calculated within the same clock cycle and the output of ι is written to a register. When we consider a shared implementation the χ -refreshing needs to be separated from χ -compression by registers to achieve non-completeness marking the end of the cycle. The difference between these two architectures are depicted in Figure 1 where the blue marks correspond to the masked version. The χ -compression is included in the ι block.

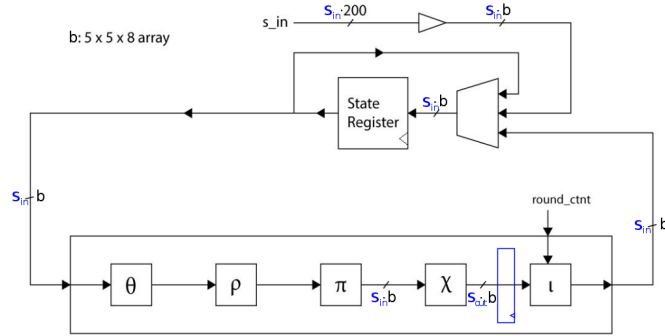


Figure 1: Plain and secure (blue) KECCAK round-based implementation

The implementation in [GSM17] uses the sharing in Eqn. (5) of χ up to the synchronization register for first order. The randomness r is taken from another (independent) part of the state. The synchronization is performed only on the *cross-domain* shares $S'_2[x]$ and $S'_3[x]$ by using a register triggered on the negative edge of the clock while triggering the state register in the positive edge (double clocking). Analyzing each linear and nonlinear step individually as is done in [GSM17], shows that the non-completeness property is satisfied in each separate step. However, the non-completeness of any implementation should be verified from register to register since the glitchy behavior can accumulate.

$$\begin{aligned}
 S'_1[x] &\leftarrow S_1[x] \oplus (S_1[x+1] \oplus 1)S_1[x+2], \\
 S'_2[x] &\leftarrow S_1[x+1]S_2[x+2] \oplus r, \\
 S'_3[x] &\leftarrow S_2[x+1]S_1[x+2] \oplus r, \\
 S'_4[x] &\leftarrow S_2[x] \oplus (S_2[x+1] \oplus 1)S_2[x+2].
 \end{aligned} \tag{5}$$

When we analyzed one round of the aforementioned implementation, we noticed that the non-completeness property from register to register is not satisfied for 112 output bits of the round output invalidating a condition for security. Below we trace back the input shares used to calculate a specific bit of the state particularizing Eqn. (1) for first-order secure implementation. A graphical representation is also provided in Fig. 10 in App. A.3.1. We use S_i^f to refer to the output of f for share S_i :

Tracing back output bit $S^x [4, 1, 0]$ of the χ permutation:

$$\begin{aligned}
\chi^{-1} : S_3^x [4, 1, 0] &\leftarrow S_2^\pi [0, 1, 0] S_1^\pi [1, 1, 0] \oplus r \\
\pi^{-1} : S_2^\pi [0, 1, 0] &\leftarrow S_2^\rho [3, 0, 0] \\
S_1^\pi [1, 1, 0] &\leftarrow S_1^\rho [4, 1, 0] \\
\rho^{-1} : S_2^\rho [3, 0, 0] &\leftarrow S_2^\theta [3, 0, 4] \\
S_1^\rho [4, 1, 0] &\leftarrow S_1^\theta [4, 1, 4] \\
\theta_1^{-1} : S_2^\theta [3, 0, 4] &\leftarrow S_2 [3, 0, 4] \oplus \bigoplus_{y'=0}^4 S_2 [2, y', 4] \oplus \bigoplus_{y'=0}^4 S_2 [4, y', 3] \\
S_1^\theta [4, 1, 4] &\leftarrow S_1 [4, 1, 4] \oplus \bigoplus_{y'=0}^4 S_1 [3, y', 4] \oplus \sum_{y'=0}^4 S_1 [0, y', 3]
\end{aligned}$$

From these expressions it can be derived that the output bit $S_3^x [4, 1, 0]$ of the non-linear layer depends, among others, on the input bits $S_1 [3, 0, 4]$ and $S_2 [3, 0, 4]$. This means that all shares of $[3, 0, 4]$ are used and hence, non-completeness fails. Note that similar complete output bits can also be found for higher-order implementations. We provide a second-order example in App. A.3.1.

3.1.1 Evaluation

To illustrate the problem presented above we implement a first-order DOM-KECCAK and test on FPGA. Our masked round-based implementation follows the structure depicted in Fig. 1. Note that we do not use negative-edge triggering for the cross-domain shares for ease of analysis leading to a two cycle per round implementation. Moreover, we always use fresh randomness to ensure that a possible problem in the second order is not caused by the degradation on the uniformity.

Platform. To evaluate our design, we deploy it into a Spartan-6 Xilinx FPGA on a Sakura-G board, which is specifically designed for side-channel evaluation. To reduce the noise during the evaluation we split the implementation into two different FPGAs: a control FPGA handles I/O with the host computer and supplies data in masked representation to the crypto FPGA. Crypto FPGA holds a PRNG, which generates the randomness for refreshing, and our masked KECCAK designs. In order to get the cleanest possible traces, we clock the PRNG in the negative edge to avoid extra noise and use a very slow 3 MHz clock so that we are sure there is no overlap in the power consumption. The design is synthesized using the Xilinx tools with the property `KEEP HIERARCHY` set to *yes*, in order to avoid optimizations among the different blocks what would compromise the security of the design. We sample at 1.0GS/s with 2000 points per frame over two rounds and a half.

TVLA. We use non-specific test vector leakage assessment (TVLA) method described in [CMG⁺13] to detect leakage. Note that this t-test based leakage detection method is not used to mount an attack and retrieve the key. From a theoretical point of view, the presence of leakage is a necessary, but not sufficient condition for an attack to succeed. However, not observing leakage gives confidence to designer. We choose our confidence level to be 99.9995% which corresponds to absolute t-values being greater than 4.5 for failure.

Two different tests are performed, namely, Fix vs. Random with masks off and with masks on. In both cases the idea is to compare the power consumption of the

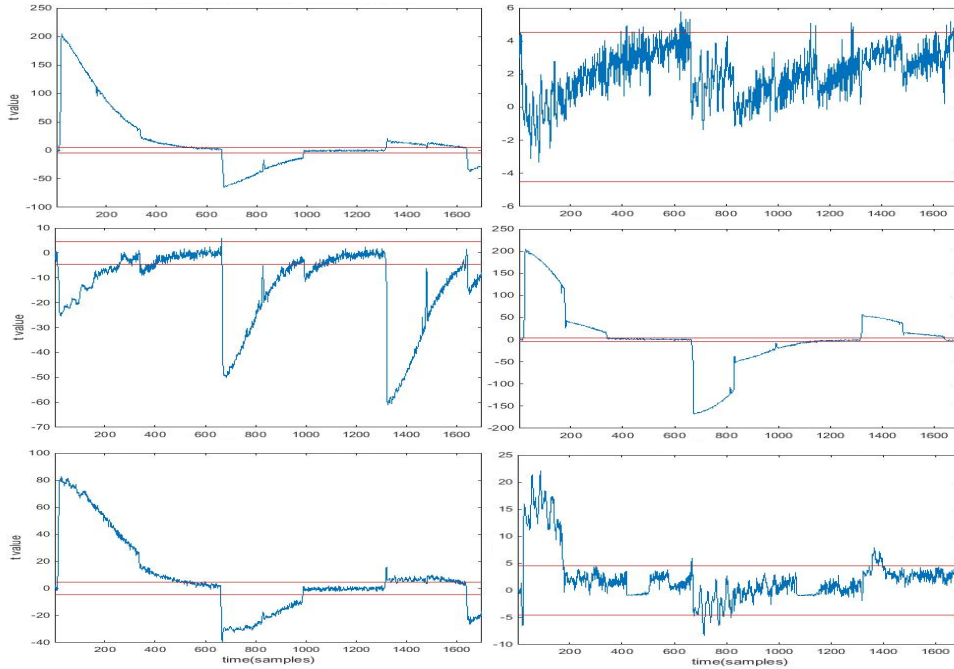


Figure 2: TVLA on first-order DOM-KECCAK implementation. Left: Masks off. Right: masks on

KECCAK- f given to different states, the first one, chosen at random and fixed through all the encryptions and the second one, randomly chosen as well, that changes for every encryption. The test determines whether it is possible to distinguish between one another and hence, find potentially exploitable leakage depending on the state. When masks are off leakage is expected, since the countermeasures are switched off. On the other hand, when masks are on we expect no first-order leakage for a first-order implementation. We ensure that the initial sharing is done properly and focus our analysis only on the first two rounds of the cipher to validate our claims. All designs analyzed in this work are done under this same conditions.

Figure 2 shows results of TVLA for 20 thousand traces when masks are off and 55 million traces when masks are on for a DOM-KECCAK implementation. In this case, although the leakage is reduced greatly, we see that the t -score goes over 4.5 for a significant number of sample points during both first and second rounds. We emphasize that we make no claims on this observation leading to a successful attack. Our goal is to make leakage assessment comparison between this implementation and our proposal which is presented in the next section.

3.2 Non-Complete Round-Based Architecture

The question that naturally arises after this analysis is the root of the non-completeness failure. Clearly, it is important to take into account the effect of the linear layers in combination with the non-linear layer within a cycle, since χ and the linear operations including θ are secure individually. A detailed observation shows the following.

Order of operations. The order of linear and non-linear operations within a round is important. If the round structure of a cryptographic algorithm follows the traditional SPN approach where the linear operation follows the non-linear operation within a round,

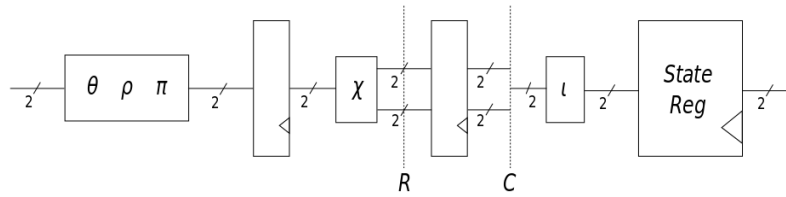


Figure 3: One round structure of the naive fix

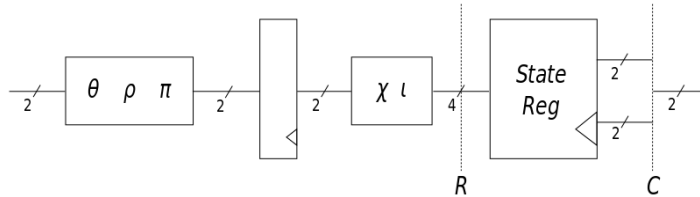


Figure 4: One round structure of our design

we would not observe this failure of non-completeness if the non-linear operation is non-complete. For higher orders, one still needs to ensure composibility, which is not the objective of this paper's discussion.

Linear transformations. The structure of KECCAK linear operations, in particular θ which shuffles and *combines* several state bits, causes the non-completeness to fail when χ is applied later. Operations ρ and π being simple wirings do not combine several bits and hence do not cause any problem.

Naive fix. Following above observations, one can clearly see that it suffices to introduce a register between the linear operations and χ to secure the design. However, this new layer of registers increases the number of cycles required per round. When combined with the register needed just right after the refreshing layer, the cycle count becomes three times bigger than the plain implementation of KECCAK- f [200] without a significant increase in maximum frequency. Fig. 3 illustrates this architecture. However, the main goal tried to achieve with a parallel implementation is a low-latency design, compared to a serial implementation that would have a higher latency but smaller area.

Our design. In order to reduce the latency, we remove the registers that come after step χ and use the state registers instead to keep the security conditions. This way the compression layer comes right after the state registers and before starting the next round computations. In this case we need two clock cycles to perform one round and hence 36 cycles for KECCAK- f [200]. Figure 4 presents this structure, with which the implementation is fulfilling non-completeness. Note that it is possible to re-use some part of the state register for synchronization after the linear layer with a careful selection of reused registers. This would reduce the area by approximately 2800GE. Here, we use two separate registers for clear analysis of the target difference.

3.2.1 Evaluation

Here we present the results for our non-complete KECCAK (NC-KECCAK) for both first- and second-order implementations, which follow the structure defined in Fig. 4. We kept

exactly the same setup as in Section 3.1.1 for comparability. In the case of the first-order secure implementation no first-order leakage appears, while in the second-order neither first- nor second-order leakage appear up to 55 million traces. Fig. 5 presents these results.

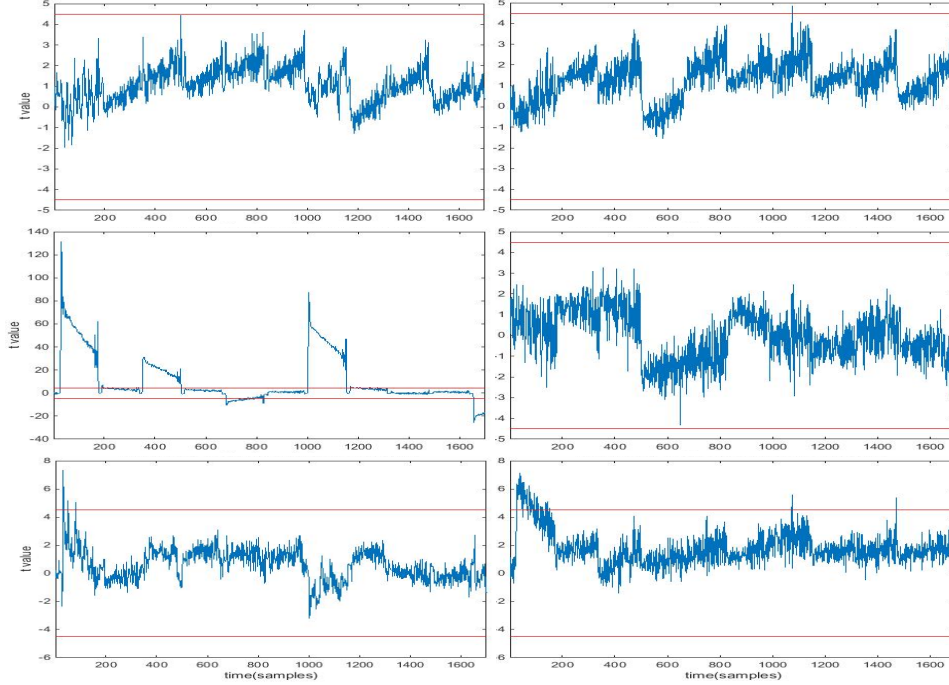


Figure 5: T-test on first (left) and second-order (right) NC-KECCAK implementations over 55M traces.

3.3 Performance Analysis

To synthesize our designs we use Synopsys Design Compiler Version I-2013.12 with NanGate 45nm Open Cell Library. The synthesis is done by using the `-compile` command and setting the flag `-exact_map` to avoid any optimization that could affect the security. The option `-no_autoungroup` is set by default when using `-compile`. This way it keeps the hierarchy of the design so that no optimizations in between modules can happen. Table 1 shows the results for this section’s implementations. Note that the number of random bits presented is for each round, i.e., for every two clock cycles, and can be reduced significantly using the ideas from [BDN⁺14, GSM17, Dae17] which is not the goal of this paper.

Table 1: Synthesis results for the different designs

DESIGN	AREA(kGE)				Rand.	Cycles	Max.Freq. (MHz)
	χ	θ	State	Total			
Plain	542.63	638.4	1 333	2 759	-	18	1 136
DOM-KECCAK 1st	9 881*	1 600	2 667	17 105	200	36	1 087
NC-KECCAK 1st	2 613	1 600	5 334*	17 493	200	36	1 300
NC-KECCAK 2nd	6 200	2 400	12 001*	35 223	600	36	1 205

* This number includes the registers layer needed before the compression

4 Speeding Up Keccak Implementations

Our goal in this section is to push the limits of low-latency masked KECCAK implementation using standard CMOS-like cells. Clearly, it is very challenging to provide a secure unrolled implementation of KECCAK that calculates more than one round in one clock cycle using a $s_{in} = d + 1$ share masking scheme. However, due to increased number of shares TI can become advantageous in this setting. In what follows, we first discuss how to implement multiple layers of nonlinear functions generically. Then, we propose a method to halve the latency of a KECCAK implementation using TI and analyze the difficulties found in the process. We conclude this section with our security analysis and synthesis results which shows the fastest secure hardware implementation published to this date.

4.1 Unrolling Quadratic Round Functions

Let \mathcal{R}_i be a quadratic round function, similar to that of KECCAK permutations. And let us initially target two rounds, $\mathcal{F} = \mathcal{R}_i \circ \mathcal{R}_{i+1}$. That is, we need to share and calculate not a quadratic, but a quartic function. According to Eqn. (2), a first-order resistant implementation of a quartic function requires $s_{in} \geq 5$ shares given the algebraic normal form (ANF) of this function. However, for KECCAK and many other schemes alike gathering and using the ANF of two rounds \mathcal{F} is not trivial due to the high diffusion of state bits. Specifically for KECCAK-f[200], it can be seen that a single output bit of \mathcal{F} depends on almost all input bits. Due to this difficulty and to avoid a case by case analysis for each target algorithm, we opt to develop a generic method.

For additional simplicity, let's assume \mathcal{R}_i is composed of a series of AND gates. The dependency for a particular output is provided in Fig. 6. It is clear that if we implement the first layer using 3 shares which is the minimum number of shares proposed for first-order security in Eqn. (2), the first layer would indeed be secure. However the second layer which uses the first layer's output can not satisfy non-completeness, i.e. every output share of \mathcal{F} would depend on all input shares of \mathcal{F} .

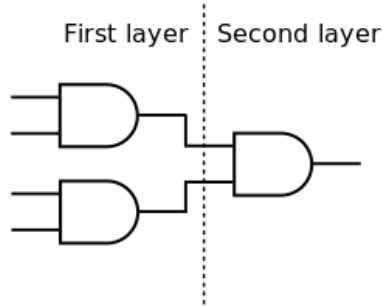


Figure 6: Quartic operation composed of two quadratic operations

We would like to achieve first-order security even at the last round of unrolled implementation. With our method, we suggest to crawl backward until the first round of \mathcal{F} and decide how-many shares we need. That is, we observe that first-order non-completeness of a quadratic function in the second layer implies combination of any two shares being independent of the secret in the input of the second layer. This implies any two share of the output of the first layer being independent of the input which is the definition of second-order non-completeness. Hence, if we implement the first and second layers such that they satisfy second- and first-order non-completeness respectively, \mathcal{F} would satisfy first-order non-completeness. This observation is naturally applicable to higher degree functions and easily extendable to more rounds.

There is still one challenge remaining: the second-order non-complete layer will increase the number of shares, hence the first order non-complete layer should be compatible to work with increased number of shares with embedded compression layer to keep the number of shares at the output of \mathcal{F} as low as possible. We call a sharing where $s_{in} > s_{out}$ a *compression sharing*.

4.2 First Attempt for Keccak

We build the second order secure sharing for \mathcal{R}_i with $s_{in} = 5$ and $s_{out} = 10$. Eqn. (6) shows the output dependencies on the input shares for an AND gate. Every combination of two outputs is independent of at least one input share.

$$\begin{aligned}
 S'_1 &= f_1(S_1, S_2) & S'_6 &= f_6(S_1, S_3) \\
 S'_2 &= f_2(S_2, S_3) & S'_7 &= f_7(S_3, S_5) \\
 S'_3 &= f_3(S_3, S_4) & S'_8 &= f_8(S_5, S_2) \\
 S'_4 &= f_4(S_4, S_5) & S'_9 &= f_9(S_2, S_4) \\
 S'_5 &= f_5(S_5, S_1) & S'_{10} &= f_{10}(S_4, S_1)
 \end{aligned} \tag{6}$$

Note that due to the linear layer of KECCAK- f , the AND gates in \mathcal{R}_{i+1} depends more than the output of two AND gates. Hence, not every first-order non-complete sharing of \mathcal{R}_{i+1} leads to a non-complete \mathcal{F} . Moreover, even though many direct $10 \rightarrow 10$ sharings of \mathcal{R}_{i+1} satisfying the above restriction can be derived following a simple procedure (every output share will combine three of the previous pairs where at most two of the initial input shares can be repeated), we would like to have a compression sharing to avoid increase of number of shares.

For a target of $10 \rightarrow 5$ compression sharing, we need to be more careful since more pairs are combined. Eqn. (7) illustrates the input-output dependency of a possible sharing for \mathcal{R}_{i+1} .

$$\begin{aligned}
 S''_1 &= f_1(S'_1, S'_2, S'_3, S'_6, S'_9, S'_{10}) && \text{(Missing input } S_5) \\
 S''_2 &= f_2(S'_1, S'_2, S'_5, S'_6, S'_7, S'_8) && \text{(Missing input } S_4) \\
 S''_3 &= f_3(S'_1, S'_4, S'_5, S'_8, S'_9, S'_{10}) && \text{(Missing input } S_3) \\
 S''_4 &= f_4(S'_3, S'_4, S'_5, S'_6, S'_7, S'_{10}) && \text{(Missing input } S_2) \\
 S''_5 &= f_5(S'_2, S'_3, S'_4, S'_7, S'_8, S'_9) && \text{(Missing input } S_1)
 \end{aligned} \tag{7}$$

4.3 Our Design

The greatest area contributor of a KECCAK parallel implementation, is the χ step. The number of gates in the non-linear operation scales with the number of input shares as follows:

- Number of AND gates = $\binom{s_{in}}{2} + s_{in}$
- Number of XOR gates = $\binom{s_{in}}{2} + s_{in}$
- Number of NOT gates = s_{in}

The area overhead of increasing the number of shares to 10 is significant, so we decided to look for another sharing for the first layer that produces less outputs. One possible

option is the $6 \rightarrow 7$ sharing proposed in [BGN⁺14a]. Our further investigations lead to $6 \rightarrow 6$ sharing schemes that we can use for both layers, which are shown in Eqn. (8).

$$\begin{aligned}
S'_1 &= f_1(S_1, S_2, S_3) & S''_1 &= f_1(S'_4, S'_5, S'_6) \quad (\text{Missing input } S_1) \\
S'_2 &= f_2(S_1, S_4, S_5) & S''_2 &= f_2(S'_2, S'_3, S'_5) \quad (\text{Missing input } S_2) \\
S'_3 &= f_3(S_1, S_4, S_6) & S''_3 &= f_3(S'_2, S'_3, S'_4) \quad (\text{Missing input } S_3) \\
S'_4 &= f_4(S_2, S_5, S_6) & S''_4 &= f_4(S'_1, S'_4, S'_5) \quad (\text{Missing input } S_4) \\
S'_5 &= f_5(S_3, S_5, S_6) & S''_5 &= f_5(S'_1, S'_3, S'_6) \quad (\text{Missing input } S_5) \\
S'_6 &= f_6(S_2, S_3, S_4) & S''_6 &= f_6(S'_1, S'_2, S'_6) \quad (\text{Missing input } S_6)
\end{aligned} \tag{8}$$

With this sharing we are able to reduce the area considerably without failing non-completeness.

4.3.1 Performance Analysis

The implementation is done in parallel by concatenating two rounds, i.e. the round which is depicted with a box in Fig. 1 is repeated twice before synchronization. We present the results for the two sharing schemes mentioned above in Table 2 for comparison.

Table 2: Results for the different double round sharings which require 9 cycles and no extra randomness for refreshing

DESIGN	AREA(kGE)					State	Total	Max.Freq. (MHz)
	χ_1	χ_2	θ_1	θ_2				
$5 \rightarrow 10 \rightarrow 5$	13 533	59 893	4 280	9 747	6 700	99 347	395.25	
$6 \rightarrow 6 \rightarrow 6$	22 400	22 160	5 662	5 662	8 033	70 120	436.7	

The first non-linear operation in $5 \rightarrow 10 \rightarrow 5$ sharing is a bit cheaper than the one using $6 \rightarrow 6 \rightarrow 6$. However, the second non-linear operation of $5 \rightarrow 10 \rightarrow 5$ is significantly more expensive due to the greater number of inputs as expected. In the $6 \rightarrow 6 \rightarrow 6$ scheme the state register is a bit bigger since there is one more share to store. All things considered, the area reduction of our design compared to a first attempt is clear.

Given the results, it is possible to see that the time needed for a single encryption is $20.61ns$. Therefore we can affirm this is the fastest round based KECCAK implementation published to this date.

4.3.2 Evaluation

We evaluate the optimized design using the $6 \rightarrow 6 \rightarrow 6$ sharing. The test is done under the conditions already presented in Section 3.1.1. Since this implementation is much faster than the previous ones, the analysis covers 10 rounds of the cipher.

The results are presented in Fig. 7 for masks off and masks on. It is possible to appreciate that the countermeasures indeed prevent leakage from appearing in the first-order. Moreover, we observe no leakage in the second or in the third order t-test using 55 million measurements. We attribute this to the noise introduced by the large number of shares used.

5 Conclusions

In this work we presented, on the one hand, a flaw in previous round based KECCAK secure implementations on hardware and how to address this issue. We propose this solution

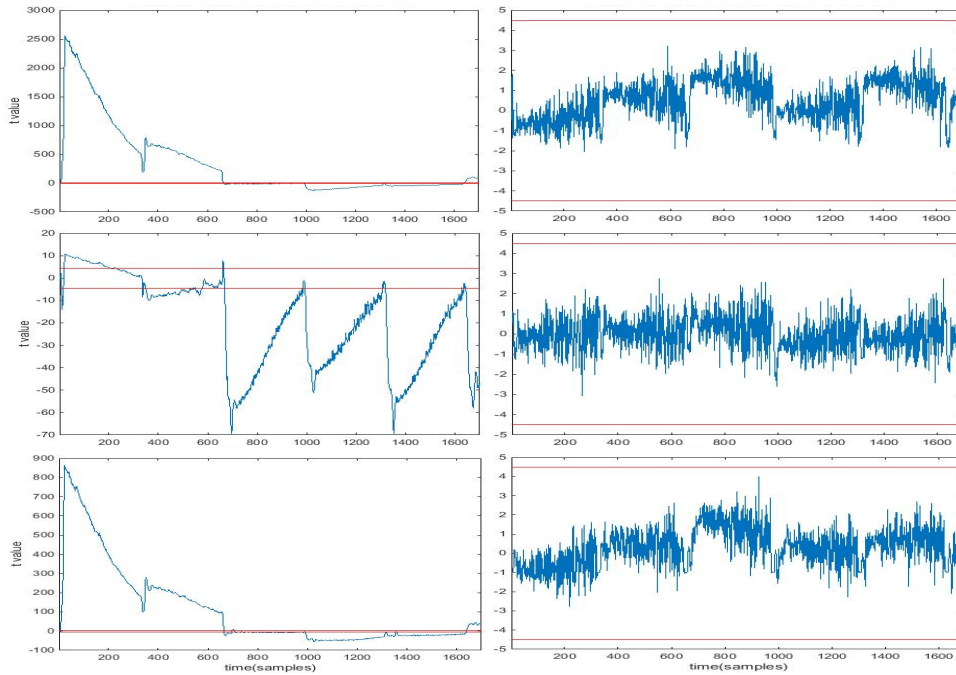


Figure 7: T-test on first order $d + 1$ KECCAK implementation. Left: Masks off. Right: DOM-KECCAK masks on

for KECCAK permutations, but it is also applicable to other algorithms that concatenate several linear and non-linear operations. On the other hand, we introduced a method to speed up masked hardware implementations. Thus, by applying this method, the fastest KECCAK implementation is presented.

References

- [BDN⁺14] B. Bilgin, J. Daemen, V. Nikov, S. Nikova, V. Rijmen, and G. Van Assche. Efficient and first-order DPA resistant implementations of keccak. in CARDIS, volume 8419 of LNCS pp 187-199, June 2014.
- [BDP⁺16a] G. Bertoni, J. Daemen, M. Peeters, G. Van Assche, and R. Van Keer. Caesar submission: Ketje v2, September 2016.
- [BDP⁺16b] G. Bertoni, J. Daemen, M. Peeters, G. Van Assche, and R. Van Keer. Caesar submission: Keyak v2, September 2016.
- [BDPA10a] G. Bertoni, J. Daemen, M. Peeters, and G. Van Assche. Building power analysis resistant implementations of keccak. Second SHA-3 candidate conference, August 2010.
- [BDPA10b] G. Bertoni, J. Daemen, M. Peeters, and G. Van Assche. The keccak reference. <http://http://keccak.noekeon.org/>, January 2010.
- [BGN⁺14a] B. Bilgin, B. Gierlichs, S. Nikova, V. Nikov, and V. Rijmen. Higher-order threshold implementations. In ASIACRYPT, volume 8874 of LNCS, pages 326-343. Springer, 2014.

- [BGN⁺14b] B. Bilgin, B. Gierlichs, S. Nikova, V. Nikov, and V. Rijmen. A more efficient aes threshold implementation. In D. Pointcheval and Damien Vergnaud, editors, *Progress in Cryptology-AFRICACRYPT 2014*, volume 8469 of *Lecture Notes in Computer Science*, pages 267–284. Springer International Publishing, 2014.
- [BGN⁺15] B. Bilgin, B. Gierlichs, S. Nikova, V. Nikov, and V. Rijmen. Trade-offs for threshold implementations illustrated on AES. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, 34(7):1188–1200, July 2015.
- [BNN⁺12] B. Bilgin, S. Nikova, V. Nikov, V. Rijmen, and G. StÅijtz. Threshold implementations of all 3×3 and 4×4 s-boxes. In Emmanuel Prouff and Patrick Schaumont, editors, *Cryptographic Hardware and Embedded Systems-CHES 2012*, volume 7428 of *Lecture Notes in Computer Science*, pages 76–91. Springer Berlin Heidelberg, 2012.
- [BNN⁺14] B. Bilgin, S. Nikova, V. Nikov, V. Rijmen, N. Tokareva, and V. Vitkup. Threshold implementations of small s-boxes. *Cryptography and Communications*, pages 1–31, 2014.
- [CBR⁺15] T. De Cnudde, B. Bilgin, O. Reparaz, V. Nikov, and S. Nikova. Higher-order threshold implementation of the AES s-box. In Naofumi Homma and Marcel Medwed, editors, *Smart Card Research and Advanced Applications - 14th International Conference, CARDIS 2015, Bochum, Germany, November 4-6, 2015. Revised Selected Papers*, volume 9514 of *Lecture Notes in Computer Science*, pages 259–272. Springer, 2015.
- [CMG⁺13] J. Cooper, E. De Mulder, G. Goodwill, J. Jaffe, G. Kenworthy, and P. Rohatgi. Test vector leakage assessment (TVLA) methodology in practice. International Cryptographic Module Conference, 2013. <http://icmc-2013.org/wp/wp-content/uploads/2013/09/goodwillkenworthtestvector.pdf>.
- [Dae17] J. Daemen. Changing of the guards: A simple and efficient method for achieving uniformity in threshold sharing. In Wieland Fischer and Naofumi Homma, editors, *Cryptographic Hardware and Embedded Systems - CHES 2017 - 19th International Conference, Taipei, Taiwan, September 25-28, 2017, Proceedings*, volume 10529 of *Lecture Notes in Computer Science*, pages 137–153. Springer, 2017.
- [DFS15] A. Duc, S. Faust, and F.-X. Standaert. *Making Masking Security Proofs Concrete*, pages 401–429. Springer Berlin Heidelberg, Berlin, Heidelberg, 2015.
- [FGP⁺17] S. Faust, V. Grosso, S. Merino Del Pozo, C. Paglialonga, and F.-X. Standaert. Composable masking schemes in the presence of physical defaults and the robust probing model. Cryptology ePrint Archive, Report 2017/711, 2017. <http://eprint.iacr.org/2017/711>.
- [GMK16] H. Gross, S. Mangard, and T. Korak. Domain-oriented masking: Compact masked hardware implementations with arbitrary protection order. Cryptology ePrint Archive, Report 2016/486, 2016. <http://eprint.iacr.org/2016/486>.
- [GMO01] Karine Gandolfi, Christophe Mourtel, and Francis Olivier. Electromagnetic analysis: Concrete results. In Çetin K. Koç, David Naccache, and Christof

- Paar, editors, *Cryptographic Hardware and Embedded Systems-CHES 2001*, volume 2162 of *LNCS*, pages 251–261. Springer Berlin Heidelberg, 2001.
- [GSM17] H. Gross, D. Schaffenrath, and S. Mangard. Higher-order side-channel protected implementations of keccak. In *2017 Euromicro Conference on Digital System Design (DSD)*, pages 205–212, Aug 2017.
- [ISW03] Y. Ishai, A. Sahai, and D. Wagner. *Private Circuits: Securing Hardware against Probing Attacks*, pages 463–481. Springer Berlin Heidelberg, Berlin, Heidelberg, 2003.
- [KJJ99] P. C. Kocher, J. Jaffe, and B. Jun. Differential power analysis. In *Advances in Cryptology - CRYPTO '99, 19th Annual International Cryptology Conference, Santa Barbara, California, USA, August 15-19, 1999, Proceedings*, pages 388–397, 1999.
- [NRR06] S. Nikova, C. Rechberger, and V. Rijmen. Threshold implementations against side-channel attacks and glitches. In *ICICS*, volume 4307 of *LNCS*, pages 529–545. Springer, 2006.
- [NRS08] S. Nikova, V. Rijmen, and M. Schlaffer. Secure hardware implementation of non-linear functions in the presence of glitches. In *ICISC*, volume 5461 of *LNCS*, pages 218–234. Springer, 2008.
- [NRS11] S. Nikova, V. Rijmen, and M. Schlaffer. Secure hardware implementation of non-linear functions in the presence of glitches. *J. Cryptology*, 24(2):292–321, 2011.
- [RBN⁺15] O. Reparaz, B. Bilgin, S. Nikova, B. Gierlichs, and I. Verbauwhede. *Consolidating Masking Schemes*, pages 764–783. Springer Berlin Heidelberg, Berlin, Heidelberg, 2015.
- [Rom14] C. H. Romine. SHA-3 standard: Permutation-based hash and extendable output functions. 2014.

A Appendix

A.1 Keccak Round Steps

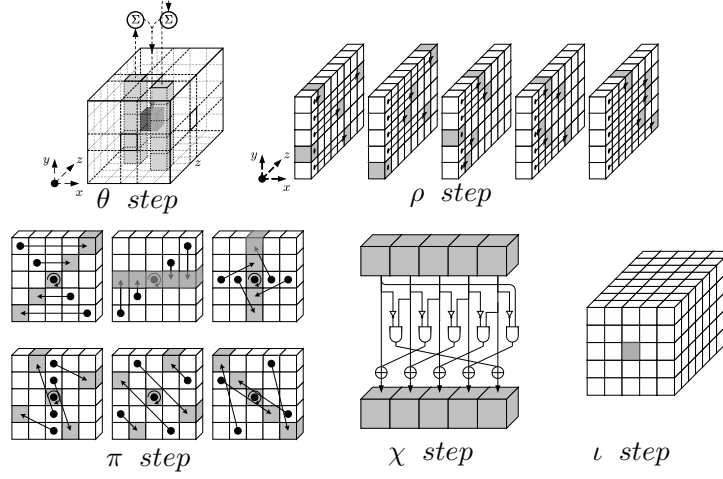


Figure 8: KECCAK- $f[200]$ state and steps [BDPA10b].

A.2 Second-order Masking

A.2.1 Threshold Implementation (TI)

$$\begin{aligned}
 z_1 &= a_1 \oplus b_1c_1 \oplus b_1c_5 \oplus b_5c_1 & z_2 &= a_2 \oplus b_2c_2 \oplus b_1c_2 \oplus b_2c_1 \\
 z_3 &= a_3 \oplus b_3c_3 \oplus b_1c_3 \oplus b_3c_1 & z_4 &= a_4 \oplus b_4c_4 \oplus b_1c_4 \oplus b_4c_1 \\
 z_5 &= a_5 \oplus b_5c_5 \oplus b_2c_5 \oplus b_5c_2 & z_6 &= b_2c_3 \oplus b_3c_2 \\
 z_7 &= b_2c_4 \oplus b_4c_2 & z_8 &= b_3c_4 \oplus b_4c_3 \\
 z_9 &= b_3c_5 \oplus b_5c_3 & z_{10} &= b_4c_5 \oplus b_5c_4
 \end{aligned} \tag{9}$$

A.2.2 Domain Oriented Masking (DOM)

$$\begin{aligned}
 z_1 &= (a_1b_1) \oplus [a_1b_2 \oplus r_1] \oplus [a_1b_3 \oplus r_2], \\
 z_2 &= [a_2b_1 \oplus r_1] \oplus (a_2b_2) \oplus [a_2b_3 \oplus r_3], \\
 z_3 &= [a_3b_1 \oplus r_2] \oplus [a_3b_2 \oplus r_3] \oplus (a_3b_3).
 \end{aligned} \tag{10}$$

A.3 Non-completeness Failure in Round-Based DOM Implementation

A.3.1 First Order.

Fig. 10 presents graphically what is demonstrated in Eqn.6 numerically:

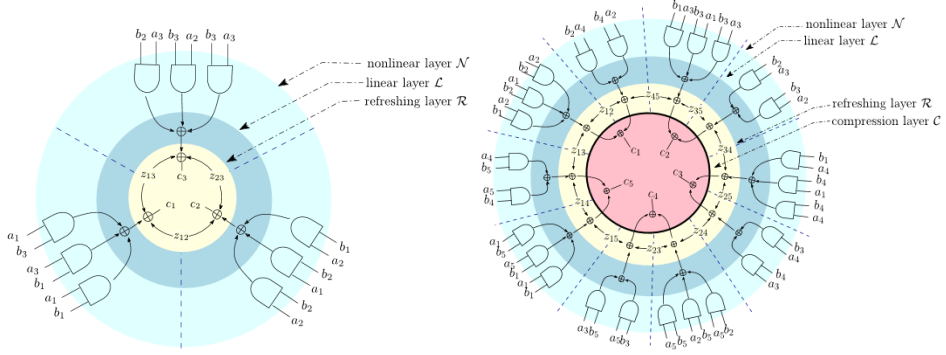


Figure 9: First- and second-order threshold implementation with randomness.

A.3.2 Second Order.

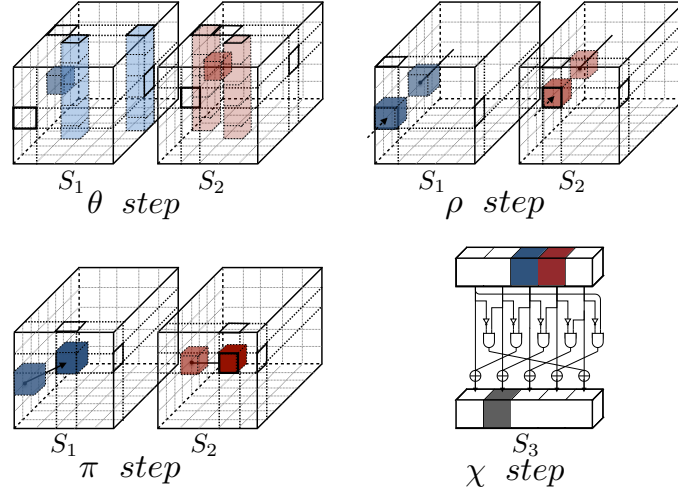
First bit

$$\chi^{-1} : S_2^X [0, 0, 7] \rightarrow S_2^\pi [0, 0, 7] \oplus S_2^\pi [1, 0, 7] S_2^\pi [2, 0, 7]$$

$$\begin{aligned} \pi^{-1} : S_2^\pi [0, 0, 7] &\rightarrow S_2^\rho [0, 0, 7] \\ S_2^\pi [1, 0, 7] &\rightarrow S_2^\rho [1, 1, 7] \\ S_2^\pi [2, 0, 7] &\rightarrow S_2^\rho [2, 2, 7] \end{aligned}$$

$$\begin{aligned} \rho^{-1} : S_2^\rho [0, 0, 7] &\rightarrow S_2^\theta [0, 0, 7] \\ S_2^\rho [1, 1, 7] &\rightarrow S_2^\theta [1, 1, 3] \\ S_2^\rho [2, 2, 7] &\rightarrow S_2^\theta [2, 2, 4] \end{aligned}$$

$$\begin{aligned} \theta_1^{-1} : S_2^\theta [0, 0, 7] &\rightarrow S_2 [0, 0, 7] \oplus \bigoplus_{y'=0}^4 S_2 [4, y', 7] \oplus \bigoplus_{y'=0}^4 S_2 [1, y', 6] \\ S_2^\theta [1, 1, 3] &\rightarrow S_2 [1, 1, 3] \oplus \bigoplus_{y'=0}^4 S_2 [0, y', 3] \oplus \sum_{y'=0}^4 S_2 [2, y', 2] \\ S_2^\theta [2, 2, 4] &\rightarrow S_2 [2, 2, 4] \oplus \bigoplus_{y'=0}^4 S_2 [1, y', 4] \oplus \sum_{y'=0}^4 S_2 [3, y', 3] \end{aligned}$$

Figure 10: Tracing back output bit $C^x [4, 1, 0]$.**Second bit**

$$\chi^{-1} : S_6^X [2, 0, 5] \rightarrow S_1^\pi [4, 0, 5] S_3^\pi [3, 0, 5]$$

$$\pi^{-1} : S_1^\pi [4, 0, 5] \rightarrow S_1^\rho [4, 4, 5]$$

$$S_3^\pi [3, 0, 5] \rightarrow S_3^\rho [3, 3, 5]$$

$$\rho^{-1} : S_1^\rho [4, 4, 5] \rightarrow S_1^\theta [4, 4, 7]$$

$$S_3^\rho [3, 3, 5] \rightarrow S_3^\theta [3, 3, 0]$$

$$\theta_1^{-1} : S_1^\theta [4, 4, 7] \rightarrow S_1 [4, 4, 7] \oplus \bigoplus_{y'=0}^4 S_1 [3, y', 7] \oplus \bigoplus_{y'=0}^4 S_1 [0, y', 6]$$

$$S_3^\theta [3, 3, 0] \rightarrow S_3 [3, 3, 0] \oplus \bigoplus_{y'=0}^4 S_3 [2, y', 0] \oplus \sum_{y'=0}^4 S_3 [4, y', 7]$$