






# Low-Latency Keccak at any Arbitrary Order

Sara Zarei<sup>1</sup> , Aein Rezaei Shahmirzadi<sup>2</sup> , Hadi Soleimany<sup>1</sup> ,  
Raziye Salarifard<sup>3</sup>  and Amir Moradi<sup>2</sup> 

<sup>1</sup> Shahid Beheshti University, Cyber Research Center, Tehran, Iran  
[sarazareei.94@gmail.com](mailto:sarazareei.94@gmail.com), [h\\_soleimany@sbu.ac.ir](mailto:h_soleimany@sbu.ac.ir)

<sup>2</sup> Ruhr University Bochum, Horst Görtz Institute for IT Security, Bochum, Germany  
[firstname.lastname@rub.de](mailto:firstname.lastname@rub.de)

<sup>3</sup> Shahid Beheshti University, Faculty of Computer Science and Engineering, Tehran, Iran  
[r\\_salarifard@sbu.ac.ir](mailto:r_salarifard@sbu.ac.ir)

**Abstract.** Correct application of masking on hardware implementation of cryptographic primitives necessitates the instantiation of registers in order to achieve the non-completeness (commonly said to stop the propagation of glitches). This sometimes leads to a high latency overhead, making the implementation not necessarily suitable for the underlying application. As a concrete example, this holds for Keccak. Application of  $d + 1$  Domain Oriented Masking (DOM) on a round-based implementation of Keccak leads to the introduction of two register stages per round, i.e., two times higher latency. On the other hand, Rhythmic-Keccak, introduced in CHES 2018, unrolls two rounds to half the latency compared to an unprotected ordinary round-based implementation. To that end,  $td + 1$  masking is used which requires a notable area, and – apart from the difficulty to construct – its extension to higher orders seems beyond the bounds of feasibility.

In this paper, we focus on  $d + 1$  masking and introduce a methodology which enables us to stay with the latency of an unprotected round-based implementation, i.e., one register stage per round. While being secure under glitch-extended probing model, we provide a general design where the desired security order can be easily adjusted without any effect on the above-given latency. Compared to the Rhythmic-Keccak, the synthesis results show that our first-order design is able to accomplish the entire operations of Keccak- $f$ [200] in the same period of time while decreasing the area by 74.5%. Notably, our implementations achieve around 30% less delay compared to the corresponding original DOM-Keccak designs.

**Keywords:** Keccak · Masking · Threshold Implementation · Domain-Oriented Masking · Hardware Implementation · Low Latency

## 1 Introduction

Side-Channel Analysis (SCA) attacks are a devastating class of threats for many security-critical devices, which can reveal sensitive information with a high success rate and low costs. These kinds of physical attacks exploit the information gained from the cryptographic algorithm implemented on the target device, such as power consumption, electromagnetic radiation, and timing information. This highlights the importance of the physical security of cryptographic devices, even though their underlying primitives are mathematically secure. One of the prominent and distinct approaches is Differential Power Analysis (DPA). It exploits the dependencies between the power consumption of the victim’s device and the processed intermediate values to reveal the secret key [KJJ99]. As every electronic device consumes power to operate, unprotected implementations are highly susceptible to DPA attacks. After its official publication [KJJ99], the scientific community has

dedicated a considerable body of research to improve SCA attacks. As a result, there are a wide range of more sophisticated attacks targeting both software and hardware implementations like Correlation Power Analysis (CPA) [BCO04], Mutual Information Analysis (MIA) [GBTP08], and Moments-Correlating DPA (MC-DPA) [MS16]. Hence, it is necessary to integrate sound countermeasures to protect the implementations of cryptographic primitives against SCA.

Compared to other ad-hoc or circuit-level countermeasures, masking schemes have attracted the most attention from both academia and industry, due to their sound theoretical basis. Masking schemes intend to make the physical properties of the target device independent of the processed sensitive data. To this end, sensitive intermediate values are randomized by means of secret sharing. Namely, a sensitive variable is split into a certain number of independent shares so that an adversary can reproduce the original sensitive variable if and only if he can achieve all shares of the variable. Various masking schemes have been proposed in the literature to mitigate SCA attacks. Seminal contributions have been made by Ishai et al. [ISW03] and Trichina [Tri03], where a first-order secure AND gate is presented. However, it has been demonstrated that these schemes fail to provide security in hardware implementations, given that different sources of leakage exist in hardware. The reason behind this is a known fact in hardware platforms called glitches, which is related to the propagation delay patterns of CMOS circuits. Extensive studies have been devoted over the last years to introduce hardware-oriented masking designs that are secure in the presence of glitches. Nikova et al. proposed an implementation strategy called Threshold Implementation (TI) [NRS11], which can provide protection against SCA attacks in presence of glitches. The authors introduced a design methodology and some properties to meet in order to make an implementation provably secure, considering the physical defaults in hardware platforms. The scheme was initially used to fulfill first-order security and successfully applied to several algorithms [PMK<sup>+</sup>11, MPL<sup>+</sup>11]. Later, it was generalized to higher-orders by Bilgin et al. [BGN<sup>+</sup>14], while its limitations have been addressed in [Rep15].

Several hardware masking schemes have been introduced [RBN<sup>+</sup>15, GMK16], requiring only  $d + 1$  shares to achieve  $d^{\text{th}}$ -order security, in contrast to TI which requires  $td + 1$  shares to achieve the same order of security for a function with the algebraic degree  $t$ . In TI, especially for first-order security, there can exist realizations where no fresh randomness is needed. However, fresh randomness should usually be applied to construct a secure  $d + 1$  masked implementation, e.g., as illustrated in [GMK16] by introducing DOM. Further, in [RBN<sup>+</sup>15], it has been shown that the first-order 2-share secure masked realization of all six 4-bit quadratic bijections (these six classes are presented in [BNN<sup>+</sup>15]) can be achieved without using any fresh masks. Later, the authors of [SM21] have presented a technique allowing them to provide first-order secure implementation of various ciphers (up to cubic function) with no fresh randomness. Despite the significant progress in providing provable first-order masked hardware implementations, their generalization to higher order appeared to be challenging. In classical TI, the required number of  $td + 1$  inputs shares might be unaffordable in small embedded devices as the implementation cost increases significantly for higher orders.  $d + 1$  schemes like DOM mitigate such a demand allowing to obtain more efficient constructions at higher orders, which seems to be increasingly important considering the need for protected implementation of standard cryptographic primitives (such as AES and Keccak).

Keccak is a family of flexible cryptographic primitives based on the sponge construction with variable input size and arbitrary output size [BPVA<sup>+</sup>11, BDPA13]. In 2012, the National Institute of Standards and Technology (NIST) announced that the Secure Hash Algorithm-3 (SHA-3) will be standardized based on Keccak. The specifications of the SHA-3 hash standard are published in FIPS PUB 202 [Dwo15]. Besides, Keccak can be used to build a wide range of cryptographic primitives including those which take a secret

key as an argument which makes it relevant to SCA attacks. Below, we briefly review the previous works regarding the masked hardware realizations of Keccak.

## 1.1 Related Works

A handful of first-order implementations of Keccak have been introduced in the literature while the number of works considering higher orders is quite limited. The first TI of Keccak was proposed by the original designers [BDPVA10]. The implementation was claimed to accomplish the first-order security employing three shares. Bilgin et al. [BDN<sup>+</sup>13] showed that such a design neglects the uniform sharing of the Keccak  $\chi$  function, so that a mandatory condition for security is not satisfied. An elementary fix is to refresh the output sharing, known as re-masking [BDN<sup>+</sup>13]. Hence, the masked 5-bit S-box ( $\chi$ ) requires 10 fresh random bits per S-box evaluation (per clock cycle), which is a considerable number when taking the whole Keccak state into account. As an alternative, the authors of [BDN<sup>+</sup>13] made use of specific properties of  $\chi$  to decrease the demand for fresh random bits to 4 bits per S-box. The other solution they provided to fulfill the uniformity was to use a distinctive construction making use of four shares, hence naturally a higher area overhead, but no need to inject any fresh randomness.

Daemen pursued another direction aiming to achieve uniformity for the entire nonlinear layer instead of satisfying uniformity for the individual S-boxes. In a nutshell, he proposes to use the shares of the neighbor S-box(es) for re-masking the output of a 3-share (non-uniform) S-box [Dae17]. His new mechanism called “Changing of the Guards” can construct uniform TI for the entire layer of bijective S-boxes. This method was later applied to AES [WM18], KETJE [ANR19], ASCON and Keyac [SD17] to achieve uniformity in first-order secure designs.

Gross et al. presented DOM implementation of Keccak [GSM17a]. Their design is extendable to higher orders more straightforwardly than the previous ones and also has a smaller area overhead. It uses  $d + 1$  shares,  $5d(d + 1)/2$  fresh randomness for each instance of  $\chi$ , and two register stages for a  $d^{\text{th}}$ -order secure implementation. Later, Arribas et al. [ABP<sup>+</sup>18] indicated a flaw in such a design violating the essential property of non-completeness, hence not achieving the desired security level. Based on this observations, the design of Gross et al. was then updated in [GSM17b] to address the reported issue. The authors of [ABP<sup>+</sup>18] also proposed a two-round unrolled architecture to construct first-order TI of Keccak, which accomplishes two Keccak rounds in one clock cycle, resulting in a decreased overall latency.

## 1.2 Our Contributions

Latency and area are two critical criteria to evaluate the efficiency of masked hardware implementations. These designs inevitably have to utilize extra registers to prevent glitch propagation, which is essential to fulfill non-completeness. Doing so increases both the latency and area of the resulting implementations. The complication intensifies when it comes to higher orders, that usually leads to an exponential increase in the area overhead due to a rapid increase in the number of input shares.

With respect to this fact, we aim at constructing a compact low-latency masked implementation of Keccak. Our design is generic, i.e., extendable to any arbitrary protection order, while being secure under glitch-extended probing model. More precisely, our construction is based on DOM, in which the minimum number of  $d + 1$  input shares is employed. However, the challenge to tackle is to keep the latency as low as an unprotected implementation. The challenges correspond to the critical security concerns related to the possible share dependencies that may occur in the absence of register stages. We overcome this challenge by introducing an alternative approach that benefits from the Keccak intrinsic specification. In short, our low-latency round-based masked implementation of

Keccak uses  $d + 1$  shares (for  $d^{\text{th}}$ -order provable security) and requires one clock cycle per round. Compared to state of the art, it is the only design with such a low latency at higher orders while keeping the number of shares at minimum, leading to smaller area overhead. The security of our new approach and the generic design is verified in two ways: first, using the recently-introduced verification tool SILVER [KSM20] under the glitch-extended probing model, and second, by FPGA-based experimental evaluations, namely the Welch's t-test.

### 1.3 Outline

The rest of this paper is organized as follows. Section 2 gives a brief description of Keccak and a short introduction to TI and DOM schemes, as well as a review on probing security notion. In Section 3, we first summarize the existing low-latency masked implementation of Keccak and analyze its limitations to achieve higher-order security. Subsequently, we demonstrate our methodology to build a generic low-latency realization of Keccak. Section 4, reports the performance figure of our constructions and compares them with the related works. Then, it presents the result of security evaluations, and finally, in Section 5, we conclude this paper by summarizing our conducted research.

## 2 Preliminaries

This section provides some concepts needed throughout this paper. It starts with a short description of Keccak followed by an overview of TI and DOM schemes. A brief explanation about the probing security notion is also given, which is a common approach in evaluation of masking schemes.

### 2.1 Keccak Algorithm

Keccak is a family of cryptographic hash functions that follows the sponge construction [BPVA<sup>+</sup>11, BDPA13]. The underlying permutation is denoted by Keccak- $f[b]$ , where  $b$  is referred to the state size. The parameter  $b$  is equal to  $25 \times W$ , where  $W = 2^L$  and  $0 \leq L \leq 6$ . Thus, the state size  $b$  is chosen from a set of seven values, i.e.,  $b \in \{25, 50, 100, 200, 400, 800, 1600\}$ . The state is represented by a 3-dimensional  $5 \times 5 \times W$  array, which is denoted as  $A[5][5][W]$ . Every bit of the state at position  $(x, y, z)$  is represented by  $A[x][y][z]$ . The structure of the permutations follows a so-called *Matryoshka* principle where the security analysis of Keccak with small sizes can be transferred to the instances with larger size and vice-versa. The number of rounds is determined by the state size  $b$  and can be calculated as  $n_r = 12 + 2L$  where  $L = \log_2(\frac{b}{25})$ .

Each round consists of five steps:  $R = \iota \circ \chi \circ \pi \circ \rho \circ \theta$ , where  $\chi$  is the only non-linear operation with the algebraic degree two. The linear operations  $\theta$ ,  $\rho$ , and  $\pi$  cause diffusion in different directions while  $\iota$  adds a constant value to the state, depending on the round number. Equation (1) describes the such five operations in which all additions and multiplications are performed in  $GF(2)$ .

$$\begin{aligned} \theta : A[x, y, z] &\leftarrow A[x, y, z] \oplus \bigoplus_{y'=0}^4 A[x-1, y', z] \oplus \bigoplus_{y'=0}^4 A[x+1, y', z-1], \\ \rho : A[x, y, z] &\leftarrow A\left[x, y, z - \frac{(t+1)(t+2)}{2}\right]; \\ &\begin{cases} \text{if } x = y = 0 \rightarrow t = -1 \\ \text{else } 0 \leq t < 24 \text{ and } \begin{pmatrix} 0 & 1 \\ 2 & 3 \end{pmatrix}^t \begin{pmatrix} 1 \\ 0 \end{pmatrix} = \begin{pmatrix} x \\ y \end{pmatrix} \text{ in } GF(5)^{2 \times 2} \end{cases} \quad (1) \end{aligned}$$

$$\pi : A[x, y] \leftarrow A[x', y'] : \begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} 0 & 1 \\ 2 & 3 \end{pmatrix} \begin{pmatrix} x' \\ y' \end{pmatrix},$$

$$\chi : A[x] \leftarrow A[x] \oplus (A[x+1] \oplus 1)A[x+2],$$

$$\iota : A \leftarrow A \oplus RC[i_r], \text{ where } RC[i_r] \text{ is the round constant in round } i_r.$$

The notations are in line with those used in [ABP<sup>+</sup>18]. For more details on Keccak we refer the reader to the original articles [BPVA<sup>+</sup>11, BDPA13].

## 2.2 Masking Schemes

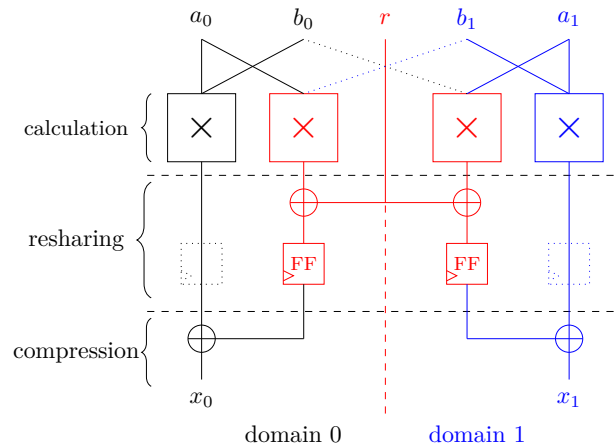
In masking schemes, each sensitive variable  $x$  is split into several shares. The minimum number of input shares should be at least  $d + 1$  to achieve  $d^{\text{th}}$ -order security. In such a setting, observing any set of  $d$  wires should not reveal any information about the variable  $x$ .

Among all proposed methodologies, Boolean masking is the most-studied and most-widely applied countermeasures against SCA attacks. Our focus in this paper is also on Boolean masking in which the value  $x$  is divided to  $s$  shares as  $x = x_0 \oplus x_2 \oplus \dots \oplus x_{s-1}$  where  $s - 1$  variables  $(x_0, x_2, \dots, x_{s-1})$  are drawn from a uniform distribution at random and the variable  $x_s$  is computed as  $x_s = x_0 \oplus \dots \oplus x_{s-2} \oplus x$ . The application of Boolean masking on linear functions (over  $GF(2)$ ) is simple as the same function can be applied on each set of shares individually. However, the challenging part is how to realize a masked variant of non-linear functions. It becomes more challenging when the algebraic degree of the given function gets larger or higher order of security is desired.

### 2.2.1 Threshold Implementation

TI [NRS11] is the first implementation strategy with immunity against a common phenomenon in CMOS technology called glitches. Let us consider the target function  $f(X) = Y$  with arbitrary input and output size, but the same number of input and output shares (for the sake of simplicity,  $s$  shares). In TI, the target function  $f(\cdot)$  is split into a set of so-called component functions  $f_{i \in \{0, \dots, s-1\}}$ . The masked realization of  $f(\cdot)$  should provide the shared output  $\mathbf{Y} = \langle Y_0, \dots, Y_{s-1} \rangle$  by receiving the input shares  $\mathbf{X} = \langle X_0, \dots, X_{s-1} \rangle$  such that the following properties are fulfilled.

- **Correctness:** This property guarantees that the masked form of the function operates correctly on all possible input shares. Namely, it implies that  $Y = \bigoplus_{\forall i} Y_i = \bigoplus_{\forall i} f_i(\mathbf{X})$  for all  $\mathbf{X}$  satisfies  $Y = f(X = \bigoplus_{\forall i} X_i)$ .
- **Non-completeness:** To resist against a  $d^{\text{th}}$ -order SCA attack, the masking of the target function should be a  $d^{\text{th}}$ -order non-complete. It means that each  $d$  (or less) combination of component functions  $f_{i \in \{0, \dots, s-1\}}$  should be independent of at least one input share.



**Figure 1:** First-order DOM-indep multiplier.

- **Uniformity:** It implies that shared output  $\mathbf{Y}$  should not be distinguishable from  $Y$  being uniformly shared, provided that the shared input  $\mathbf{X}$  is a uniform sharing of  $X$ . Note that the output of a masked function is usually used as the input of other masked functions. Hence, not satisfying this property may cause leakage in subsequent functions.

A strategy called *direct sharing* has been introduced in [NRS11] to trivially achieve non-completeness in which the number of input shares  $s_{in}$  and output shares  $s_{out}$  can be calculated as

$$s_{in} \geq td + 1, \quad s_{out} \geq \binom{s_{in}}{t}, \quad (2)$$

where  $t$  and  $d$  stand for the algebraic degree of the target function and the desired security order, respectively. However, achieving uniformity is usually less trivial and more challenging as there is no solid methodology to fulfill it except *remasking* in which fresh masks are used to refresh the sharing. For instance, a uniform TI of a simple 2-input AND gate [NRS11] or  $\chi$  function of Keccak without remasking has not been reported yet [BDN<sup>+</sup>13].

### 2.2.2 Domain Oriented Masking

DOM [GMK16] attains  $d^{th}$ -order security in hardware as well but using  $d + 1$  input shares compared to  $td + 1$  in TI. According to the DOM scheme, the component functions and shares are divided into  $d + 1$  independent sets, called *domains*. The main idea is to maintain this independence throughout the whole implementation.

Linear operations can be performed straightforwardly as the independent domains are not combined during their execution. For nonlinear operations, the idea is to write the target function as a series of 2-input AND (multiplication) and XOR operations. As stated, XORs are easy to mask, while the authors of [GMK16] have provided a methodology to achieve a masked variant of 2-input multiplier at any arbitrary order  $d$ . The DOM-indep multiplier, where it is supposed that the sharing of the given inputs are completely independent of each other, consists of three steps: calculation, resharing, and compression. Focusing on first-order security  $d = 1$ , as the calculation step, the AND operation over two shared inputs  $a = a_0 \oplus a_1$  and  $b = b_0 \oplus b_1$  generates four terms  $a_0b_0$ ,  $a_0b_1$ ,  $a_1b_0$  and  $a_1b_1$ . Naturally, the terms are of two kinds: either they preserve the independence of domains ( $a_0b_0$  and  $a_1b_1$ ) or violate it ( $a_0b_1$  and  $a_1b_0$ ). Note that domains are expressed in black, and blue colors in Figure 1. In order to be able to XOR some of such terms and generate

output shares  $x = x_0 \oplus x_1 = ab$ , the cross-domain terms (expressed in red color in Figure 1) should be re-masked using fresh randomness, i.e., resharing step. As shown in Figure 1, each term (cross-domain ones after being refreshed) are stored in registers to avoid the propagation of the glitches or let say to maintain non-completeness. Registers which store the inner-domain terms are represented with the dotted line in the figure. The reason is that they are optional and can be ignored. However, they are usually used to create a pipeline stage. The compression layer XORs each two register outputs and generates the output shares  $x_0$  and  $x_1$ . Generally speaking, for a  $d^{\text{th}}$ -order DOM multiplier,  $d(d+1)/2$  fresh masks are required.

## 2.3 Probing Security

The probing security is a model used to evaluate the security of masking schemes, firstly introduced by Ishai et al. [ISW03]. According to this model, a  $d^{\text{th}}$ -order attack corresponds to  $d$  independent probes that an attacker puts on the target circuit. A design is called  $d^{\text{th}}$ -order secure if any combination of up to  $d$  of the intermediate values observed by those probes does not result in reproducing sensitive information. Note that  $d$ -probing is a stronger attacker model compared to the  $d^{\text{th}}$ -order DPA attack as the attacker observes noisy signals in DPA. Therefore, although looking simple, the model allows the highest access level to the intermediate values. Hence, if a design is  $d$ -probing secure, it is claimed to be provably secure against any DPA attack up to the  $d^{\text{th}}$  order under certain leakage assumptions.

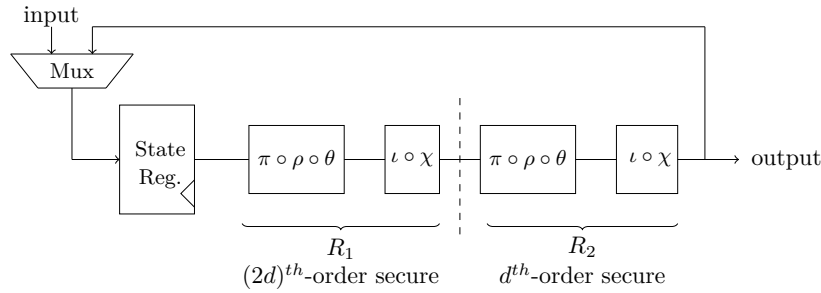
The probing model works properly in the software implementations where the instructions are performed sequentially. Omitting the leakage may originate from micro-architecture, each instruction can be seen as an atomic gate, i.e., the output is ready after a certain time regardless of the delay of its inputs. However, this high-level of abstraction excludes the physical defaults like glitches inherently happening in CMOS hardware implementations. To the best of our knowledge, *robust* probing model [FGP<sup>+</sup>18a] is the best-revised variant of the  $d$ -probing model, considering the physical defaults of hardware implementations. The mode is extended to cover glitches and assumes that any probe placed on any point of a circuit reveals information about not only that probed wire but also all the intermediate values in the path back to the synchronization point, i.e., registers. Since our main focus in this paper is hardware implementation of masking schemes, we consider such a glitch-extended probing model in our designs and evaluations. To this end, we further use the recently-introduced open-source verification tool SILVER [KSM20] under robust probing model to assess the security of our constructions.

## 3 Generic Low-Latency Keccak Design

In this section, we first briefly review the low-latency TI-Keccak presented in [ABP<sup>+</sup>18]. Then, we discuss the difficulties encountered in its generalization from first order to higher orders. We focus on the costs and benefits of the DOM scheme with an emphasis on using it as an underlying secure structure for Keccak. Subsequently, we introduce our idea to realize a low-latency implementation of the original DOM-Keccak. The security order of our design can arbitrarily be adjusted while attaining a lower area compared to the [ABP<sup>+</sup>18]. We investigate the associated challenges and present our corresponding solutions.

### 3.1 Rhythmic Keccak: Two-Round Unrolled TI-Keccak

A recent study by Arribas et al. [ABP<sup>+</sup>18] provides a method to reduce the number of clock cycles in the masked realization of Keccak. Their method unrolls two rounds without



**Figure 2:** Two-rounds unrolled architecture of TI-Keccak.

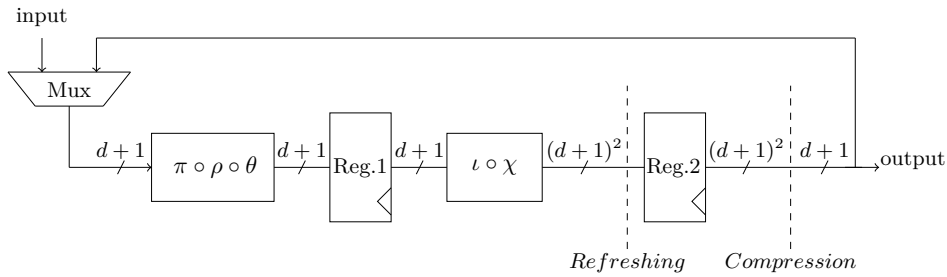
placing any registers in between to accelerate the execution of the function. Figure 2 illustrates the outline of their suggested structure. They construct a two-round unrolled implementation of a first-order secure Keccak by eliminating the state register of the second round. The implementation achieves a halved number of clock cycles which results in reduced latency.

Register omission can pose potential challenges in hardware masking schemes. These challenges seem to be more severe in the DOM scheme rather than TI due to using the minimum number of input shares. DOM's security is based on the fact that the domains should be kept independent and registers are used to avoid propagation of glitches. Hence, for the sake of simplicity, Arribas et al. preferred utilizing the TI scheme with 5 and 6 input shares to realize a first-order implementation. They pointed out that the composition of nonlinear masked functions without any intermediate register can cause share combination, i.e., violating non-completeness, and the design may exhibit leakage. To cope with this issue, the authors maintained the non-completeness with the provision that the security order of the first non-linear masked function should be at least two times that of the successive function when no register is placed in between. We should highlight that this is only a necessary condition and still the extended-glitch probing security should carefully be examined. For example, if a two-round unrolled implementation of Keccak is expressed as  $F = R_2 \circ R_1$ , the first-order non-completeness property of  $F$  states that any combination of two input shares of  $R_2$  is required to be independent of the secrets. In other words, any combination of two output shares of  $R_1$  must be independent of the input. Then, the security of the implementation should be checked by following the propagation of output bits to the input bits backwards under the glitch-extended probing model. The authors presented two first-order secure designs by applying the aforementioned idea; one with 5 input shares ( $5 \rightarrow 10 \rightarrow 5$ ) and the other one with 6 input shares ( $6 \rightarrow 6 \rightarrow 6$ ). The results demonstrated that the latter yields a more compact implementation compared to the former.

**Extension to Higher Orders.** A similar technique as described above can be used to provide  $d^{th}$ -order security for the implementation of an  $n$ -round unrolled Keccak. Simply put, each round  $R_i$  should be implemented with the security order of  $(d \times 2^{n-i})$  where  $1 \leq i \leq n$ . For instance, the required security order for  $R_1$  and  $R_2$  is equal to 4 and 2 to achieve a second-order secure two-rounds unrolled design. Likewise, the required security order for  $R_1$  and  $R_2$  is equal to 6 and 3 to achieve a third-order secure one.

Using this approach, there exist two serious limitations while implementing these higher-order secure multi-round unrolled designs of Keccak. The first challenge arises from the difficulty of finding non-complete and efficient sharings for the 5-bit  $\chi$  function. According to Equation (2), one can determine the minimum number of input shares required to fulfill the  $d^{th}$ -order non-completeness in TI. Given that  $\chi$  is a quadratic function, i.e., with the algebraic degree of 2, the minimum number of input shares for a  $4^{th}$ -order and





**Figure 3:** Round function of the original DOM-Keccak with two register stages.

$6^{th}$ -order secure  $\chi$  function is  $2 \times 4 + 1 = 9$  and  $2 \times 6 + 1 = 13$  shares, respectively. In addition, every of such masked implementations of  $\chi$  should satisfy the uniform sharing at their output. Finding such implementations with uniform output sharing is a non-trivial and intricate task for low number of shares [BNN<sup>+</sup>12, Bil15]. Although it seems that having a higher number of shares than  $td + 1$  to achieve  $d^{th}$ -order security would ease to satisfy the uniformity of output sharing, no uniform masked implementation of  $\chi$  with the aforementioned numbers of shares has been reported yet. Alternatively, fresh randomness should be added, i.e., mask refreshing, to maintain the uniformity of their output sharing.

The second important drawback is that even supposing that non-complete and uniform masked realization of  $\chi$  with 9 and 13 shares are found, the implementation costs would be considerably high. Note that the area overhead and the latency of a masked implementation grows approximately quadratically with respect to the number of shares [FGP<sup>+</sup>18b].

### 3.2 Original DOM-Keccak

Gross et al. [GSM17a] presented the first- and second-order secure implementation of Keccak using DOM. Their design contained two register stages, leading to the latency of two clock cycles per Keccak round. The first register stage, so-called state register, was placed before the linear operations, and the other one after the refreshing layer of the DOM multiplier used in  $\chi$  function. Later, it has been shown in [ABP<sup>+</sup>18] that these designs do not maintain non-completeness. One fix is to relocate the state register, as shown in Figure 3, which maintains the same latency. Hence, the original authors have updated their design in [GSM17b] and [GitHub](#) accordingly.

Although this design – requiring two clock cycles per round – has a higher latency compared to the equivalent TI designs, it has a notably smaller area, which makes it more appropriate particularly at higher orders. This fact motivates us to develop a tweaked design to decrease the latency. Indeed, we intend to take the advantage of the small number of shares in DOM and overcome its high latency. To this end, our goal is to use a single register stage in each round, achieving a lower latency compared to the original DOM implementation and lower area compared to the Rhythmic TI-Keccak while being adjustable to any arbitrary protection order.

### 3.3 Our Single Register per Round Realization

Lowering the latency of the original DOM-Keccak to one clock cycle per round implies removing one of the two registers of the architecture depicted in Figure 3. To this end, we relocate the *compression* layer in such a way that the round function needs only one register layer. Namely, the *compression* layer is performed after the linear operation  $\theta$ , and hence, it should be instantiated  $(d + 1)^2$  times instead of  $d + 1$ .

### 3.3.1 Composability Issue

The first linear operation following the  $\chi$  function is the  $\theta$  operation in the next round. The specification of  $\theta$  reveals that composing  $\chi$  and  $\theta$  functions without any register in the middle potentially leaks information.

As expressed in Equation (1),  $\theta$  is defined such that it adds the summation of ten bits as a parity to each processed bit. One of these ten bits is exactly adjacent to the processed bit. Thus, the  $\theta$  function includes the XOR of every two adjacent output bits of each  $\chi$  instance. In the following, we demonstrate a potential security failure that can occur if  $\chi$  and  $\theta$  are composed without any register and any compression layer. Let us denote five inputs of a  $\chi$  instance by  $\langle a, b, c, d, e \rangle$  and its outputs by  $\langle a', b', c', d', e' \rangle$  as shown in Equation (3).

$$\begin{aligned} a' &= a \oplus \bar{b}c \\ b' &= b \oplus \bar{c}d \\ c' &= c \oplus \bar{d}e \\ d' &= d \oplus \bar{e}a \\ e' &= e \oplus \bar{a}b \end{aligned} \quad (3)$$

In order to show the security issue, we exemplarily focus on the first- and second-order cases. Note that, in this setting, the masked form of  $\chi$  receives  $d + 1$  input shares and provides  $(d + 1)^2$  output shares. Following the DOM principle, the component functions and output shares of the masked  $\chi$  are as defined as given below. The dashed line categorizes the output shares, which are combined in the subsequent compression layer, after being refreshed (fresh randomness) and stored in a register layer.

First-order output shares					
output share	$a'$	$b'$	$c'$	$d'$	$e'$
0	$\bar{b}_0 c_0 \oplus a_0$	$\bar{c}_0 d_0 \oplus b_0$	$\bar{d}_0 e_0 \oplus c_0$	$\bar{e}_0 a_0 \oplus d_0$	$\bar{a}_0 b_0 \oplus e_0$
1	$b_0 c_1$	$c_0 d_1$	$d_0 e_1$	$e_0 a_1$	$a_0 b_1$
-----					
2	$\bar{b}_1 c_0$	$c_1 d_0$	$\bar{d}_1 e_0$	$e_1 a_0$	$a_1 b_0$
3	$\bar{b}_1 c_1 \oplus a_1$	$\bar{c}_1 d_1 \oplus b_1$	$\bar{d}_1 e_1 \oplus c_1$	$\bar{e}_1 a_1 \oplus d_1$	$\bar{a}_1 b_1 \oplus e_1$
Second-order output shares					
output share	$a'$	$b'$	$c'$	$d'$	$e'$
0	$\bar{b}_0 c_0 \oplus a_0$	$\bar{c}_0 d_0 \oplus b_0$	$\bar{d}_0 e_0 \oplus c_0$	$\bar{e}_0 a_0 \oplus d_0$	$\bar{a}_0 b_0 \oplus e_0$
1	$b_0 c_1$	$c_0 d_1$	$d_0 e_1$	$e_0 a_1$	$a_0 b_1$
2	$b_0 c_2$	$c_0 d_2$	$d_0 e_2$	$e_0 a_2$	$a_0 b_2$
-----					
3	$\bar{b}_1 c_0$	$c_1 d_0$	$\bar{d}_1 e_0$	$e_1 a_0$	$a_1 b_0$
4	$\bar{b}_1 c_1 \oplus a_1$	$\bar{c}_1 d_1 \oplus b_1$	$\bar{d}_1 e_1 \oplus c_1$	$\bar{e}_1 a_1 \oplus d_1$	$\bar{a}_1 b_1 \oplus e_1$
5	$b_1 c_2$	$c_1 d_2$	$d_1 e_2$	$e_1 a_2$	$a_1 b_2$
-----					
6	$b_2 c_0$	$c_2 d_0$	$d_2 e_0$	$e_2 a_0$	$a_2 b_0$
7	$b_2 c_1$	$c_2 d_1$	$d_2 e_1$	$e_2 a_1$	$a_2 b_1$
8	$\bar{b}_2 c_2 \oplus a_2$	$\bar{c}_2 d_2 \oplus b_2$	$\bar{d}_2 e_2 \oplus c_2$	$\bar{e}_2 a_2 \oplus d_2$	$\bar{a}_2 b_2 \oplus e_2$

Let us focus on output share 1 in the first-order case. When it enters to the  $\theta$  operation, as stated above the adjacent output bits are XORed. The below-given expressions show that both shares of some input variables appear in the corresponding XOR. In other words, placing a glitch-extended probe exemplary on  $a'_1 \oplus b'_1$  would propagate to both  $c_0$  and  $c_1$ , hence violating the non-completeness and potential information leakage. This is due to

the absence of any register layer between  $\chi$  and  $\theta$ .

$$\begin{aligned} a'_1 \oplus b'_1 &= b_0 c_1 \oplus c_0 d_1 \\ b'_1 \oplus c'_1 &= c_0 d_1 \oplus d_0 e_1 \\ c'_1 \oplus d'_1 &= d_0 e_1 \oplus e_0 a_1 \\ d'_1 \oplus e'_1 &= e_0 a_1 \oplus a_0 b_1 \\ e'_1 \oplus a'_1 &= a_0 b_1 \oplus b_0 c_1 \end{aligned}$$

The same can be seen for the second order. For example, an instance of  $\theta$  placed on the output share 1 would reveal both  $c_0$  and  $c_1$ . Hence, placing a second probe on  $c_2$  would violate the desired second-order security.

### 3.3.2 Solving the Non-completeness Issue

Our solution is a realignment of component functions such that all component functions of an instance of  $\chi$  which belong to (or let say generate) an output share, receive only a single share of every input. Therefore, placing any linear function (including  $\theta$ ) on output shares of  $\chi$  would not combine more than one input share. Hence, the non-completeness would be maintained. In the following, we explain the underlying idea in more detail.

Below, our solutions for the first and second order are given. Note that, compared to what has been given in Section 3.3.1, we just reordered the component function for each output bit. The functionality is exactly the same, but it can be seen that in each output share (i.e., in each row of the below-given tables), at most one share of each input variable shows up. For example, in contrast to what has been shown before, output share 1 contains only  $\{a_1, b_0, c_1, d_0, e_1\}$ . Therefore, any linear function which follows these masked realizations of  $\chi$  would not violate the (first- and higher-order) non-completeness. Hence, we can easily compose masked  $\chi$  and  $\theta$  without any security issue.

First-order solution					
output share	$a'$	$b'$	$c'$	$d'$	$e'$
0	$\overline{b_0}c_0 \oplus a_0$	$\overline{c_0}d_0 \oplus b_0$	$\overline{d_0}e_0 \oplus c_0$	$\overline{e_0}a_0 \oplus d_0$	$\overline{a_0}b_0 \oplus e_0$
1	$\overline{b_0}c_1$	$c_1d_0$	$\overline{d_0}e_1 \oplus c_1$	$e_1a_1$	$a_1b_0 \oplus e_1$
2	$b_1c_0 \oplus a_1$	$\overline{c_0}d_1 \oplus b_1$	$d_1e_0$	$\overline{e_0}a_1 \oplus d_1$	$a_1b_1$
3	$b_1c_1$	$c_1d_1$	$d_1e_1$	$e_1a_0$	$\overline{a_0}b_1$

Second-order solution					
output share	$a'$	$b'$	$c'$	$d'$	$e'$
0	$\overline{b_0}c_0 \oplus a_0$	$\overline{c_0}d_0 \oplus b_0$	$\overline{d_0}e_0 \oplus c_0$	$\overline{e_0}a_0 \oplus d_0$	$\overline{a_0}b_0 \oplus e_0$
1	$\overline{b_0}c_1$	$c_1d_0$	$\overline{d_0}e_1 \oplus c_1$	$e_1a_1$	$a_1b_0 \oplus e_1$
2	$\overline{b_0}c_2$	$c_2d_0$	$\overline{d_0}e_2 \oplus c_2$	$e_2a_2$	$a_2b_0 \oplus e_2$
3	$b_1c_0 \oplus a_1$	$\overline{c_0}d_1 \oplus b_1$	$d_1e_0$	$\overline{e_0}a_1 \oplus d_1$	$a_1b_1$
4	$b_1c_1$	$c_1d_1$	$d_1e_1$	$e_1a_2$	$a_2b_1$
5	$b_1c_2$	$c_2d_1$	$d_1e_2$	$e_2a_0$	$\overline{a_0}b_1$
6	$b_2c_0 \oplus a_2$	$\overline{c_0}d_2 \oplus b_2$	$d_2e_0$	$\overline{e_0}a_2 \oplus d_2$	$a_2b_2$
7	$b_2c_1$	$c_1d_2$	$d_2e_1$	$e_1a_0$	$\overline{a_0}b_2$
8	$b_2c_2$	$c_2d_2$	$d_2e_2$	$e_2a_1$	$a_1b_2$

**Extension to Higher Orders.** Without losing generality, we can follow a rule for implementing the  $\chi$  function such that it meets our security requirement. We refer to our solution for the first order as an example. Let us represent the share index of input variables given to the component functions by a table so-called *index configuration* as

shown below<sup>1</sup>. It indeed reflects the share index of each input variable that is involved in the calculation of each output share.

**Table 1:** Index configuration of our first-order solution

output share	$a$	$b$	$c$	$d$	$e$
0	0	0	0	0	0
1	1	0	1	0	1
2	1	1	0	1	0
3	0	1	1	1	1

For example, the first row indicates that  $\{a_0, b_0, c_0, d_0, e_0\}$  are only involved in the generation of the first output share. Similarly, the component functions of the second output share receive  $\{a_1, b_0, c_1, d_0, e_1\}$  as their inputs. Note that based on this configuration, the place of all quadratic monomials of the  $\chi$  function becomes clear. However, there is freedom for the linear monomials. For example, in our first-order solution, based on the Table 1, linear monomial  $a_0$  (in output  $a'$ ) can be generated by the first or the last component function.

This strategy can be easily extended to any arbitrary order. Since the index configuration table inherently implies that only one share from each input is given to the component functions of an output share, the only fact which we need to consider is to make sure that the correctness of the masking is maintained. More precisely, looking at Equation (3), the  $\chi$  function involves all quadratic monomials between every two adjacent variables. Therefore, the index configuration should assure that every (rotate-wise) two adjacent columns cover all possible combinations between sharing indexes. For example, every two adjacent columns of the above-given index configuration in Table 1 cover all  $(0, 0)$ ,  $(0, 1)$ ,  $(1, 0)$  and  $(1, 1)$  cases. This would allow, for example, the generation of all quadratic monomials  $a_i b_j$  for  $ab$  as a term in  $e'$  (see Equation (3)).

Our observation is that the configuration of four adjacent columns can be easily fixed by alternating between two cases. This can be seen in Table 1, where columns associated to  $b$  and  $d$  are equal, and the same holds for  $c$  and  $e$ . However, that of  $a$  is different from all others, as it should provide all possible combinations with  $b$  as well as with  $e$ . Following this principle, we introduce our solution for any arbitrary order. In the index configuration given in Table 2, we denote the number of shares by  $s$  required for security at order  $s - 1$ .<sup>2</sup> As stated, the configuration of columns  $b$  and  $d$  are straightforward. The same holds for that of  $c$  and  $e$ . This means that all possible combinations  $(i, j)$  for  $0 \leq i, j < s$  between columns  $(b, c)$ ,  $(c, d)$  and  $(d, e)$  are covered.

For the column  $a$ , we actually take the first  $s$  elements as in the column  $c$ . For every next  $s$  elements in  $a$ , we just rotate them one element upwards. A pseudo-code is given in Algorithm 1 which illustrates this process. This guarantees all possible combinations  $(i, j)$  between columns  $(a, b)$  and between columns  $(e, a)$ , hence achieving our requirements. As stated before, no special condition should be considered for the linear monomials. The index configuration inherently allows multiple choices for the place of the linear monomials. For example,  $b_0$  can be covered by the component functions of any of the first  $s$  output shares.

We would like to highlight a few side points about our solution:

- The expressed index configuration is not unique. There may be other solutions to distribute input shares to the  $\chi$  component functions while achieving the same goal.

<sup>1</sup>A similar table representation has been used in [BKN19].

<sup>2</sup>As  $d$  is used as an input to the  $\chi$  function, we represent the security order here by  $s - 1$  which is inline with the notations given in Section 2.2.1.

**Table 2:** Index configuration of our  $d^{\text{th}}$ -order solution

	output share	$a$	$b$	$c$	$d$	$e$
$i = 0$	$j = 0$	0	0	0	0	0
	$j = 1$	1	1	0	1	1
	$j = 2$	2	2	0	2	2
	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
	$j = s - 1$	$s - 1$	$s - 1$	0	$s - 1$	$s - 1$
$i = 1$	$j = 0$	$s$	1	1	0	1
	$j = 1$	$s + 1$	2	1	1	1
	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
	$j = s - 2$	$2s - 2$	$s - 1$	1	$s - 2$	1
	$j = s - 1$	$2s - 1$	0	1	$s - 1$	1
$i = 2$	$j = 0$	$2s$	2	2	0	2
	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
	$j = s - 3$	$3s - 3$	$s - 1$	2	$s - 3$	2
	$j = s - 2$	$3s - 2$	0	2	$s - 2$	2
	$j = s - 1$	$3s - 1$	1	2	$s - 1$	2
...	...	...	...	...	...	...
$i = s - 1$	$j = 0$	$s(s - 1)$	$s - 1$	$s - 1$	0	$s - 1$
	$j = 1$	$s(s - 1) + 1$	0	$s - 1$	1	$s - 1$
	$j = 2$	$s(s - 1) + 2$	1	$s - 1$	2	$s - 1$
	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
	$j = s - 1$	$s^2 - 1$	$s - 2$	$s - 1$	$s - 1$	$s - 1$

- Although developed for  $\theta$ , if the masked  $\chi$  function is realized based on our solution, any linear function can be placed right after the  $\chi$  without having any impact on the security level. This can be justified with respect to being secure in the glitch-extended probing model. The ordering, in which the component functions are sorted in our solution, guarantees the appearance of at most one input share in all component functions of each output share. Hence, neither  $\theta$  nor any other linear operation would have any effect on the non-completeness.
- This is a solution dedicated to the  $\chi$  function, and it cannot be straightforwardly extended to any (even quadratic) function.

---

**Algorithm 1** Generating index configurations for output bits in  $\chi$  function
 

---

**Require:** security order  $(s - 1)$ .

**Ensure:** index configurations  $\alpha_{i,j}, \beta_{i,j}, \gamma_{i,j}, \delta_{i,j}, \epsilon_{i,j}$  satisfying  $a_{\alpha_{i,j}}, b_{\beta_{i,j}}, c_{\gamma_{i,j}}, d_{\delta_{i,j}}, e_{\epsilon_{i,j}}$  in the row identified by  $i$  and  $j$  in Table 2

```

1:
2: for  $i = 0$  to  $(s - 1)$  do
3:   for  $j = 0$  to  $(s - 1)$  do
4:      $\alpha_{i,j} \leftarrow (i + j) \bmod s$ 
5:      $\beta_{i,j} = \delta_{i,j} \leftarrow i$ 
6:      $\gamma_{i,j} = \epsilon_{i,j} \leftarrow j$ 
7:   end for
8: end for

```

---

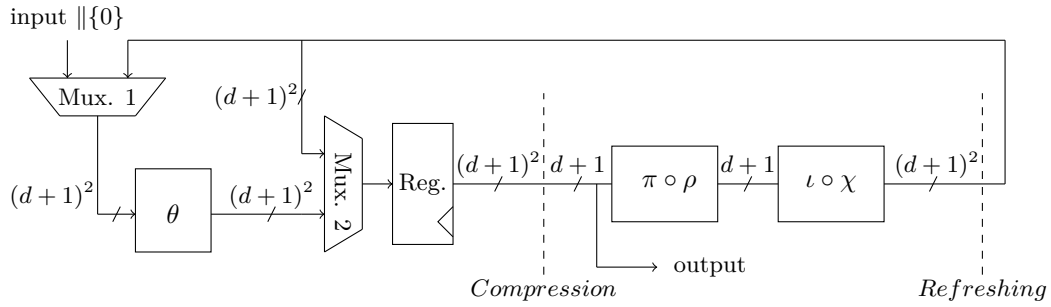


Figure 4: Our generic low-latency DOM-Keccak.

### 3.3.3 Design Architecture

Figure 4 presents our general design architecture for the low-latency DOM-Keccak, which has only one register stage per round. Compared to the original DOM-Keccak, the compression layer has been moved after the application of  $\theta$ . To this end, the  $\theta$  function should be instantiated  $(d+1)^2$  times, since  $\chi$  turns  $d+1$  input shares to  $(d+1)^2$ . However, since the primary input of the Keccak- $f$  is first given to the diffusion layer (see Section 2.1), we need to pad the sharing of the primary input with 0 to make  $(d+1)^2$  shares. Note that, this should be done with special care considering the subsequent compression layer to avoid multiple primary input shares to be combined (XORed). We provide more information about the compression layer in Section 3.4, but as a matter of fact, we need to make sure that every  $i$ -th subsequent  $d+1$  shares of the result of padding contains only the  $i$ -th share of the primary input.

Further, at the last round, the primary output is provided by the  $\chi$  function followed by the round constant addition, i.e.,  $\iota$ , which in our design has  $(d+1)^2$  shares. Therefore, we instantiated another multiplexer, identified as Mux. 2, which bypasses the  $\theta$  function. This allows us to take the primary output directly from the compression layer which turns the number of shares back to  $d+1$ . Note that the added multiplexer is a relatively large module,  $b(d+1)^2$  instances for Keccak- $f[b]$  with  $b$ -bit state. As an alternative solution, a dedicated  $b(d+1)^2$ -bit register and a compression layer can be instantiated for the primary output, which is expected to have a higher area overhead.

Compared to the original DOM-Keccak, our design instantiates  $b(d+1)$  less registers, but in return uses  $b(d(d+1) + (d+1)^2)$  more multiplexers (for Mux. 1 and Mux. 2, respectively<sup>3</sup>) and more importantly needs to use  $(d+1)^2$  instances of the  $\theta$  function compared to  $d+1$  instances in original DOM-Keccak (see Figure 3). Note that  $\theta$  is a relatively large module (XORs ten bits, for each bit of the  $p$ -bit state), and the number of extra  $\theta$  instances and multiplexers, which our design needs, increase exponentially with  $d$ . Therefore, the area overhead of our low-latency DOM-Keccak may be comparable to that of the original DOM-Keccak for low security order. However, we expect this to be not the case at higher orders. We present detailed performance results in Section 4.

## 3.4 Refreshing and Compression Layers

We close this section with a brief discussion on the refreshing and compression layers we have used in our design. In short, with some small differences, they follow the same concept as in DOM [GMK16]. Inner-domain component functions, i.e., those which include quadratic monomials with the same sharing index, are not blinded with a fresh mask. In

<sup>3</sup>Any multiplexer of MUX. 1, where of its inputs is tied to '0', is simplified to an AND gate, i.e., smaller area compared to a multiplexer.

contrast, analogous cross-domain component functions, e.g.,  $a_i b_j$  and  $a_j b_i$  for  $0 \leq i \neq j < s$ , are blinded by the same fresh mask, i.e.,  $a_i b_j \oplus r^{i,j}$  and  $a_j b_i \oplus r^{j,i}$  while  $r^{i,j} = r^{j,i}$ . As an example, the below-given table illustrates this procedure for the first order. Note that since a single fresh mask bit is required for each output share, we present  $r_k^{0,1} = r_k^{1,0}$  by  $r_{k \in \{0, \dots, 4\}}$ . We further provided a general algorithm for any arbitrary order for the entire component functions of the  $\chi$  function including the place of each monomial as well as fresh masks in [Algorithm 2](#).

First-order solution after refreshing					
output share	$a'$	$b'$	$c'$	$d'$	$e'$
0	$\overline{b_0}c_0 \oplus a_0$	$\overline{c_0}d_0 \oplus b_0$	$\overline{d_0}e_0 \oplus c_0$	$\overline{e_0}a_0 \oplus d_0$	$\overline{a_0}b_0 \oplus e_0$
1	$\overline{b_0}c_1 \oplus r_0$	$c_1 d_0 \oplus r_1$	$\overline{d_0}e_1 \oplus c_1 \oplus r_2$	$e_1 a_1$	$a_1 b_0 \oplus e_1 \oplus r_4$
2	$b_1 c_0 \oplus a_1 \oplus r_0$	$\overline{c_0}d_1 \oplus b_1 \oplus r_1$	$d_1 e_0 \oplus r_2$	$\overline{e_0}a_1 \oplus d_1 \oplus r_3$	$a_1 b_1$
3	$b_1 c_1$	$c_1 d_1$	$d_1 e_1$	$e_1 a_0 \oplus r_3$	$\overline{a_0}b_1 \oplus r_4$

A careless combination layer would split the output shares into two groups and combine output shares (0, 1) and (2, 3). While this does not pose an issue for  $a'$ ,  $b'$ ,  $c'$ , and  $e'$ , it is not a valid compression for  $d'$ . Therefore, right before the compression, we need to rearrange the output shares of  $d'$  in order to keep its compression valid. The rearrangement can be done for example following the DOM concept. This is shown below. Note that such a rearrangement is done after the application of  $\theta$ , but it can be done either before the register or afterwards (see [Figure 4](#)).

First-order solution before compression					
output share	$a'$	$b'$	$c'$	$d''$	$e'$
0	$\overline{b_0}c_0 \oplus a_0$	$\overline{c_0}d_0 \oplus b_0$	$\overline{d_0}e_0 \oplus c_0$	$\overline{e_0}a_0 \oplus d_0$	$\overline{a_0}b_0 \oplus e_0$
1	$\overline{b_0}c_1 \oplus r_0$	$c_1 d_0 \oplus r_1$	$\overline{d_0}e_1 \oplus c_1 \oplus r_2$	$\overline{e_0}a_1 \oplus d_1 \oplus r_3$	$a_1 b_0 \oplus e_1 \oplus r_4$
2	$b_1 c_0 \oplus a_1 \oplus r_0$	$\overline{c_0}d_1 \oplus b_1 \oplus r_1$	$d_1 e_0 \oplus r_2$	$e_1 a_1$	$a_1 b_1$
3	$b_1 c_1$	$c_1 d_1$	$d_1 e_1$	$e_1 a_0 \oplus r_3$	$\overline{a_0}b_1 \oplus r_4$

The same holds for higher orders. In other words, the compression of the shares of all state bits except those corresponding to the output bit  $d'$  in every  $\chi$  instance, are straightforwardly done by XORing every  $s = d + 1$  consecutive shares. For  $d'$ , we need to apply a rearrangement before the same compression. In short, if we denote the rearranged output shares by  $d''$ , we can write  $d''_k = d'_{(k \bmod s) \times s + \lfloor k/s \rfloor}$  for  $0 \leq k < s^2 - 1$ .

We would like to highlight that the above-given construction for the shared  $\chi$  function in addition to the  $\theta$  operation being applied on each share before the compression layer allows us to remove the register stage at the beginning of the  $\chi$  function. In other words, the diffusion property of  $\theta$  plays an important role. Otherwise, the composition of two pure shared  $\chi$  functions would necessitate the instantiation of a register stage at the beginning of each  $\chi$  function.

## 4 Evaluations

In this section, we evaluate our low-latency DOM-Keccak in two aspects: first, we present the results of ASIC implementation of the first- to fifth-order secure designs to analyze the performance. To serve the purpose, we compare our results with the related works. Next, we evaluate the security of our design with experimental analyses, including the SILVER verification tool and the t-test. Similar to the state of the art, we focus on the implementation of one of the small Keccak- $f$  permutations with the state size  $b = 200$ , and the round number  $n_r = 18$  accordingly.

---

**Algorithm 2** Generating composable output shares in  $\chi$  function for the low-latency DOM-Keccak
 

---

**Require:** security order  $(s - 1)$ , input shares  $a, b, c, d, e$ , each  $s$  shares

**Ensure:** output shares  $a', b', c', d', e'$ , each  $s^2$  shares

```

1: for  $i = 0$  to  $(s - 1)$  do
2:   for  $j = i + 1$  to  $(s - 1)$  do
3:      $r_0^{i,j} = r_0^{j,i} \xleftarrow{\$} \mathbb{F}_2$ 
4:      $r_1^{i,j} = r_1^{j,i} \xleftarrow{\$} \mathbb{F}_2$ 
5:      $r_2^{i,j} = r_2^{j,i} \xleftarrow{\$} \mathbb{F}_2$ 
6:      $r_3^{i,j} = r_3^{j,i} \xleftarrow{\$} \mathbb{F}_2$ 
7:      $r_4^{i,j} = r_4^{j,i} \xleftarrow{\$} \mathbb{F}_2$ 
8:   end for
9: end for
10: for  $i = 0$  to  $(s - 1)$  do
11:   for  $j = 0$  to  $(s - 1)$  do
12:      $k \leftarrow i \times s + j$ 
13:     if  $i = 0, j = 0$  then
14:        $a'_k \leftarrow \bar{b}_i c_j \oplus a_{[(i+j) \bmod s]}$ 
15:        $b'_k \leftarrow \bar{c}_j d_i \oplus b_i$ 
16:        $c'_k \leftarrow \bar{d}_i e_j \oplus c_j$ 
17:        $d'_k \leftarrow \bar{e}_j a_{[(i+j) \bmod s]} \oplus d_i$ 
18:        $e'_k \leftarrow \bar{a}_{[(i+j) \bmod s]} b_i \oplus e_j$ 
19:     end if
20:     if  $i = 0, j \neq 0$  then
21:        $a'_k \leftarrow \bar{b}_i c_j \oplus r_0^{i,j}$ 
22:        $b'_k \leftarrow c_j d_i \oplus r_1^{i,j}$ 
23:        $c'_k \leftarrow \bar{d}_i e_j \oplus c_j \oplus r_2^{i,j}$ 
24:        $d'_k \leftarrow e_j a_{[(i+j) \bmod s]}$ 
25:        $e'_k \leftarrow a_{[(i+j) \bmod s]} b_i \oplus e_j \oplus r_4^{[(i+j) \bmod s], i}$ 
26:     end if
27:     if  $i \neq 0, j = 0$  then
28:        $a'_k \leftarrow b_i c_j \oplus a_{[(i+j) \bmod s]} \oplus r_0^{i,j}$ 
29:        $b'_k \leftarrow \bar{c}_j d_i \oplus b_i \oplus r_1^{i,j}$ 
30:        $c'_k \leftarrow d_i e_j \oplus r_2^{i,j}$ 
31:        $d'_k \leftarrow \bar{e}_j a_{[(i+j) \bmod s]} \oplus d_i \oplus r_3^{[(i+j) \bmod s], j}$ 
32:        $e'_k \leftarrow a_{[(i+j) \bmod s]} b_i$ 
33:     end if
34:     if  $i \neq 0, j \neq 0$  then
35:       if  $i = j$  then
36:          $a'_k \leftarrow b_i c_j$ 
37:          $b'_k \leftarrow c_j d_i$ 
38:          $c'_k \leftarrow d_i e_j$ 
39:       else
40:          $a'_k \leftarrow b_i c_j \oplus r_0^{i,j}$ 
41:          $b'_k \leftarrow c_j d_i \oplus r_1^{i,j}$ 
42:          $c'_k \leftarrow d_i e_j \oplus r_2^{i,j}$ 
43:       end if
44:        $d'_k \leftarrow e_j a_{[(i+j) \bmod s]} \oplus r_3^{[(i+j) \bmod s], j}$ 
45:       if  $(i + j) \bmod s = 0$  then
46:          $e'_k \leftarrow \bar{a}_{[(i+j) \bmod s]} b_i \oplus r_4^{[(i+j) \bmod s], i}$ 
47:       else
48:          $e'_k \leftarrow a_{[(i+j) \bmod s]} b_i \oplus r_4^{[(i+j) \bmod s], i}$ 
49:       end if
50:     end if
51:   end for
52: end for

```

---

▷ filling fresh mask matrices  
 ▷ for only  $i \neq j$  while  $r^{i,j} = r^{j,i}$



**Table 3:** Synthesis result of different Keccak- $f[200]$  designs, using NanGate 45 nm standard cell library.

Design	Security Order	Total Area	CPD*	Freq.	Latency	Delay	Rand.
		(kGE)	(ns)	(MHz)	(cycles)	(ns)	(bits)
<b>Our Constructions</b>							
Low-Latency	1	17.90	1.14	877	18	20.52	200
	2	39.97	1.14	877	18	20.52	600
	3	70.84	1.16	862	18	20.88	1200
	4	111.76	1.17	854	18	21.06	2000
	5	160.19	1.19	840	18	21.42	3000
<b>Related Works [ABP<sup>+</sup>18]</b>							
Original DOM	1	17.50	0.77	1300	36	27.72	200
Original DOM	2	35.22	0.83	1205	36	29.88	600
Low-Latency TI	1	70.12	2.29	436	9	20.61	–

\* Critical Path Delay

## 4.1 Performance Analysis

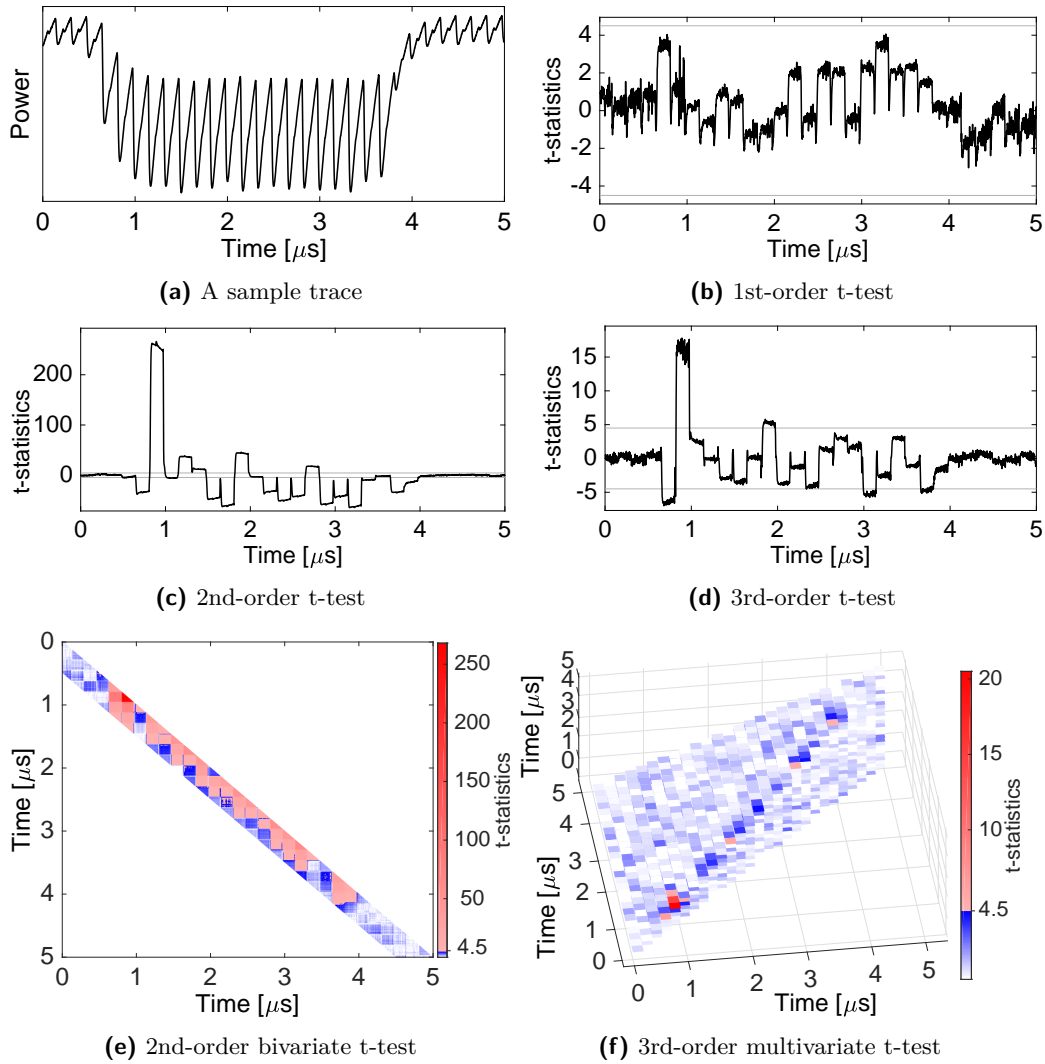
We implemented our design (Figure 4) by an HDL (Verilog) code, and used Synopsys Design Compiler with the NanGate 45 nm Open Cell Library to synthesize it. The `-no_autoungroup` flag was set during the synthesis to keep the hierarchy of the design, and avoid optimizations affecting the security requirements, i.e., maintaining non-completeness. The implementation results are shown in Table 3.

The given results reveal that the first and second-order implementations of our design have about 26%, and 31% less delay compared to the equivalent original DOM implementations. Our design has approximately a constant delay in higher orders. At the same time, the area consumption of the first-order implementation of our design is about 74.5% less than that of the low-latency TI-Keccak, whereas their delay are almost equal. Note that with *delay* in Table 3 we refer to the time required to accomplish the entire Keccak- $f[200]$  operation. Further, the number of random bits given in the table demonstrates the required fresh random bits for each round, which are indeed the same in our design and the original DOM.

## 4.2 Experimental Analyses

As stated before, we have verified the security of our constructed masked  $\chi$  function at different security orders by means of SILVER [KSM20] under glitch-extended probing model. Since the entire design (even only a round function) is out of the feasibility limits of SILVER and any other known relevant verification tools, we have no other choice than examining our full constructions by experimental analyses. To this end, we made use of a Spartan-6 FPGA-based evaluation platform (SAKURA-G [GIS14]), and collected power consumption traces of our designs following the measurement strategy and procedure explained in [SM15] to conduct fixed-versus-random t-test.

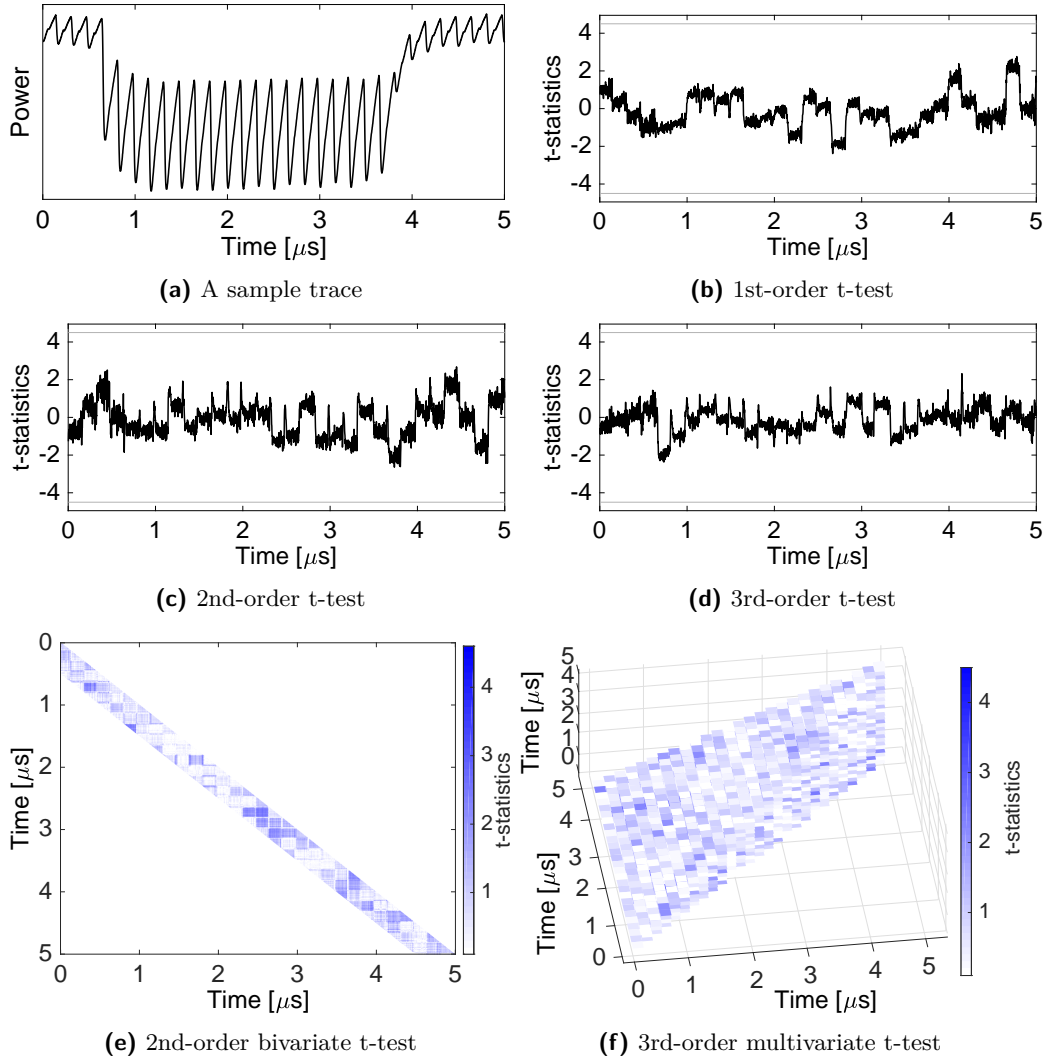
While monitoring the AC-amplified voltage drop over a  $1\ \Omega$  shunt resistor placed on the VDD path of the target FPGA, the underlying design under test was being operated by a stable and jitter-free clock at a frequency of 6 MHz. The power consumption traces have been collected by sampling the aforementioned signal at a sampling frequency of 500 MS/s. As given in Section 3, our constructions – independent of the selected security order – need one clock cycle per Keccak round, i.e., 18 cycles for Keccak- $f[200]$ . Therefore, we made sure to cover all corresponding clock cycles in our measured power traces. An



**Figure 5:** Experimental analysis of our two-share KECCAK- $f[200]$  design using 100 million traces.

example can be seen in Figure 5a. As a side note, our constructions require fresh masks updated at every clock cycle (see Table 3). Therefore, we instantiated Pseudo-Random Number Generators (PRNGs) to supply such fresh randomness. To this end, we made use of the FPGA-optimized construction illustrated in [MMW18], which realizes a 31-bit Linear Feedback Shift Register (LFSR) with the feedback polynomial  $x^{31} + x^{28} + 1$  by means of only three 6-to-1 Look-Up Tables (LUTs). More precisely, for each required fresh mask bit, we instantiated an LFSR seeded randomly at the power up of the device and updated at every clock cycle.

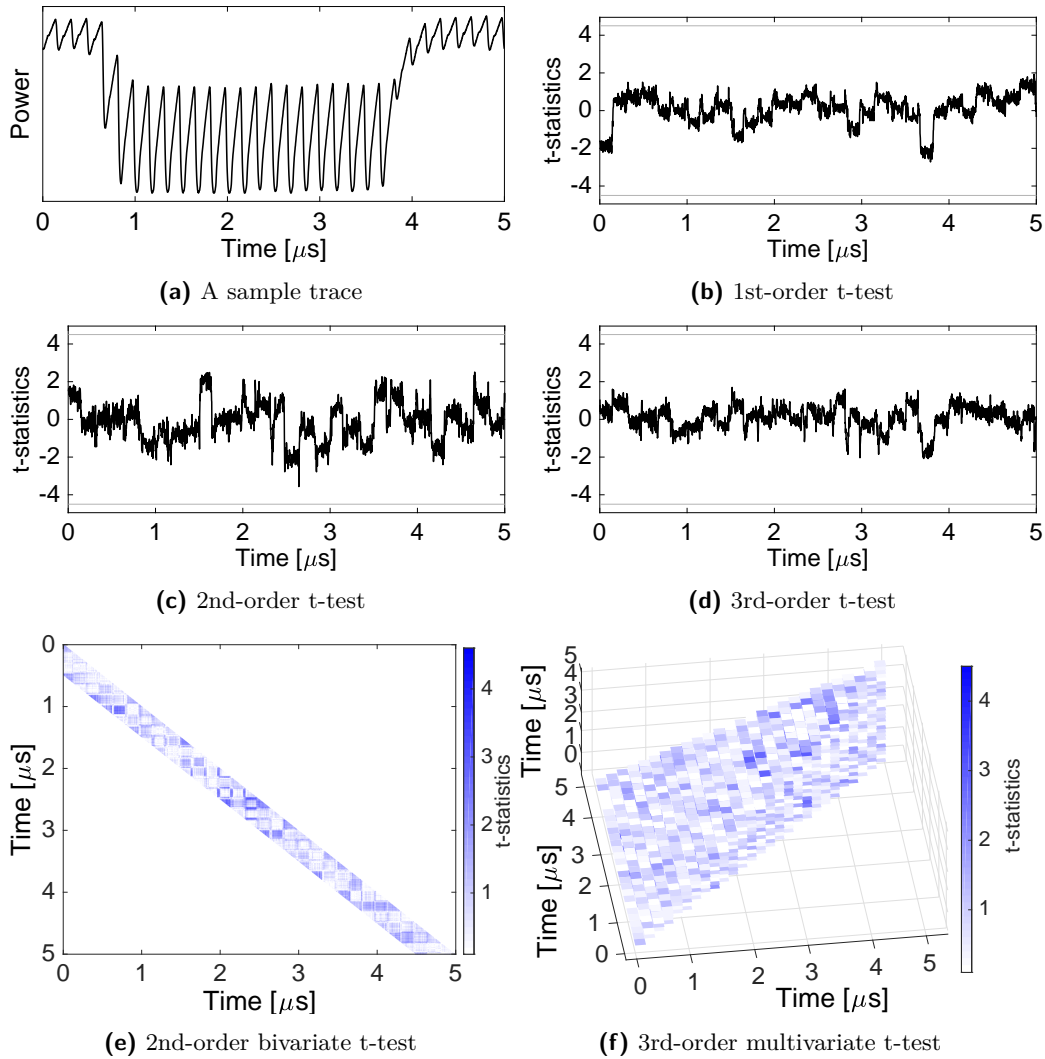
We conducted various forms of fixed-versus-random t-test – also known as TVLA [CDG<sup>+</sup>13] – as the analysis scheme. It is a well-known leakage assessment technique that is able to detect SCA leakages in measurements collected from cryptographic implementations. Starting with our first-order design ( $d=1$ ), we performed the ordinary t-test on each sample point individually (i.e., univariate), whose corresponding result shown in Figure 5b confirms its first-order security. However, this does not hold true when conducting high-order t-tests. For higher orders, we made the traces mean-free (for



**Figure 6:** Experimental analysis of our three-share KECCAK- $f[200]$  design using 100 million traces.

each group of fixed and random individually). Afterwards, each mean-free sample point is squared (resp. cubed) before calculating the t-statistics for univariate second-order (resp. univariate third-order) t-tests (see Figure 5c and Figure 5d).

For the bivariate second-order t-test, we should perform an individual t-test for each combination of every two possible sample points by multiplying the corresponding mean-free power values. In our experiments, each power trace contains 2 500 sample points. This translates to  $2\,500 \times 2\,500 / 2 = 3\,124\,000$  individual t-tests, which is a very time intensive computation even using large machines (e.g., with 24 CPU cores). Since our constructions make use of fresh masks (updated every clock cycle), the bivariate leakage is expected to be present for a combination between sample points which are not far from each other. Therefore, we limited our bivariate analysis to the sample points with at most 3 clock cycles distance. This strongly reduces the amount of computations and allows us to accomplish it in a reasonable time frame. The corresponding results shown in Figure 5e conform the existence of second-order bivariate leakages, as expected.



**Figure 7:** Experimental analysis of our four-share KECCAK- $f[200]$  design using 100 million traces.

For the third-order multivariate analysis, the situation is even worse. If we limit the maximum distance between the sample points to e.g., 3 clock cycles, the number of possible combinations of three sample points is way above the feasibility threshold. It can be seen in Figure 5c to Figure 5e, that the amount of leakage associated to a clock cycle is approximately the same for the entire clock cycle. Therefore, an appropriate sample point per clock cycle should suffice for such analyses. This is actually known as memory effect in power consumption measurements due to the low-pass filter inherently built by the elements involved in the measurement setup, e.g., the shunt resistor, the chip package, and the Printed Circuit Board (PCB) [MM13]. Therefore, we down-sampled the traces by taking a sample point for each clock cycle (carefully selected at the middle of the cycle). Note that such a down sampling and restricting the bivariate analysis to a small period of time has been done in the state of the art as well [CRB<sup>+</sup>16]. The result of this analysis is shown by a 3D pyramid in Figure 5f indicating a few tuples (of three points) whose combination (mean-free product) leads to a detectable leakage. Note that higher-order univariate and multivariate leakages are expected in case of this first-order design. We

showed the detailed results of such analyses as a proof of functionality of our setup.

We have conducted the same analyses on our second- and third-order designs ( $d=2$  and  $d=3$  respectively)<sup>4</sup>. The corresponding results are depicted in Figure 6 and Figure 7 respectively. As an interesting fact, none of such analyses at any order and any variate shows a detectable leakage. This is due to the high noise originating from a high number of fresh masks updated at every clock cycle. More precisely, 600 and 1200 LFSRs are clocked at the same time in second-order and third-order designs, respectively (see Table 3). Such a high amount of noise makes the prediction of higher-order statistical moments with a limited number of measurements challenging, hence hardening the detection of higher-order leakages [PRB09]. Note that we have verified the correctness of our setup and its ability to detect univariate and multivariate higher-order leakages using our first-order design (Figure 5).

## 5 Conclusions

This research made several noteworthy contributions to the state-of-the-art protected implementations of Keccak, which are particularly beneficial for developing feasible low-latency higher-order hardware masking of this cryptographic primitive. First, we identified some challenges when extending the known techniques for protecting the implementations of Keccak to higher orders. Next, we described a methodology to address these challenges. We made use of the specifications of the Keccak structure to adopt the concept of DOM in a low-latency architecture and decrease the number of clock cycles by 50% compared to the state-of-the-art original DOM implementation. Naturally, our design achieves a lower delay and hence a higher throughput while its area footprint is comparable to that of original DOM-Keccak. More importantly, compared to the only-known low-latency design, i.e., Rhythmic-Keccak, our construction needs significantly less area while being easily extendable to higher orders. We have provided a parametric Verilog design for Keccak- $f$ [200], accessible through [https://github.com/Chair-for-Security-Engineering/Low-Latency\\_Keccak](https://github.com/Chair-for-Security-Engineering/Low-Latency_Keccak), where the desired security order can be easily adjusted.

## Acknowledgments

The work described in this paper has been supported in part by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) under Germany's Excellence Strategy - EXC 2092 CASA - 390781972 and through the project 406956718 SuCCESS.

## References

- [ABP<sup>+</sup>18] Victor Arribas, Begül Bilgin, George Petrides, Svetla Nikova, and Vincent Rijmen. Rhythmic Keccak: SCA Security and Low Latency in HW. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2018(1):269–290, 2018.
- [ANR19] Victor Arribas, Svetla Nikova, and Vincent Rijmen. Guards in action: First-order SCA secure implementations of KETJE without additional randomness. *Microprocess. Microsystems*, 71, 2019.
- [BCO04] Eric Brier, Christophe Clavier, and Francis Olivier. Correlation Power Analysis with a Leakage Model. In *CHES 2004*, volume 3156 of *Lecture Notes in Computer Science*, pages 16–29. Springer, 2004.

---

<sup>4</sup>Larger designs – including the LFSRs as the PRNG – could not fit into the target FPGA of our platform.

- [BDN<sup>+</sup>13] Begül Bilgin, Joan Daemen, Ventzislav Nikov, Svetla Nikova, Vincent Rijmen, and Gilles Van Assche. Efficient and First-Order DPA Resistant Implementations of Keccak. In *CARDIS 2013*, volume 8419 of *Lecture Notes in Computer Science*, pages 187–199. Springer, 2013.
- [BDPA13] Guido Bertoni, Joan Daemen, Michaël Peeters, and Gilles Van Assche. Keccak. In *EUROCRYPT 2013*, volume 7881 of *Lecture Notes in Computer Science*, pages 313–314. Springer, 2013.
- [BDPVA10] Guido Bertoni, Joan Daemen, Michaël Peeters, and Gilles Van Assche. Building power analysis resistant implementations of Keccak. In *Second SHA-3 candidate conference*, volume 142. Citeseer, 2010.
- [BGN<sup>+</sup>14] Begül Bilgin, Benedikt Gierlichs, Svetla Nikova, Ventzislav Nikov, and Vincent Rijmen. Higher-Order Threshold Implementations. In *ASIACRYPT 2014*, volume 8874 of *Lecture Notes in Computer Science*, pages 326–343. Springer, 2014.
- [Bil15] Begül Bilgin. *Threshold implementations : as countermeasure against higher-order differential power analysis*. PhD thesis, University of Twente, Enschede, Netherlands, 2015.
- [BKN19] Dusan Bozilov, Miroslav Knezevic, and Ventzislav Nikov. Optimized Threshold Implementations: Minimizing the Latency of Secure Cryptographic Accelerators. In *CARDIS 2019*, volume 11833 of *Lecture Notes in Computer Science*, pages 20–39. Springer, 2019.
- [BNN<sup>+</sup>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*, volume 7428 of *Lecture Notes in Computer Science*, pages 76–91. Springer, 2012.
- [BNN<sup>+</sup>15] Begül Bilgin, Svetla Nikova, Ventzislav Nikov, Vincent Rijmen, Natalia N. Tokareva, and Valeriya Vitkup. Threshold implementations of small S-boxes. *Cryptogr. Commun.*, 7(1):3–33, 2015.
- [BPVA<sup>+</sup>11] Guido Bertoni, Michaël Peeters, Gilles Van Assche, et al. The Keccak Reference. 2011.
- [CDG<sup>+</sup>13] Jeremy Cooper, Elke DeMulder, Gilbert Goodwill, Joshua Jaffe, Gary Kenworthy, Pankaj Rohatgi, et al. Test vector leakage assessment (TVLA) methodology in practice. In *International Cryptographic Module Conference*, volume 20, 2013.
- [CRB<sup>+</sup>16] Thomas De Cnudde, Oscar Reparaz, Begül Bilgin, Svetla Nikova, Ventzislav Nikov, and Vincent Rijmen. Masking AES with  $d+1$  Shares in Hardware. In *CHES 2016*, volume 9813 of *Lecture Notes in Computer Science*, pages 194–212. Springer, 2016.
- [Dae17] Joan Daemen. Changing of the Guards: A Simple and Efficient Method for Achieving Uniformity in Threshold Sharing. In *CHES 2017*, volume 10529 of *Lecture Notes in Computer Science*, pages 137–153. Springer, 2017.
- [Dwo15] Morris J Dworkin. SHA-3 standard: Permutation-based hash and extendable-output functions. Technical report, 2015.

- [FGP<sup>+</sup>18a] Sebastian Faust, Vincent Grosso, Santos Merino Del Pozo, Clara Paglialonga, and François-Xavier Standaert. Composable Masking Schemes in the Presence of Physical Defaults & the Robust Probing Model. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2018(3):89–120, 2018.
- [FGP<sup>+</sup>18b] Sebastian Faust, Vincent Grosso, Santos Merino Del Pozo, Clara Paglialonga, and François-Xavier Standaert. Composable Masking Schemes in the Presence of Physical Defaults & the Robust Probing Model. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2018(3):89–120, 2018.
- [GBTP08] Benedikt Gierlichs, Lejla Batina, Pim Tuyls, and Bart Preneel. Mutual Information Analysis. In *CHES 2008*, volume 5154 of *Lecture Notes in Computer Science*, pages 426–442. Springer, 2008.
- [GIS14] Hendra Guntur, Jun Ishii, and Akashi Satoh. Side-channel attack user reference architecture board SAKURA-G. In *GCCE 2014*, pages 271–274. IEEE, 2014.
- [GMK16] Hannes Gross, Stefan Mangard, and Thomas Korak. Domain-Oriented Masking: Compact Masked Hardware Implementations with Arbitrary Protection Order. In *TIS@CCS 2016*, page 3. ACM, 2016.
- [GSM17a] Hannes Groß, David Schaffenrath, and Stefan Mangard. Higher-Order Side-Channel Protected Implementations of KECCAK. In *DSD 2017*, pages 205–212. IEEE Computer Society, 2017.
- [GSM17b] Hannes Groß, David Schaffenrath, and Stefan Mangard. Higher-Order Side-Channel Protected Implementations of Keccak. *IACR Cryptol. ePrint Arch.*, 2017:395, 2017.
- [ISW03] Yuval Ishai, Amit Sahai, and David A. Wagner. Private Circuits: Securing Hardware against Probing Attacks. In *CRYPTO 2003*, volume 2729 of *Lecture Notes in Computer Science*, pages 463–481. Springer, 2003.
- [KJJ99] Paul C. Kocher, Joshua Jaffe, and Benjamin Jun. Differential Power Analysis. In *CRYPTO '99*, volume 1666 of *Lecture Notes in Computer Science*, pages 388–397. Springer, 1999.
- [KSM20] David Knichel, Pascal Sasdrich, and Amir Moradi. SILVER - Statistical Independence and Leakage Verification. In *ASIACRYPT 2020*, volume 12491 of *Lecture Notes in Computer Science*, pages 787–816. Springer, 2020.
- [MM13] Amir Moradi and Oliver Mischke. On the Simplicity of Converting Leakages from Multivariate to Univariate - (Case Study of a Glitch-Resistant Masking Scheme). In *CHES 2013*, volume 8086 of *Lecture Notes in Computer Science*, pages 1–20. Springer, 2013.
- [MMW18] Lauren De Meyer, Amir Moradi, and Felix Wegener. Spin Me Right Round Rotational Symmetry for FPGA-Specific AES. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2018(3):596–626, 2018.
- [MPL<sup>+</sup>11] Amir Moradi, Axel Poschmann, San Ling, Christof Paar, and Huaxiong Wang. Pushing the Limits: A Very Compact and a Threshold Implementation of AES. In *EUROCRYPT 2011*, volume 6632 of *Lecture Notes in Computer Science*, pages 69–88. Springer, 2011.
- [MS16] Amir Moradi and François-Xavier Standaert. Moments-Correlating DPA. In *Theory of Implementation Security - TIS@CCS 2016*, pages 5–15. ACM, 2016.

- 
- [NRS11] Svetla Nikova, Vincent Rijmen, and Martin Schl affer. Secure Hardware Implementation of Nonlinear Functions in the Presence of Glitches. *J. Cryptology*, 24(2):292–321, 2011.
- [PMK<sup>+</sup>11] Axel Poschmann, Amir Moradi, Khoongming Khoo, Chu-Wee Lim, Huaxiong Wang, and San Ling. Side-Channel Resistant Crypto for Less than 2, 300 GE. *J. Cryptol.*, 24(2):322–345, 2011.
- [PRB09] Emmanuel Prouff, Matthieu Rivain, and R egis Bevan. Statistical Analysis of Second Order Differential Power Analysis. *IEEE Trans. Computers*, 58(6):799–811, 2009.
- [RBN<sup>+</sup>15] Oscar Reparaz, Beg ul Bilgin, Svetla Nikova, Benedikt Gierlichs, and Ingrid Verbauwhede. Consolidating Masking Schemes. In *CRYPTO 2015*, volume 9215 of *Lecture Notes in Computer Science*, pages 764–783. Springer, 2015.
- [Rep15] Oscar Reparaz. A note on the security of Higher-Order Threshold Implementations. *IACR Cryptol. ePrint Arch.*, 2015:1, 2015.
- [SD17] Niels Samwel and Joan Daemen. DPA on hardware implementations of Ascon and Keyak. In *CF 2017*, pages 415–424. ACM, 2017.
- [SM15] Tobias Schneider and Amir Moradi. Leakage Assessment Methodology - A Clear Roadmap for Side-Channel Evaluations. In *CHES 2015*, volume 9293 of *Lecture Notes in Computer Science*, pages 495–513. Springer, 2015.
- [SM21] Aein Rezaei Shahmirzadi and Amir Moradi. Re-Consolidating First-Order Masking Schemes Nullifying Fresh Randomness. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2021(1):305–342, 2021.
- [Tri03] Elena Trichina. Combinational Logic Design for AES SubByte Transformation on Masked Data. *IACR Cryptol. ePrint Arch.*, 2003:236, 2003.
- [WM18] Felix Wegener and Amir Moradi. A First-Order SCA Resistant AES Without Fresh Randomness. In *COSADE 2018*, volume 10815 of *Lecture Notes in Computer Science*, pages 245–262. Springer, 2018.