IACR Transactions on Cryptographic Hardware and Embedded Systems
ISSN 2569-2925, Vol. 2024, No. 4, pp. 231–257. DOI:10.46586/tches.v2024.i4.231-257

Breaking Ground: A New Area Record for Low-Latency First-Order Masked SHA-3

Advancing from the 4× Area Era to the 3× Area Era

Cankun Zhao[1], Hang Zhao[1], Jiangxue Liu[1], Bohan Yang[1], Wenping Zhu[1], Shuying Yin[1], Min Zhu[2], Shaojun Wei[1] and Leibo Liu[1,*]

[1] Beijing National Research Center for Information Science and Technology, School of Integrated Circuits, Tsinghua University, Beijing, China, {zck22,zhao-h21,liujx21}@mails.tsinghua.edu.cn;{bohanyang,zhuwp,yinshuying,wsj,liulb}@tsinghua.edu.cn

[2] Wuxi Micro Innovation Integrated Circuit Design Co., Ltd., Wuxi, China, zhumin@mucse.com

[*] Corresponding author.

Abstract. SHA-3, the latest hash standard from NIST, is utilized by numerous cryptographic algorithms to handle sensitive information. Consequently, SHA-3 has become a prime target for side-channel attacks, with numerous studies demonstrating successful breaches in unprotected implementations. Masking, a countermeasure capable of providing theoretical security, has been explored in various studies to protect SHA-3. However, masking for hardware implementations may significantly increase area costs and introduce additional delays, substantially impacting the speed and area of higher-level algorithms. In particular, current low-latency first-order masked SHA-3 hardware implementations require more than four times the area of unprotected implementations. To date, the specific structure of SHA-3 has not been thoroughly analyzed for exploitation in the context of masking design, leading to difficulties in minimizing the associated area costs using existing methods. We bridge this gap by conducting detailed leakage path and data dependency analyses on two-share masked SHA-3 implementations. Based on these analyses, we propose a compact and low-latency first-order SHA-3 masked hardware implementation, requiring only three times the area of unprotected implementations and almost no fresh random number demand. We also present a complete theoretical security proof for the proposed implementation in the glitch+register-transition-robust probing model. Additionally, we conduct leakage detection experiments using PROLEAD, TVLA and VerMI to complement the theoretical evidence. Compared to state-of-the-art designs, our implementation achieves a 28% reduction in area consumption. Our design can be integrated into first-order implementations of higher-level cryptographic algorithms, contributing to a reduction in overall area costs.

Keywords: SHA-3 · Keccak · Masking · Side-Channel Attacks · Glitch · Hardware Implementation · Low Latency

1 Introduction

The Secure Hash Algorithm-3 (SHA-3) is the latest hash standard according to the National Institute of Standards and Technology (NIST). Upon its release in FIPS 202 [NIS15], SHA-3 has been adopted as a common building block in a variety of cryptographic schemes. Many of these schemes use SHA-3 to process some form of secret input. For example, in NIST's Post-Quantum Cryptography (PQC) project, SHA-3 is utilized in multiple key encapsulation mechanisms (KEMs) and digital signature schemes to process secret seeds, generate secret pseudorandom numbers, and directly handle private keys

Received: 2024-04-15 Accepted: 2024-06-15 Published: 2024-09-05

[SAB+22, LDK+22, HBD+22]. Moreover, in NIST's recommendation SP800-185 [KjCP16], the Keccak Message Authentication Code (KMAC) employs SHA-3 for processing secret keys. These applications make SHA-3 a potential single point of failure of the system.

Numerous studies have investigated side-channel attacks targeting SHA-3. In the case of both hardware and software implementations of KMAC, differential power analysis (DPA) has been demonstrated to be effective in key recovery [LFF+14, LFF+15, TS13b, TS13a]. Additionally, for software implementations of SHA-3, template attacks have been shown to be able to recover inputs [KPP20, YK20, YK21]. These works highlight the importance of safeguarding SHA-3 against side-channel attacks.

Masking is one of the most investigated countermeasures against side-channel attacks, and its theoretical security has been thoroughly analyzed and proven in several formal models [CJRR99, ISW03, PR13, DDF14]. Masking schemes randomly split a secret into several shares, severing the connection between the secret and the low-order statistical moments of the side-channel leakage distributions [BDF+17]. As a result, attackers must rely on higher-order statistical moments to extract secret information, a task that becomes exponentially difficult with sufficient noise [BDF+17]. Exploiting second-order information for attacks exponentially increases costs, which is especially challenging for highly parallelized hardware with high noise levels. Thus, first-order hardware masking schemes are particularly appealing and have been the subject of considerable research [BNN+12, SM21a, KM22] due to the good trade-off between implementation cost and security. Thus, this paper focuses on first-order hardware masking for SHA-3.

In multiple application scenarios of masking SHA-3, speed is a critical factor. For the PQC algorithms Kyber and Dilithium, SHA-3 operations constitute a significant portion of the computational workload [ABCG20, GKS21], and they can become a speed bottleneck in masked implementations [BGR+21, ABC+23]. For KMAC and hash-based signatures, SHA-3 is used for nearly all of the computations [KjCP16, HBD+22]. Therefore, the speed of SHA-3 directly impacts the overall speed of these schemes. In addition, when masking schemes are applied in hardware implementations, glitches in combinational logic may introduce additional leakage [FGP+18]. One direct approach to mitigate this issue is to use additional registers; however, this approach leads to additional delays [ABP+18, SM21b]. Consequently, numerous studies have focused on reducing the latency of masked hardware implementations of SHA-3 [BDN+13, ABP+18, ZSS+21], preventing additional delays introduced by side-channel countermeasures.

Hardware masking of SHA-3 also results in substantial area overhead. Given that the SHA-3 module typically occupies a high proportion of the area of the cryptographic hardware engine [ZZW+22, AMI+23], masking SHA-3 significantly influences the overall area needed by the cryptographic implementation. Several studies have attempted to minimize the area of masked round-based implementations of SHA-3. These works can be classified into two primary categories based on the masking method: threshold implementation (TI)-based methods [NRR06] and domain-oriented masking (DOM)-based methods [GMK16]. Among them, TI-based methods [BDN+13, ABP+18] require more than four times the area of unprotected SHA-3 implementations, and since the number of shares already approaches the theoretical limit, further optimization within the TI framework becomes challenging. DOM-based implementations [GSM17a, ZSS+21] need more than three times the area, introducing either additional registers, which introduce additional cycles, or high random number consumption, which leads to significant additional area costs for the pseudorandom number generator (PRNG). For existing masking methods, reducing the area of low-latency masked SHA-3 proves to be highly challenging.

SHA-3 functions are based on instances of the Keccak algorithm, and the number of state bits b can vary from 25 to 1600. Several existing works [ABP+18, ZSS+21] have focused on optimizing the masking scheme for the general Keccak. However, SHA-3 exclusively utilizes the $b = 1600$ version of Keccak, and an optimal masking scheme for this particular version has not yet been developed.

In this paper, we propose a compact and low-latency first-order masked SHA-3 hardware implementation, achieving a new area record. Our contributions are summarized as follows:

- We conducted a detailed analysis of leakage paths and data dependencies for the $b = 1600$ version of the DOM masking of Keccak, which enables more refined SHA-3 masking designs.
- We propose a compact and low-latency first-order masked SHA-3 hardware implementation, reducing the area cost by 28% compared to state-of-the-art designs and consuming only 2 bits of fresh random numbers per cycle.
- We provide theoretical security proofs for the proposed design under the glitch+register-transition-robust probing model. Furthermore, we experimentally evaluated the security of our design using PROLEAD, TVLA, and VerMI.

We release the RTL codes of our design and all the programs we used during the security proofs of our design at https://github.com/zck15/CLLFO-SHA-3.

Outline. Section 2 provides background knowledge on Keccak and hardware masking. Section 3 presents the leakage analysis of SHA-3's DOM masking and our design rationale. Section 4 describes the details of our hardware architecture. Section 5 provides theoretical security proofs under robust probing models. Section 6 presents the experimental leakage evaluation results. Section 7 presents the synthesis results and comparisons with related works. Finally, Section 8 provides conclusions and discussions on composability.

Related Works. The designers of Keccak [BDPA10a] proposed a first-order hardware masking Keccak design with three shares based on TI. However, this design failed to achieve uniformity and is thus not provably secure [BDN+13]. Bilgin et al. [BDN+13] proposed enhanced first-order TI-based Keccak masking designs, including a 3-share version requiring 4-bit fresh randomness per cycle and a 4-share version requiring no fresh randomness, with area overheads 4.1 and 5.0 times greater than the area of the unprotected design, respectively. Daemen [Dae17] extended the method used in [BDN+13] and introduced the famous Changing of the Guards (COTG) method, which can achieve uniformity at a low cost of fresh randomness. Arribas et al. [ABP+18] proposed a two-round unrolled TI-based architecture using 5 to 10 shares, reducing the number of clock cycles by half but requiring several tens of times the area cost. Due to the limitations of the TI method, all first-order TI-based Keccak implementations require at least 3 shares, limiting further area reduction. Additionally, these works are only designed to resist glitch-induced leakage, without considering potential leakage introduced by transitions.

Some other works have attempted to use the minimum number of shares to achieve smaller-area Keccak masking implementations, i.e., $d + 1$ shares for d-th order security. Gross et al. [GSM17b] presented arbitrary-order $d+1$-share Keccak masking designs based on DOM. Their first-order implementation requires only 3 times the original area without fresh randomness but fails to achieve uniformity. Additionally, in their Keccak-$f[200]$ implementations, Arribas et al. [ABP+18] identified glitch-induced leaks. To mitigate glitch propagation, Gross et al. added an additional register stage in an updated version of the implementation [GSM17a], resulting in double the number of clock cycles. Zarei et al. [ZSS+21] proposed single-cycle-per-round implementations that are resilient to glitches. These implementations are based on DOM and uses $d + 1$ shares for dth-order security. However, a significant number of logic circuits are used to resist glitch-induced leakage, and a large amount of randomness is applied to achieve uniformity. Shahmirzadi et al. [SM21b] proposed first- and second-order Keccak nonlinear layers with $d + 1$ shares that achieve uniformity without fresh randomness. However, they need 3 clock cycles per round

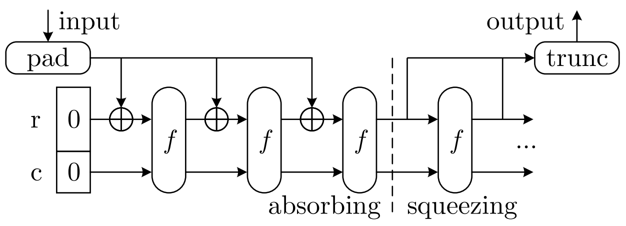


Figure 1: The Keccak sponge function. f represents permutation Keccak-$f[b]$.

to prevent glitch-induced leakage, and their first-order implementation requires 4.2 times the area. These $d+1$-share works either require more cycles per round or maintain an area overhead of more than 4 times the original level. Additionally, these works only consider glitch-robust security and do not consider transitions.

The S-boxes of Xoodyak and Ascon are similar to those of Keccak. Peng et al. [PYY+23] proposed a low-randomness first-order Xoodyak implementation. Their S-box structure is similar to that of Gross for Keccak [GSM17b] and thus suffers from glitch-induced leakage and nonuniformity when applied to Keccak. Prasad et al. [PMSN23] proposed randomness-free Ascon implementations, which use two cycles per round to resist glitches and thus cannot be directly used for single-cycle-per-round Keccak masking designs.

2 Notations and Preliminaries

2.1 Notations

We denote variables with lower-case italic a, specific values with upper-case italic A, i-th share of a variable with subscripts a_i, set of all shares with hats $\hat{a} = (a_1, a_2, \dots)$, and vectors with bold italic $\boldsymbol{a}$.

For variables in the Keccak operation, we denote coordinates in the state array in triple form as $[x, y, z]$ or in pair form as $[x, j]$, where $j = 64y + z$ represents the row index. When describing relationships that apply to every row, we ignore row indices and use $[x]$ to denote the coordinates. We denote variables in the n-th round with superscripts a^n, input for the n-th round with s^n, output of θ with t^n, output of ρ with u^n, output of π with v^n, output of χ with w^n, flip-flops in χ with d^n, result of $\pi(\rho(s^n))$ (bypassing θ) with s'^n, fresh random bits with r^n, and guards generated using COTG-like methods with g^n.

2.2 Keccak

Keccak is a family of sponge functions that forms the basis of the NIST standard SHA-3 [NIS15]. All six functions standardized in SHA-3 are instances of the Keccak sponge functions (Figure 1) with different parameters. The primary operation in these sponge functions is the Keccak permutation, known as Keccak-$f[b]$. The Keccak-$f[b]$ permutation family, in its original definition, includes seven permutations with different bit widths b, ranging from 25 to 1600. In the SHA-3 standard, only the permutation Keccak-$f[1600]$ is utilized, which is the protection target of this paper.

Keccak-$f[1600]$ operates on a three-dimensional 1600-bit state array $\boldsymbol{s}$, where the bit at coordinates (x, y, z) is denoted as $s[x, y, z]$. The Keccak-$f[1600]$ permutation consists of 24 rounds, with each round composed of the following 5 steps:

$$round = \iota \circ \chi \circ \pi \circ \rho \circ \theta, \text{with}$$

$$\begin{aligned}
\theta :&\quad t^n[x,y,z] \leftarrow s^n[x,y,z] \oplus \bigoplus_{y'=0}^{4} s^n[x-1,y',z] \oplus \bigoplus_{y'=0}^{4} s^n[x+1,y',z-1],\\
\rho :&\quad u^n[x,y,z] \leftarrow t^n[x,y,z-\mathrm{OFFS}_\rho[x,y]],\ \text{where } \mathrm{OFFS}_\rho \text{ are constant offsets},\\
\pi :&\quad v^n[x,y,z] \leftarrow u^n[x+3y,x,z],\\
\chi :&\quad w^n[x,y,z] \leftarrow v^n[x,y,z] \oplus \overline{v^n[x+1,y,z]}v^n[x+2,y,z],\\
\iota :&\quad \boldsymbol{s}^{n+1} \leftarrow \boldsymbol{w}^n \oplus \mathrm{RC}^n,\ \text{where } \mathrm{RC}^n \text{ is the round constant for round } n.
\end{aligned}$$

Note that because the size of the state array $\boldsymbol{s}$ is $5 \times 5 \times 64$, the addition and subtraction operations for the x and y coordinates are modulo 5, while the operations for the z coordinate are modulo 64. The values of OFFS_ρ and RC can be found in Table 2 of [NIS15] and Section 3.2.5 of [NIS15], respectively.

2.3 Masking

Masking schemes split sensitive data randomly into several shares; as a result, an adversary needs all shares to obtain sensitive information. Formal theoretical analysis indicates that the security of masking relies on two fundamental requirements [PR13]: each side-channel sample depends on a limited number of shares (ideally one), and leakage from shares contains sufficient noise. The first requirement of independence is closely tied to algorithmic-level scheme design, which is the primary focus of this paper. The second requirement of noise is more circuit-level and product specific, and we refer to [GS18] for further discussion on this aspect.

To assess whether specific masking schemes satisfy the independence requirement, various attacker models have been developed to evaluate this aspect of the design, as introduced in Section 2.3.1. Multiple masking design frameworks have been developed to meet the independence requirement; here, we introduce the TI and DOM frameworks in Sections 2.3.2 and 2.3.3, respectively.

2.3.1 Probing Security

To assess the independence between side-channel leakage and secret information, probing security models, introduced by Ishai et al. [ISW03], are widely employed. In the probing security model, a d^{th}-order attacker directly reads the intermediate values on d wires in the circuit. If a circuit does not leak any sensitive information when facing such attackers, it is referred to as a d^{th}-order probing secure. The probing security model does not rely on specific leakage models or attack methods but rather directly examines the independence between intermediate variables and sensitive information, aiming to ensure the first fundamental assumption in the theoretical security of masking operations.

The probing security model works well in software masking implementations. However, in hardware implementations, physical defaults such as glitches and transitions can generate side-channel information that combines multiple intermediate variables [MPG05, MPO05]. This may reduce the security in the bounded order model [BDF+17] and cannot be detected by the probing security model. To address this issue, Faust et al. [FGP+18] proposed a robust probing security model that considers the possible combinations caused by physical defaults. For combinational logic circuits, applying a glitch-extended probe on any output of a gadget allows an adversary to observe all the inputs to the gadget. For memory cells, applying a transition-extended probe on a register allows an adversary to observe any pair of values stored in two consecutive invocations. If a design does not leak sensitive information when facing glitch-extended or transition-extended probes, the design is termed glitch-robust or transition-robust probing secure. Cassiers et al. [CS21] proposed a more conservative model considering the transitions of all wires rather than just transitions of memory cells. To distinguish between these two models, we refer to them as the register-transition-robust and wire-transition-robust probing models, respectively.

2.3.2 Threshold Implementation

Threshold implementation (TI), introduced by Nikova et al. [NRR06], is a popular hardware masking method due to its easily understandable security requirements, particularly for first-order masking. TI requires that the circuit satisfies the non-completeness property, preventing any combinatorial logic operations from having access to all the shares of sensitive variables and thus preventing leaks caused by glitches. The specific requirements for first-order TI are as follows. A target function $z = f(x, y, \dots)$ is divided into n component functions $f_1, f_2, \dots, f_n$. The component functions must satisfy the following two properties:

Property 1 (Non-completeness [NRR06])**.** Every function is independent of at least one share of each of the input variables $x, y, \dots$.

Property 2 (Correctness [NRR06])**.** The sum of the output shares gives the desired output.

For input variables $x, y, \dots$, their shares $\hat{x} = (x_1, x_2, \dots, x_m), \hat{y} = (y_1, y_2, \dots, y_m), \dots$ must satisfy the following property:

Property 3 (Uniformity [NRR06])**.** The conditional probability distribution $\Pr(\hat{x} = \hat{X}, \hat{y} = \hat{Y}, \dots | x = X, y = Y, \dots)$ of every possible sharing vector is uniform:

$$\Pr(\hat{x} = \hat{X}, \hat{y} = \hat{Y}, \dots) = c \Pr(x = \bigoplus_i X_i, y = \bigoplus_i Y_i, \dots).$$

Nikova et al. [NRR06] proved that implementations satisfying the above conditions are first-order secure in the presence of glitches, as stated in the following theorem.

Theorem 1 ([NRR06])**.** *In a circuit implementing a set of functions satisfying Property 1 and Property 2, when the input satisfies Property 3, all the intermediate results are independent of the inputs $x, y, \dots$ and the output z. Additionally, the power consumption and any other characteristic of each individual function f_i are independent of $x, y, \dots$ and z.*

Since the input for the subsequent function is formed by the output of the previous function, the input uniformity of the subsequent function can be achieved by ensuring that the previous function satisfies the balanced property.

Property 4 (Balance [NRR06])**.** Functions are balanced if for all input share distributions satisfying Property 3, the output distribution satisfies Property 3.

This property holds if the function is invertible. However, achieving the balance property with a minimal number of shares is sometimes not trivial. In theory, the lower bound for the number of shares in a TI of the χ function in SHA-3 is 3 [NRR06]. However, a balanced 3-share TI for χ has not yet been found [BDN$^+$13].

2.3.3 Domain-Oriented Masking

Domain-oriented masking (DOM), introduced by Gross et al. [GMK16], aims to provide d^{th}-order security using only $d + 1$ shares. In DOM implementations, each share of a variable is associated with one share domain. For linear operations, calculations are performed within each domain. For nonlinear operations, the authors proposed two types of AND gates, namely, DOM-indep and DOM-dep, for cases in which the two inputs are

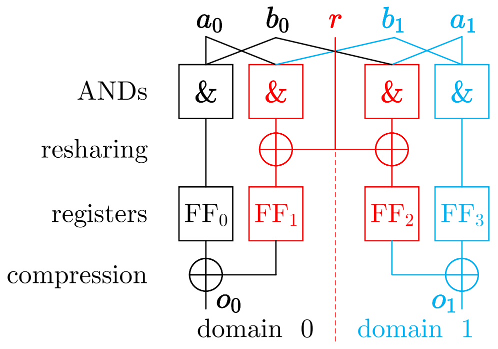


Figure 2: First-Order DOM-Indep Gate.

independent and dependent, respectively. We focus on the first-order DOM-indep gate (Figure 2), which we abbreviate as the DOM gate for brevity. The calculation process of the DOM gate involves computing products, including two inner-domain products and two cross-domain products; resharing the cross-domain products with a fresh random bit; and compressing the outputs of registers within the same domain. Two points must be considered when using the DOM scheme. First, the shares of the two inputs to the DOM gate must be independent to prevent potential leakage in the cross-domain products. Second, each DOM gate requires a fresh random bit, which may lead to a relatively high overall demand for fresh random numbers when many DOM gates are used.

3 Leakage Analysis and Design Rationale

In this section, we analyze potential leaks in two-share DOM-based Keccak masking schemes and rationalize our design choices. Section 3.1 discusses leaks in the DOM-Keccak scheme [GSM17b], clarifying that the leaks reported by Arribas et al. [ABP+18] do not exist in the $f[1600]$ version and pointing out two additional issues: a new source of leakage and uniformity. Section 3.2 introduces the new source of leakage and discusses its classification, sources and countermeasures. Section 3.3 discusses solutions to the uniformity issue and its impact on potential leakage. Section 3.4 presents our final design choices and explains the reasons for selecting this approach.

3.1 Analysis of Leakage in DOM-Keccak

Gross et al. [GSM17b] proposed a two-share DOM-based masking implementation of Keccak. In their design, linear operations are performed separately on each of the two shares, and the AND gates in χ are replaced with DOM gates. Arribas et al. [ABP+18] reported that the non-completeness property is compromised in the $f[200]$ version and verified the leakage through experiments. The reason for this leakage is that the inputs to the DOM gates in χ come from the outputs of the preceding linear operations, and the diffusion in θ causes the two input variables of some DOM gates to be correlated with a common θ input variable. In other words, under the glitch-robust probing model, a probe on the cross-domain terms in the DOM gate can extend to both shares of the related θ input simultaneously. This implies that side-channel information induced by glitches may reveal the secret value of the θ input.

We examined the $f[1600]$ version of DOM-Keccak and found that the same type of leakage did not occur. The reason is discussed as follows. The diffusion in θ is confined to states with the same and neighboring z values, while the subsequent ρ operation disperses the states in the z direction. As $f[1600]$ has a larger z-direction length, with $W = 64$, the

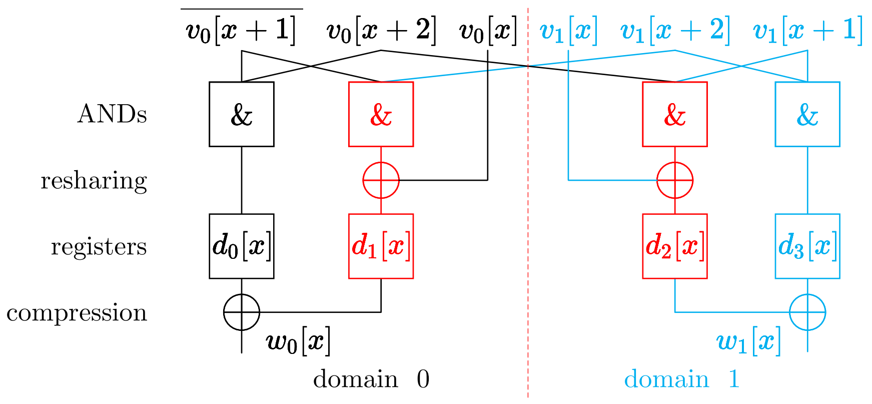


Figure 3: Modified DOM Gate with No Fresh Random Number for χ Operations by [GSM17b].

related states become more scattered in the z direction after ρ. Consequently, in the χ operations, for which the inputs come from states in the same row and thus have the same z coordinates, the inputs are no longer correlated. This finding indicates that there is no first-order leakage under the glitch-robust probing model for the first-round operation of DOM-Keccak-f[1600] due to the non-completeness property.

However, this security holds only for the first round. There are two other issues in subsequent rounds: there is a new source of leakage, and the input of these rounds does not satisfy the uniformity property. Regarding the new leakage issue, starting from the second round, glitch-extended probes can be extended to the DOM registers $\boldsymbol{d}_i^{n-1}$ in the previous round instead of being confined to the compressed round input $\boldsymbol{s}_i^n$. The knowledge of these registers' values may reveal some unshared values of the corresponding χ^{n-1} inputs $\boldsymbol{v}^{n-1}$. Regarding the uniformity property, the authors replaced the fresh random bit in the original DOM gate with adjacent χ input shares (Figure 3). This randomness optimization prevents the χ operation from maintaining the uniformity of the state, as noted by [GSM17b]. This implies that starting from the second round, the input does not satisfy the uniformity property, making it challenging to guarantee theoretical security.

3.2 New Leakage Types and Countermeasures

In this section, a new source of leakage in DOM-Keccak-f[1600] occurring after the first round is discussed. In the n-th round, a glitch-extended probe can obtain the corresponding DOM register values $\boldsymbol{d}_i^{n-1}$. Known values of multiple registers in the same row may reveal the unshared values of their common χ inputs. We categorize this potential leakage into two types, as shown in Table 1. Below, we elaborate on how each type arises and discuss possible countermeasures.

Table 1: New Leakage Types in DOM-Keccak-f[1600]: Each Row Represents a Potential Leak Where a Glitch-Extended Probe's Input List Contains Multiple Registers in the Same Row.

Type I (Inner-Domain)	Type II (Inner-Domain)	Type II (Cross-Domain)
$\boldsymbol{d}_{0,1}^{n-1}[x], \boldsymbol{d}_{0,1}^{n-1}[x+1]$	$\boldsymbol{d}_{0,1}^{n-1}[0], \boldsymbol{d}_{0,1}^{n-1}[3]$	$\boldsymbol{d}_{0,1}^{n-1}[0], \boldsymbol{d}_{2,3}^{n-1}[3]$
$\boldsymbol{d}_{2,3}^{n-1}[x], \boldsymbol{d}_{2,3}^{n-1}[x+1]$	$\boldsymbol{d}_{2,3}^{n-1}[0], \boldsymbol{d}_{2,3}^{n-1}[3]$	$\boldsymbol{d}_{2,3}^{n-1}[0], \boldsymbol{d}_{0,1}^{n-1}[3]$
	$\boldsymbol{d}_{0,1}^{n-1}[0], \boldsymbol{d}_{0,1}^{n-1}[1], \boldsymbol{d}_{0,1}^{n-1}[3]$	$\boldsymbol{d}_{0,1}^{n-1}[0], \boldsymbol{d}_{0,1}^{n-1}[1], \boldsymbol{d}_{2,3}^{n-1}[3]$
	$\boldsymbol{d}_{2,3}^{n-1}[0], \boldsymbol{d}_{2,3}^{n-1}[1], \boldsymbol{d}_{2,3}^{n-1}[3]$	$\boldsymbol{d}_{2,3}^{n-1}[0], \boldsymbol{d}_{2,3}^{n-1}[1], \boldsymbol{d}_{0,1}^{n-1}[3]$

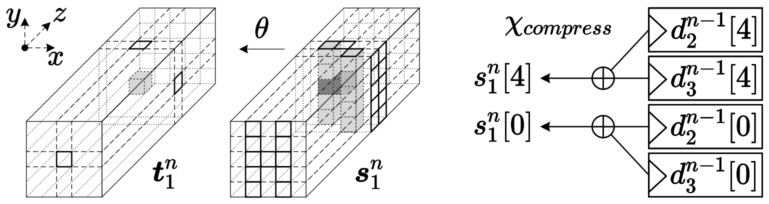


Figure 4: Type I Leakage Illustration: A Glitch-Extended Probe's Input List Contains Four X-Axis Adjacent Registers in the Same Domain.

Table 2: An Example of Type I Leakage: Solving Equations with Given Register Values Reveals All Possible Input Vectors Corresponding to a Fixed Unshared Value.

$d_2[4]$	$d_3[4]$	$d_2[0]$	$d_3[0]$	$v_1[4]$	$v_1[0]$	$v_1[1]$	$v_0[1]$	$v_1[2]$	$v_0[2]$	$v[2]$
1	1	1	1	0	1	1	1	**1**	**0**	**1**
1	1	1	1	1	1	1	0	**1**	**0**	**1**

Type I leakage is illustrated in Figure 4. When a glitch-extended probe is applied to any output of the θ operation, it extends to four consecutive DOM registers in the same row. An example is provided in Table 2, which demonstrates how the values of these four registers reflect the secret value. When all four register values are read as ones, all possible input vectors correspond to the sensitive value $v[2] = 1$. This type of leakage occurs ubiquitously, as any two adjacent x-axis χ outputs in the same row will be XORed together in some θ of the subsequent round.

Type II leakage occurs in 384 glitch-extended probes preceding registers $\boldsymbol{d}_{0,1,2,3}^n[0,0,z]$ and $d_{1,2}^n[1,0,z]$. An example of a probe placed on $d_i^n[0,0,44]$ is shown in Figure 5. All these probes simultaneously extend to χ inputs $v_i^n[1,0,z]$ and $v_k^n[2,0,z]$, which are the causes of leakage. The corresponding indices for these χ inputs before steps π, ρ and θ are as follows:

$$\begin{aligned}
v_i^n[1,0,z] \leftarrow u_i^n[1,1,z] \leftarrow t_i^n[1,1,z-44] &\leftarrow \boldsymbol{s}_i^n[0,y,z-44], s_i^n[1,1,z-44], \boldsymbol{s}_i^n[2,y,z-45] \\
&\leftarrow \boldsymbol{d}_{2i,2i+1}^{n-1}[0,y,z-44], \boldsymbol{d}_{2i,2i+1}^{n-1}[1,1,z-44], \boldsymbol{d}_{2i,2i+1}^{n-1}[2,y,z-45] \\
v_k^n[2,0,z] \leftarrow u_k^n[2,2,z] \leftarrow t_k^n[2,2,z-43] &\leftarrow \boldsymbol{s}_k^n[1,y,z-43], s_k^n[2,2,z-43], \boldsymbol{s}_k^n[3,y,z-44] \\
&\leftarrow \boldsymbol{d}_{2k,2k+1}^{n-1}[1,y,z-43], \boldsymbol{d}_{2k,2k+1}^{n-1}[2,2,z-43], \boldsymbol{d}_{2k,2k+1}^{n-1}[3,y,z-44]
\end{aligned}$$

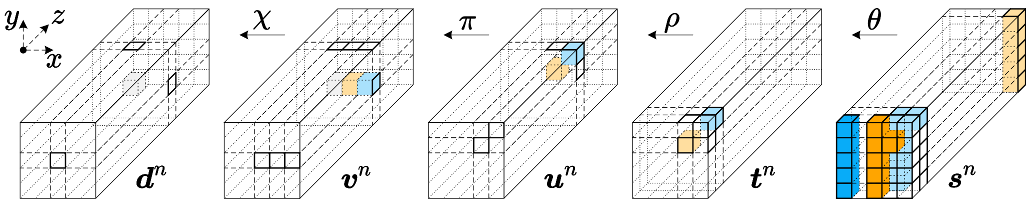


Figure 5: Type II Leakage Illustration: The Glitch-Extended Probe on $d_i^n[0,0,44]$ Extends to θ Inputs in the Same Row.

The corresponding θ inputs $\boldsymbol{s}_i^n[\dots]$ have no overlap; thus, they do not impact the security in the first round. However, they include θ inputs in the same row, highlighted in orange and cyan. This causes the values in the preceding registers $\boldsymbol{d}_{2i,2i+1}^{n-1}[\dots]$ to be related to some common χ inputs. Placing a probe on $d_{0/3}^n[0,0,z]$ or $d_{1/3}^n[1,0,z]$ will simultaneously extend to χ inputs $v_i^n[1,0,z]$ and $v_k^n[2,0,z]$ from the same domain ($i = k$), causing five inner-domain Type II leaks. Placing a probe on $d^n_{1/2}[0, 0, z]$ will simultaneously extend to χ inputs $v^n_i[1, 0, z]$ and $v^n_k[2, 0, z]$ from different domains ($i \neq k$), causing five cross-domain Type II leaks.

Next, we discuss countermeasures against these leaks. Type I leaks occur in all x-adjacent registers in the same domain, so we aim to develop countermeasures with minimal cost. We tested all possible variations of no-randomness DOM gates in Figure 3, including different positions of the NOT gates and whether $v_{0,1}[x]$ were XORed with the inner-domain products or cross-domain products. We found that the joint distribution of the four registers' values in the Type I case is statistically independent of sensitive information when both of the following conditions are satisfied. First, the NOT operation on $v[x + 1]$ is dispersed on products in different domains, i.e., either $(\overline{v_0[x + 1]}v_0[x + 2], \overline{v_1[x + 1]}v_1[x + 2])$ or $(\overline{v_0[x + 1]}v_1[x+2], \overline{v_1[x + 1]}v_0[x+2])$. Second, $v^{[x]}_{0,1}$ are XORed with cross-domain products, not inner-domain products.

For Type II leaks, only $(\boldsymbol{d}^{n-1}_{0,1}[0], \boldsymbol{d}^{n-1}_{0,1}[3])$ and $(\boldsymbol{d}^{n-1}_{2,3}[0], \boldsymbol{d}^{n-1}_{2,3}[3])$ can be addressed using the above method. A possible countermeasure for addressing Type II leaks is to use random numbers to reshare the products of the χ operations for $x = 0, 1, 3$. Note that in Type II leaks, $\boldsymbol{d}^{n-1}_{2i,2i+1}[0]$ and $\boldsymbol{d}^{n-1}_{2i,2i+1}[1]$ are always in the same domain, while $\boldsymbol{d}^{n-1}_{2k,2k+1}[3]$ can be in a different domain from $\boldsymbol{d}^{n-1}_{2i,2i+1}[0]$ and $\boldsymbol{d}^{n-1}_{2i,2i+1}[1]$. For each χ operation, resharing can be performed in several ways: resharing the products before registers d^{n-1}_0 and d^{n-1}_3, denoted as (d_0, d_3); resharing (d_1, d_2); resharing (d_0, d_1) and (d_2, d_3); or resharing (d_0, d_3) and (d_1, d_2). The last method is the most effective but requires two independent random bits.

3.3 Achieving Uniformity

The nonuniformity of the current module's output affects the security of subsequent modules. The χ operation in DOM-Keccak is shown as follows:

$$\begin{aligned} w_0[x] &= v_0[x] \oplus \overline{v_0[x + 1]}v_0[x + 2] \oplus \overline{v_0[x + 1]}v_1[x + 2] \\ w_1[x] &= v_1[x] \oplus v_1[x + 1]v_0[x + 2] \oplus v_1[x + 1]v_1[x + 2] \end{aligned} \tag{1}$$

The output of a single χ operation satisfies uniformity, but the outputs of the five χ operations in the same row do not exhibit a jointly uniform distribution, which can be checked through the noninvertibility of the distribution. A similar situation arises in the 3-share TI-Keccak proposed by the Keccak designers [BDPA10a], as reported in [BDN+13].

To achieve uniformity for this 5-bit S-box, there are two methods discussed in the literature: restricting the input to the original 5 bits and searching for functions that satisfy uniformity [SM21b] or employing additional inputs such as fresh random numbers [ZSS+21] or guards in the COTG method [BDN+13, Dae17]. The first method requires two additional register layers before and after the S-box, resulting in a triple delay. Therefore, our focus and discussion here primarily revolve around the COTG method.

Bilgin et al. [BDN+13] proposed a COTG-like method for 3-share TI-Keccak. The principle can be summarized as follows: the uniformity of $\hat{w}[2]$ is ensured by $\hat{v}[2]$, $\hat{w}[1]$ by $\hat{v}[1]$, and $\hat{w}[0]$ by $\hat{v}[0]$; however, the uniformity of $\hat{w}[3]$ and $\hat{w}[4]$ cannot be ensured by $\hat{v}[3]$ and $\hat{v}[4]$, as $\hat{v}[3]$ and $\hat{v}[4]$ have already appeared in the computation of $\hat{w}[2]$. Therefore, another row's $v_0[3, j']$ and $v_0[4, j']$ are used to ensure this property. This method requires the following conditions: the two coordinates to be uniformed by other rows must be consecutive but can otherwise be arbitrarily chosen. The random numbers ($v_0[3, j']$, $v_0[4, j']$) do not have to be from an adjacent row and only need to be from an unrelated row. As these two rows can be considered connected, all 320 rows should form a cycle-free chain, and two fresh random bits are used at the end of this chain.

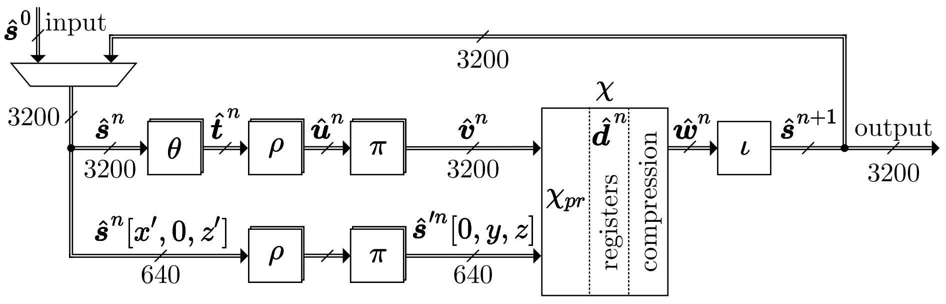


Figure 6: Hardware Architecture.

However, this method introduces additional challenges to ensuring non-completeness. Using χ inputs from other rows in the χ operation introduces additional data dependencies, providing extra information to glitch-robust probing attackers and potentially exacerbating the new leakage introduced in Section 3.2. Therefore, careful selection of the source rows of randomness is necessary.

3.4 Our Design Choices

For Type I leaks, we employ the countermeasures discussed in Section 3.2, which mitigate such leaks at minimal cost. We address Type II leaks and the issue of uniformity together to simplify the design. We use the approach discussed in Section 3.3 to address uniformity, i.e., resharing the x-axis adjacent outputs of the current row using the two inputs from other rows as random numbers. Moreover, we use these two random numbers to counteract Type II leaks, specifically choosing to reshare $(d[0], d[1])$, $(d[2], d[3])$ or $(d[3], d[4])$. By computing the distribution of the outputs, we find that using only these two random numbers is insufficient for counteracting Type II leaks. Therefore, for a DOM gate at $w^{n-1}[x = 0/1/3, y, z]$, we leverage two shares $\boldsymbol{s}'^{n-1}_{0,1}[x, y, z]$, which are the middle states $\boldsymbol{s}^{n-1}_{0,1}[x', y', z']$ of the corresponding θ inputs $(\boldsymbol{s}^{n-1}_{0,1}[x' - 1, 0 \cdots 4, z'], \boldsymbol{s}^{n-1}_{0,1}[x', y', z'], \boldsymbol{s}^{n-1}_{0,1}[x' + 1, 0 \cdots 4, z' - 1])$ of the χ inputs $\boldsymbol{v}^{n-1}_{0,1}[x, y, z]$, simultaneously resharing products $(d^{n-1}_0[x, y, z], d^{n-1}_1[x, y, z])$ and $(d^{n-1}_2[x, y, z], d^{n-1}_3[x, y, z])$ within the two domains. Compared to using additional χ inputs $\boldsymbol{v}^{n-1}_i$ as random numbers, this approach does not introduce new data dependencies, meaning that glitch-extended probes do not extend to additional registers.

Based on these considerations, the remaining decisions to be made include determining which row's χ inputs v^{n-1}_i should be used as random numbers, the x coordinates of these two χ inputs, whether these two χ inputs are XORed with inner-domain products or cross-domain products, and which DOM gate should be reshared with θ inputs. Regarding the first point, we observe that χ inputs adjacent to the z-axis are more likely to be mutually independent in preceding linear operations. Therefore, we use $j = 64y + z$ to label the row number and aim to use rows with similar j values as random numbers. Concerning the remaining options, we calculate the distribution of register values to identify several viable choices for preventing Type II leaks, which are listed in Table 3. Finally, we explore the design space using a computer program and select the final design parameters, described in Section 4, through algorithmic verification of the security results, as discussed in Section 5.

Table 3: Viable Design Choices for Preventing Type II Leaks.

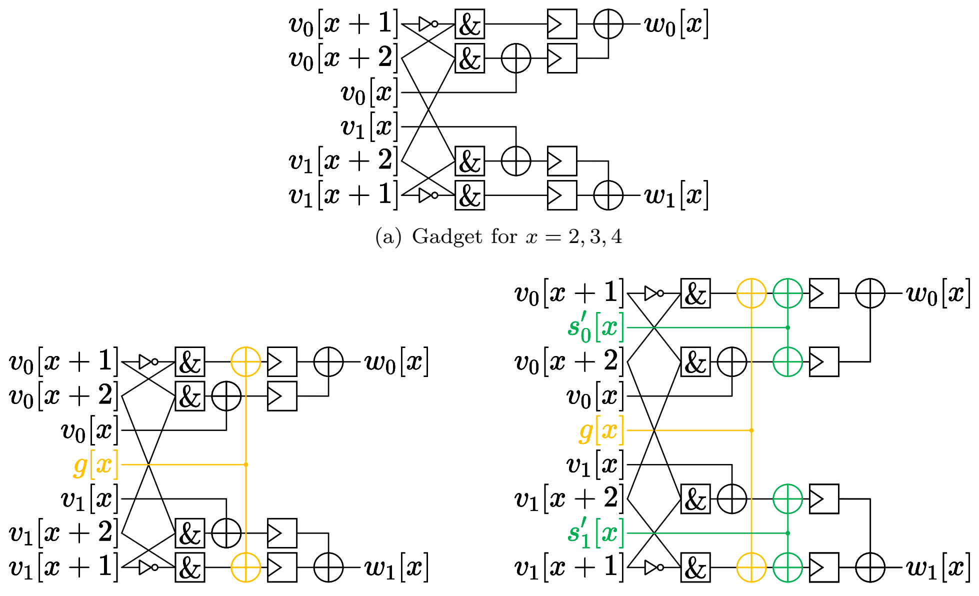
x Coordinates of v^{n-1}_i as Rand.	([0], [1])	([0], [1])	([0], [1])	([2], [3])	([3], [4])
XORed Products[+]	(i/c,i)	(c,i/c)	(c,i/c)	(i/c,i/c)	(i/c,i/c)
x Coordinate of $\hat{s}'^{n-1}$ as Rand.	[0]	[1]	[3]	[3]	[3]

[+] i: inner-domain products; c: cross-domain products.

Table 4: χ_{pr} Operations.

Registers	$x=0$	$x=1$	$x=2,3,4$	Products
$d_0^n[x]$	$p_0^n[x] \oplus g^n[x] \oplus s_0'^n[x]$	$p_0^n[x] \oplus g^n[x]$	$p_0^n[x]$	$p_0^n[x] = \overline{v_0^n[x+1]}v_0^n[x+2]$
$d_1^n[x]$	$p_1^n[x] \oplus v_0^n[x] \oplus s_0'^n[x]$	$p_1^n[x] \oplus v_0^n[x]$	$p_1^n[x] \oplus v_0^n[x]$	$p_1^n[x] = v_0^n[x+1]v_1^n[x+2]$
$d_2^n[x]$	$p_2^n[x] \oplus v_1^n[x] \oplus s_1'^n[x]$	$p_2^n[x] \oplus v_1^n[x]$	$p_2^n[x] \oplus v_1^n[x]$	$p_2^n[x] = v_1^n[x+1]v_0^n[x+2]$
$d_3^n[x]$	$p_3^n[x] \oplus g^n[x] \oplus s_1'^n[x]$	$p_3^n[x] \oplus g^n[x]$	$p_3^n[x]$	$p_3^n[x] = \overline{v_1^n[x+1]}v_1^n[x+2]$

[*] For $j = 64y + z > 0$, $g^n[x,j] = v_0^n[x, j-11]$; and for $j = 0$, $g^n[x,j] = r_x^n$.



(a) Gadget for $x = 2, 3, 4$

(b) Gadget for $x = 1$

(c) Gadget for $x = 0$

Figure 7: The Proposed Gadgets for χ.

4 Hardware Architecture

Our overall hardware architecture is illustrated in Figure 6. The inputs, outputs, and intermediate variables each consist of two uniform shares of their corresponding secret variables. Two θ, π, and ρ circuits operate on the two shares separately, while one ι circuit operates on the first share. Additionally, the 640 bits of θ input, where $y = 0$, bypass θ, pass directly through ρ and π, and are then fed into χ as pseudo-random numbers. The χ operation uses three types of gadgets, as shown in Figure 7.

All gadgets used in χ employ the technique to counteract Type I leakage, which distributes the NOT operation across $v_0[x+1]$ and $v_1[x+1]$. This ensures that the values of the register combinations, corresponding to Type I leakage (Table 1), are statistically independent of the secret values. To resist Type II leakage, we use $g[x]$ to reshare $d_0[x]$ and $d_3[x]$ in the gadget for $x = 1$, and both $g[x]$ and $s'_{0,1}[x]$ to reshare the four registers in the gadget for $x = 0$. These two gadgets ensure that the values of the register combinations, corresponding to Type II leakage (Table 1), are statistically independent of the secret values. The $g[x]$ used in these two gadgets is the guard ensuring uniformity through the COTG method, reused to counteract Type II leakage. The $s'_{0/1}[x]$ in the gadget for $x = 0$ is one of the 11 θ inputs corresponding to $v_{0/1}[x]$. Therefore, using $s'_{0/1}[x]$ does not provide additional information to a glitch-extended probing attacker. These gadgets are designed to introduce minimal extra data dependency while resisting leakage and ensuring uniformity, which ensures simplicity in our design and consequently reduces the area required.

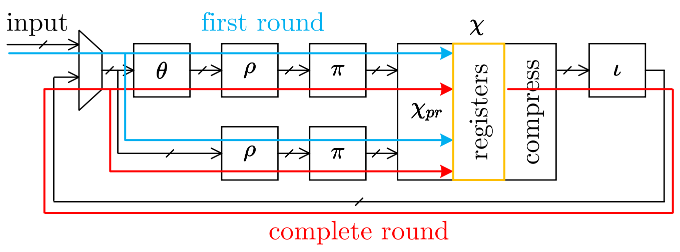


Figure 8: Illustration of the First Round and the Complete Round Processes.

5 Security Proofs

This section provides security proofs for the proposed design under the first-order glitch+register-transition-robust probing model. Given that the probe propagation in the glitch-robust probing model is bounded by registers, we provide security proofs for different stages with the registers as boundaries. As illustrated in Figure 8, we use the term "first round" to refer to the process from the input to the registers and the term "complete round" to denote the process from the registers through one round of combinational logic back to the registers. The data paths for the first round and the complete round are not identical, so we present security proofs for the first round and the complete round in Sections 5.1 and 5.2, respectively, under the glitch-robust probing model. In Section 5.3, we provide security proofs under the combined glitch+register-transition-robust probing model.

5.1 Glitch-Robust Probing Security in the First Round

We aim to demonstrate that the first round is a first-order non-complete TI implementation with two-share inputs and four-share outputs ($2 \rightarrow 4$). These outputs correspond to registers $\hat{\boldsymbol{d}}^0$ in χ, where every four registers $\boldsymbol{d}^0_{0,1,2,3}[x, y, z]$ constitute four shares of a variable $d^0[x, y, z]$. First-order non-completeness implies that each output share $d^0_i[x, y, z]$ is independent of at least one share of each input variable $s^0_k[x', y', z']$.

We utilize Algorithm 1 to generate an index list of related inputs for each output share to assess non-completeness. This algorithm labels each input share $s^0_i[x, y, z]$ with an index (x, y, z, i) and simulates the circuit described in Section 4 to generate an index list for each intermediate variable, containing all related input indices.

Algorithm 1 Glitch-extended probing index list generation.
```
1: for all $x \in \{0, \dots, 4\}, y \in \{0, \dots, 4\}, z \in \{0, \dots, 63\}, i \in \{0, 1\}$ do
2:     list_$s[x, y, z, i] \leftarrow [(x, y, z, i)]$                                        ▷ Input indices
3:     list_$t[x, y, z, i] \leftarrow [(x, y, z, i), (x-1, 0\cdots4, z, i), (x+1, 0\cdots4, z-1, i)]$     ▷ $\theta$
4: list_$s' \leftarrow$ PI(RHO(list_$s$))
5: list_$v \leftarrow$ PI(RHO(list_$t$))
6: list_$d \leftarrow$ CHI_PR(list_$s'$, list_$v$)                                     ▷ Defined in Table 4
7: output list_$d$
```

Theorem 2. *When the input $\hat{\boldsymbol{s}}^0$ satisfies Property 3, the first round is secure under the first-order glitch-robust probing model.*

Proof. The correctness of the circuit is easy to verify. An review of the index lists generated by Algorithm 1 indicates that each list contains at most one share of each input variable. Thus, first-order non-completeness is satisfied in the first round. Consequently, according to Theorem 1, when the input satisfies Property 3, the characteristics of any individual

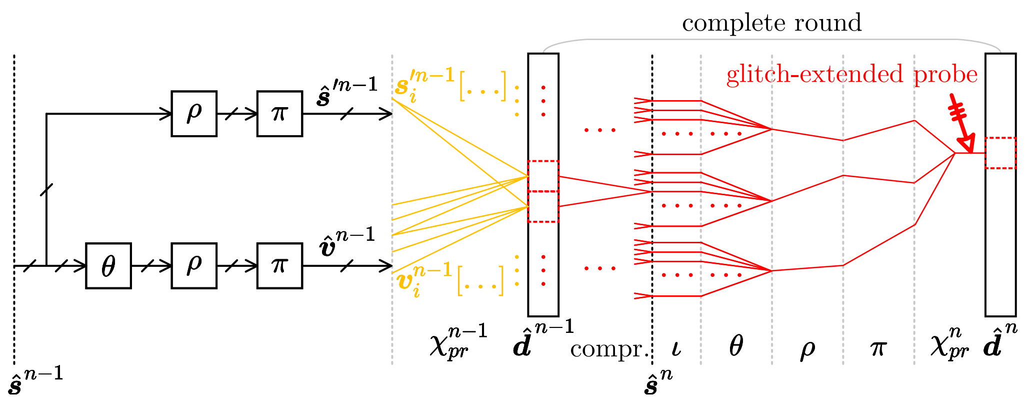

complete round
glitch-extended probe
$\hat{\boldsymbol{s}}'^{n-1}$
$\boldsymbol{s}_i'^{n-1}[\ldots]$
$\hat{\boldsymbol{v}}^{n-1}$
$\boldsymbol{v}_i^{n-1}[\ldots]$
ρ
π
θ
χ_{pr}^{n-1}
$\hat{\boldsymbol{d}}^{n-1}$
compr.
ι
θ
ρ
π
χ_{pr}^{n}
$\hat{\boldsymbol{d}}^{n}$
$\hat{\boldsymbol{s}}^{n}$
$\hat{\boldsymbol{s}}^{n-1}$

Figure 9: Analysis of the Complete Round Under the Glitch-Robust Probing Model: The Red Part Demonstrates the Extending Process of Probes, and the Yellow Part Shows the Related χ_{pr}^{n-1} Operations.

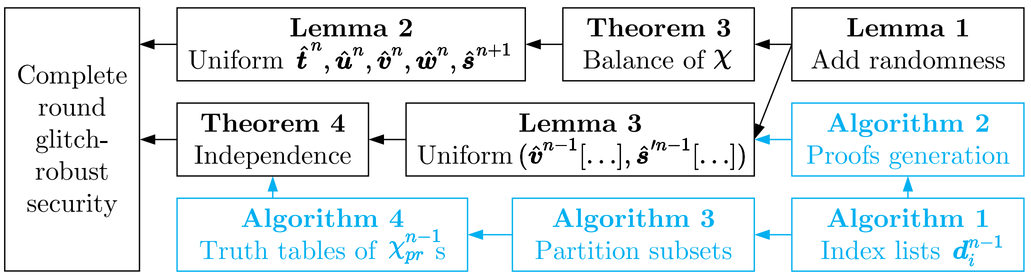

Complete round glitch-robust security
Lemma 2
Uniform $\hat{\boldsymbol{t}}^n, \hat{\boldsymbol{u}}^n, \hat{\boldsymbol{v}}^n, \hat{\boldsymbol{w}}^n, \hat{\boldsymbol{s}}^{n+1}$
Theorem 3
Balance of χ
Lemma 1
Add randomness
Theorem 4
Independence
Lemma 3
Uniform $(\hat{\boldsymbol{v}}^{n-1}[\ldots], \hat{\boldsymbol{s}}'^{n-1}[\ldots])$
Algorithm 2
Proofs generation
Algorithm 4
Truth tables of χ_{pr}^{n-1}s
Algorithm 3
Partition subsets
Algorithm 1
Index lists $\boldsymbol{d}_i^{n-1}$

Figure 10: Relationships among the Theorems, Lemmas, and Algorithms Used in the Security Proof of the Complete Round.

function are independent of sensitive data. As attackers can obtain information about only individual functions in the first-order glitch-robust probing model, they cannot obtain sensitive information. □

5.2 Glitch-Robust Probing Security in the Complete Round

The main difference between the complete round and the first round lies in the compression in χ^{n-1}, which allows the glitch-robust attacker to gain additional access to the values of uncompressed registers $\boldsymbol{d}_i^{n-1}$. Through slightly modified Algorithm 1, it can be verified that first-order non-completeness is satisfied in the complete round. However, the uncompressed inputs $\hat{\mathbf{d}}^{n-1}$ do not satisfy the uniformity conditions, thus rendering Theorem 1 inadequate for proving security in this case.

To demonstrate the security of the complete round, we generate lists of inputs related to each glitch-extended probe using a slightly modified Algorithm 1. As depicted in Figure 9, the values of registers $\boldsymbol{d}_k^{n-1}$ in the list are computed from the corresponding χ_{pr}^{n-1} inputs $\boldsymbol{s}_i'^{n-1}[\ldots]$ and $\boldsymbol{v}_i^{n-1}[\ldots]$. In the glitch-robust probing model, attackers possess knowledge of these register values $\boldsymbol{d}_k^{n-1}$, and their targets are the unshared secret values of the related inputs $\boldsymbol{s}'^{n-1}[\ldots]$ or $\boldsymbol{v}^{n-1}[\ldots]$. We prove in Theorem 4 that these register values are independent of the secret values of the corresponding inputs.

As a first step, we need to establish two lemmas regarding uniformity. First, we demonstrate in Lemma 3 that the χ_{pr}^{n-1} inputs $(\boldsymbol{s}_i'^{n-1}[\ldots], \boldsymbol{v}_i^{n-1}[\ldots])$ corresponding to the registers in each list, as highlighted in yellow in Figure 9, satisfy uniformity, forming the basis for Theorem 4. Second, in Lemma 2, we prove that all five operations $\theta, \rho, \pi, \chi, \iota$ in our design maintain the uniformity of outputs, allowing us to iteratively apply our proofs in each subsequent round. The relationships between these theorems and lemmas are illustrated in Figure 10.

To establish the uniformity of the χ output, we present the following lemma, which is a variant of Lemma 1 from [BDN+13]. The proof of this lemma follows a similar structure to that presented in the original text and is omitted here for conciseness.

Lemma 1 (A Variant of Lemma 1 in [BDN+13])**.** *Let $(\boldsymbol{a}_0, \boldsymbol{a}_1)$ be shares (not necessarily uniform) of an n-bit variable $\boldsymbol{a}$ and $(\boldsymbol{b}_0, \boldsymbol{b}_1)$ be uniform shares of an m-bit variable $\boldsymbol{b}$. Let $(\boldsymbol{c}_0, \boldsymbol{c}_1)$ be uniform n-bit shares statistically independent of $(\boldsymbol{a}_0, \boldsymbol{a}_1)$ and $(\boldsymbol{b}_0, \boldsymbol{b}_1)$. Then, $((\boldsymbol{a}_0 + \boldsymbol{c}_0, \boldsymbol{b}_0), (\boldsymbol{a}_1 + \boldsymbol{c}_1, \boldsymbol{b}_1))$ are uniform $n + m$-bit shares.*

In our χ circuit, as described in Table 4, for $x \in \{2, 3, 4\}$, the compressed output $(w_0^n[x], w_1^n[x])$ satisfies Equation 2, while for $x \in \{0, 1\}$, it satisfies Equation 3, where $j = 64y + z$ and $\boldsymbol{r}_{0,1}$ are two fresh random bits.

$$\begin{aligned} w_0^n[x,j] &\leftarrow \chi'_{x,j}(\boldsymbol{v}_0^n, \boldsymbol{v}_1^n) \triangleq \overline{v_0^n[x+1,j]}v_0^n[x+2,j] \oplus v_0^n[x+1,j]v_1^n[x+2,j] \oplus v_0^n[x,j] \\ w_1^n[x,j] &\leftarrow \chi'_{x,j}(\boldsymbol{v}_1^n, \boldsymbol{v}_0^n) \triangleq \overline{v_1^n[x+1,j]}v_1^n[x+2,j] \oplus v_1^n[x+1,j]v_0^n[x+2,j] \oplus v_1^n[x,j] \end{aligned} \tag{2}$$

$$\begin{aligned} w_0^n[x,j] &\leftarrow \chi'_{x,j}(\boldsymbol{v}_0^n, \boldsymbol{v}_1^n) \oplus g^n[x,j] \\ w_1^n[x,j] &\leftarrow \chi'_{x,j}(\boldsymbol{v}_1^n, \boldsymbol{v}_0^n) \oplus g^n[x,j] \end{aligned} \quad, \text{ where } g^n[x,j] \triangleq \begin{cases} v_0^n[x, j-11] & \text{for } j \neq 0 \\ r_x^n & \text{for } j = 0 \end{cases} \tag{3}$$

Theorem 3. *When the inputs $(\boldsymbol{v}_0^n, \boldsymbol{v}_1^n)$ to Equation 2 for $x \in \{2, 3, 4\}$ and to Equation 3 for $x \in \{0, 1\}$ are shared uniformly, the outputs $(\boldsymbol{w}_0^n, \boldsymbol{w}_1^n)$ are shared uniformly.*

Proof. All the operations involving j below are performed modulo 320, and we omit the modulo notation for convenience. We begin with the row where $j = 11 \times 319 = 309$. For simplicity, we use the notation $\alpha_{x,j}(\boldsymbol{v}_0^n, \boldsymbol{v}_1^n) \triangleq \overline{v_0^n[x+1,j]}v_0^n[x+2,j] \oplus v_0^n[x+1,j]v_1^n[x+2,j]$, which implies that $\chi'_{x,j}(\boldsymbol{v}_0^n, \boldsymbol{v}_1^n) = v_0^n[x,j] \oplus \alpha_{x,j}(\boldsymbol{v}_0^n, \boldsymbol{v}_1^n)$ and that $\alpha_{x,j}(\boldsymbol{v}_0^n, \boldsymbol{v}_1^n)$ is related only to $(\boldsymbol{v}_0^n[x+1 \cdots x+2, j], \boldsymbol{v}_1^n[x+1 \cdots x+2, j])$. Since $(v_0^n[4,j], v_1^n[4,j])$ is uniform and independent of $(\alpha_{4,j}(\boldsymbol{v}_0^n, \boldsymbol{v}_1^n), \alpha_{4,j}(\boldsymbol{v}_1^n, \boldsymbol{v}_0^n))$, it follows from Lemma 1 that $(w_0^n[4,j], w_1^n[4,j])$ is uniform. Because $(v_0^n[3,j], v_1^n[3,j])$ is uniform and independent of $(\alpha_{3,j}(\boldsymbol{v}_0^n, \boldsymbol{v}_1^n), \alpha_{3,j}(\boldsymbol{v}_1^n, \boldsymbol{v}_0^n))$ and $(w_0^n[4,j], w_1^n[4,j])$, we obtain that $(\boldsymbol{w}_0^n[3 \cdots 4, j], \boldsymbol{w}_1^n[3 \cdots 4, j])$ is also uniform. Similarly, by utilizing the uniformity of $(v_0^n[2,j], v_1^n[2,j])$ and its independence from $(\alpha_{2,j}(\boldsymbol{v}_0^n, \boldsymbol{v}_1^n), \alpha_{2,j}(\boldsymbol{v}_1^n, \boldsymbol{v}_0^n))$ and $(\boldsymbol{w}_0^n[3 \cdots 4, j], \boldsymbol{w}_1^n[3 \cdots 4, j])$, we find that $(\boldsymbol{w}_0^n[2 \cdots 4, j], \boldsymbol{w}_1^n[2 \cdots 4, j])$ is uniform. Next, since $(\boldsymbol{v}_0^n[0 \cdots 1, j-11], \boldsymbol{v}_0^n[0 \cdots 1, j-11])$ is uniform and independent of $(\boldsymbol{\chi'}_{0 \cdots 1, j}(\boldsymbol{v}_0^n, \boldsymbol{v}_1^n), \boldsymbol{\chi'}_{0 \cdots 1, j}(\boldsymbol{v}_1^n, \boldsymbol{v}_0^n)$ and $(\boldsymbol{w}_0^n[2 \cdots 4, j], \boldsymbol{w}_1^n[2 \cdots 4, j])$, the entire row $(\boldsymbol{w}_0^n[0 \cdots 4, j], \boldsymbol{w}_1^n[0 \cdots 4, j])$ is uniform.

We can apply this process recursively, with $j = 11i$ starting at $i = 319$ and decreasing to $i = 0$. In each case, the aim is to show that if rows $(\boldsymbol{w}_0^n[0 \cdots 4, 11 \times ((i+1) \cdots 319)]$, $\boldsymbol{w}_1^n[0 \cdots 4, 11 \times ((i+1) \cdots 319)])$ are uniform, then rows $(\boldsymbol{w}_0^n[0 \cdots 4, 11 \times (i \cdots 319)]$, $\boldsymbol{w}_1^n[0 \cdots 4, 11 \times (i \cdots 319)])$ are also uniform.

For each row, the proof process is similar to that of the first row. By applying Lemma 1 recursively, we can sequentially add $(w_0^n[4, 11i]$, $w_1^n[4, 11i])$, $(w_0^n[3, 11i]$, $w_1^n[3, 11i])$, and $(w_0^n[2, 11i]$, $w_1^n[2, 11i])$ to the uniform set, leveraging the independence of $(v_0^n[4, 11i]$, $w_1^n[4, 11i])$, $(v_0^n[3, 11i]$, $w_1^n[3, 11i])$, $(v_0^n[2, 11i]$, $w_1^n[2, 11i])$ and the existing uniform set, respectively. Subsequently, leveraging the uniform and independent $(\boldsymbol{g}^n[0 \cdots 1, 11i]$, $\boldsymbol{g}^n[0 \cdots 1, 11i])$, we can add $(\boldsymbol{w}_0^n[0 \cdots 1, 11i]$, $\boldsymbol{w}_1^n[0 \cdots 1, 11i])$ to the uniform set; thus, rows $(\boldsymbol{w}_0^n[0 \cdots 4, 11 \times (i \cdots 319)]$, $\boldsymbol{w}_1^n[0 \cdots 4, 11 \times (i \cdots 319)])$ are uniform. □

Lemma 2. *For the scheme described in Section 4, when the external input $\hat{\boldsymbol{s}}^0$ satisfies Property 3, the outputs of the θ, ρ, π, χ, and ι steps in each round, i.e., $\hat{\boldsymbol{t}}^n$, $\hat{\boldsymbol{u}}^n$, $\hat{\boldsymbol{v}}^n$, $\hat{\boldsymbol{w}}^n$ and $\hat{\boldsymbol{s}}^{n+1}$ $(n = 0, 1, \dots)$, individually satisfy Property 3.*

Algorithm 2 Generating the proof steps for the uniformity of $(\hat{\boldsymbol{v}}^{n-1}[\dots], \hat{\boldsymbol{s}}'^{n-1}[\dots])$ for each glitch-extended probe.

```
for all $list \in index\_lists$ generated by Algorithm 1 do
    $v\_list, s'\_list \leftarrow$ INPUTS OF CHI($list$)
    ▷ Calculate corresponding indices before $\rho$ and $\pi$ ◁
    $t\_list, s\_list \leftarrow$ INDEX BACKWARDS($v\_list, s'\_list$)
    repeat
        $s\_counts \leftarrow$ the occurrences of each $s$ in the calculation of $t\_list$.
        $s\_counts += $ the occurrences of each $s$ in $s\_list$.
        for all $s, count \in s\_counts$ do
            if $count = 1$ then
                Remove the $t$ using $s$ as input from $t\_list$.
                Record $(s, t)$ to $steps$.
    until $t\_list$ is empty or no $t$ in $t\_list$ can be removed
    if $t\_list$ is empty then
        return $steps$ in reversed order ▷ Pass
    else
        return Fail
```

Proof. The χ step maintains the uniformity of the output, as proven in Theorem 3. The ρ, π, and ι steps are merely shift and constant addition operations and thus do not affect uniformity. The θ step is invertible [BDPA10b], thus satisfying Property 4 [NRR06] and ensuring a uniform output. □

Lemma 3. *When $\hat{\boldsymbol{s}}^{n-1}$ satisfies Property 3, for the χ_{pr}^{n-1} operations before registers $\boldsymbol{d}_i^{n-1}[\dots]$ corresponding to each glitch-extended probe, their input $(\hat{\boldsymbol{v}}^{n-1}[\dots], \hat{\boldsymbol{s}}'^{n-1}[\dots])$ satisfies Property 3.*

Proof. As the entire state of the θ input $\hat{\boldsymbol{s}}^{n-1}$ is uniform, it is evident that subset $\hat{\boldsymbol{s}}'^{n-1}[\dots]$ is also uniform. We refer to the currently known maximum uniform input set as the 'uniform set', which initially consists of $\hat{\boldsymbol{s}}'^{n-1}[\dots]$. The variable $\hat{v}^{n-1}[x, y, z]$ in $\hat{\boldsymbol{v}}^{n-1}[\dots]$ is calculated by XORing 11 θ inputs $\hat{\boldsymbol{s}}^{n-1}[\dots]$. If among these 11 θ inputs there exists one input $\hat{s}^{n-1}[x', y', z']$ independent of the uniform set, Lemma 1 allows us to add it to the uniform set. If this process can be repeated until all variables in $\hat{\boldsymbol{v}}^{n-1}[\dots]$ are added to the uniform set, this proves that $(\hat{\boldsymbol{v}}^{n-1}[\dots], \hat{\boldsymbol{s}}'^{n-1}[\dots])$ satisfies uniformity. Using Algorithm 2, we examine the case for each glitch-extended probe, checking whether the above process can be completed and providing specific steps. The computation results of Algorithm 2 indicate that $(\hat{\boldsymbol{v}}^{n-1}[\dots], \hat{\boldsymbol{s}}'^{n-1}[\dots])$ for each glitch-extended probe satisfies uniformity. □

Table 5: Classified Subsets of Index Lists: Each Subset Comprises Registers with Common χ_{pr}^{n-1} Inputs and May Cause Leakage.

One-Row Subsets		Two-Row Subsets
$\boldsymbol{d}_{0,1}^{n-1}[x], \boldsymbol{d}_{0,1}^{n-1}[x+1]$	$\boldsymbol{d}_{0,1}^{n-1}[0], \boldsymbol{d}_{0,1}^{n-1}[1], \boldsymbol{d}_{0,1}^{n-1}[3]$	$\boldsymbol{d}_{0,1}^{n-1}[0,j], \boldsymbol{d}_{0,1}^{n-1}[1,j+11]$
$\boldsymbol{d}_{2,3}^{n-1}[x], \boldsymbol{d}_{2,3}^{n-1}[x+1]$	$\boldsymbol{d}_{2,3}^{n-1}[0], \boldsymbol{d}_{2,3}^{n-1}[1], \boldsymbol{d}_{2,3}^{n-1}[3]$	$\boldsymbol{d}_{0,1}^{n-1}[0,j], \boldsymbol{d}_{2,3}^{n-1}[1,j+11]$
$\boldsymbol{d}_{0,1}^{n-1}[0], \boldsymbol{d}_{0,1}^{n-1}[3]$	$\boldsymbol{d}_{0,1}^{n-1}[0], \boldsymbol{d}_{0,1}^{n-1}[1], \boldsymbol{d}_{2,3}^{n-1}[3]$	$\boldsymbol{d}_{0,1}^{n-1}[0,j], \boldsymbol{d}_{0,1}^{n-1}[1,j], \boldsymbol{d}_{0,1}^{n-1}[1,j+11]$
$\boldsymbol{d}_{2,3}^{n-1}[0], \boldsymbol{d}_{2,3}^{n-1}[3]$	$\boldsymbol{d}_{2,3}^{n-1}[0], \boldsymbol{d}_{2,3}^{n-1}[1], \boldsymbol{d}_{0,1}^{n-1}[3]$	$\boldsymbol{d}_{0,1}^{n-1}[0,j], \boldsymbol{d}_{0,1}^{n-1}[1,j], \boldsymbol{d}_{2,3}^{n-1}[1,j+11]$
$\boldsymbol{d}_{0,1}^{n-1}[0], \boldsymbol{d}_{2,3}^{n-1}[3]$	$\boldsymbol{d}_{0,1}^{n-1}[0], \boldsymbol{d}_{0,1}^{n-1}[3], \boldsymbol{d}_{0,1}^{n-1}[4]$	$\boldsymbol{d}_{0,1}^{n-1}[0,j], \boldsymbol{d}_{0,1}^{n-1}[4,j], \boldsymbol{d}_{0,1}^{n-1}[0,j+11]$
$\boldsymbol{d}_{2,3}^{n-1}[0], \boldsymbol{d}_{0,1}^{n-1}[3]$	$\boldsymbol{d}_{0,1}^{n-1}[0], \boldsymbol{d}_{2,3}^{n-1}[3], \boldsymbol{d}_{2,3}^{n-1}[4]$	$\boldsymbol{d}_{0,1}^{n-1}[0,j], \boldsymbol{d}_{0,1}^{n-1}[4,j], \boldsymbol{d}_{2,3}^{n-1}[0,j+11]$

Algorithm 3 Partitioning index lists into independent subsets.

```
1: for all $list \in index\_lists$ generated by Algorithm 1 do
2:     repeat
3:         Pop an element $d_i^{n-1}[x, y, z]$ from $list$.
4:         Put $d_i^{n-1}[x, y, z]$ into a new empty $subset$.
5:         Put all $d_k^{n-1}[x', y', z']$ sharing common $\chi_{pr}^{n-1}$ inputs with $d_i^{n-1}[x, y, z]$ into a new
           empty $check\_list$.
6:         while $check\_list$ is not empty do
7:             Pop an element $d$ from $check\_list$.
8:             if $d \in list$ then
9:                 Pop $d$ from $list$ and add $d$ into $subset$.
10:                Add all $d_k^{n-1}[x', y', z']$ sharing common $\chi_{pr}^{n-1}$ inputs with $d$ into
                   $check\_list$.
11:        Store $subset$.
12:    until $list$ is empty
13: Classify all $subset$s.
```

Below, we demonstrate the independence between the register values $\boldsymbol{d}_i^{n-1}[\dots]$ known to the attacker and the target secret values $(\boldsymbol{s}'^{n-1}, \boldsymbol{v}^{n-1})$. As the number of these variables is too large to directly check the independence, we partition these registers into several independent subsets and check the independence separately. We consider involving at least one same input variable during the χ_{pr}^{n-1} operation as an equivalence relation, partitioning each index list into multiple subsets by Algorithm 3. Thus, registers in different subsets have completely independent inputs. Table 5 summarizes cases in which the subset size is greater than 1.

Algorithm 4 Checking the independence of the register values $\boldsymbol{d}_i^{n-1(k)}$ and secret values $\boldsymbol{v}^{n-1(k)}$ and $\boldsymbol{s}'^{n-1(k)}$ for each subset.

```
1: for all $subset^{(k)}$ ∈subsets generated by Algorithm 3 do
2:     $\boldsymbol{c}^{(k)} \leftarrow \{\boldsymbol{v}^{(k)}, \boldsymbol{s}'^{(k)}\}$.          ▷ The inputs are uniformly denoted as c
3:     $n_c^{(k)}, n_d^{(k)} \leftarrow$ the bit widths of $\boldsymbol{c}^{(k)}, \boldsymbol{d}_i^{(k)}$.
4:     $f_d^{(k)} \leftarrow$ the function maps from $\hat{\boldsymbol{c}}^{(k)}$ to $\boldsymbol{d}_i^{(k)}$.
5:     procedure TRUTH TABLE GENERATION($n_c^{(k)}, n_d^{(k)}, f_d^{(k)}$)
6:         for all possible input value $\hat{\boldsymbol{C}}^{(k)} \in \{0,1\}^{2n_c^{(k)}}$ do
7:             Calculate corresponding register values $\boldsymbol{D}_i^{(k)} = f_d^{(k)}(\hat{\boldsymbol{C}}^{(k)})$.
8:             Calculate corresponding unshared secret values $\boldsymbol{C}^{(k)} = \boldsymbol{C}_0^{(k)} \oplus \boldsymbol{C}_1^{(k)}$.
9:     for all possible register value $\boldsymbol{D}_i^{(k)} \in \{0,1\}^{n_d^{(k)}}$ do
10:        $n_D^{(k)} \leftarrow$ the number of rows where $\boldsymbol{d}_i^{(k)} = \boldsymbol{D}_i^{(k)}$ in $truth\_table$.
11:        for all possible secret value $\boldsymbol{C}^{(k)} \in \{0,1\}^{n_c^{(k)}}$ do
12:            $n_{D,C}^{(k)} \leftarrow$ the number of rows where $\boldsymbol{d}_i^{(k)} = \boldsymbol{D}_i^{(k)}$ and $\boldsymbol{c}^{(k)} = \boldsymbol{C}^{(k)}$ in
               $truth\_table$.
13:            if $n_{D,C}^{(k)} \neq n_D^{(k)}/2^{n_c^{(k)}}$ then
14:                return Fail
15: return Pass
```

Theorem 4. *When the θ inputs in the previous round $\hat{\boldsymbol{s}}^{n-1}$ satisfy Property 3, the complete round is secure under the first-order glitch-robust probing model.*

Proof. To establish first-order glitch-robust security, our discussion focuses on the circuit preceding a single glitch-extended probe, with these arguments applicable to any such probe. Initially, we demonstrate the independence of the register values ($\boldsymbol{d}_i^{n-1}$) and secrets ($\boldsymbol{v}^{n-1}$ and $\boldsymbol{s}'^{n-1}$) for each subset. Subsequently, we establish the independence of the register values and secrets for the entire circuit preceding a probe.

Considering the uniformity of all inputs ($\hat{\boldsymbol{v}}^{n-1}[\dots], \hat{\boldsymbol{s}}'^{n-1}[\dots]$) corresponding to a probe, as confirmed by Lemma 3, we represent them uniformly as $\hat{\boldsymbol{c}}$, where $\hat{\boldsymbol{c}} = \{\hat{\boldsymbol{v}}^{n-1}[\dots], \hat{\boldsymbol{s}}'^{n-1}[\dots]\}$. Within the k-th subset, variables are denoted with the superscript (k). The bit width of inputs $\hat{\boldsymbol{c}}^{(k)}$ is represented as $2n_c^{(k)}$, with $n_c^{(k)}$ indicating the bit width of unshared inputs $\boldsymbol{c}^{(k)}$. Additionally, $n_d^{(k)}$ denotes the bit width of the register values $\boldsymbol{d}_i^{(k)}$.

For each subset, we analyze the truth table generated by Algorithm 4. According to Property 3, the probability associated with each row in the truth table $\Pr(\hat{\boldsymbol{c}}^{(k)} = \hat{\boldsymbol{C}}^{(k)})$, abbreviated as $\Pr(\hat{\boldsymbol{C}}^{(k)})$, can be expressed as:

$$\Pr(\hat{\boldsymbol{C}}^{(k)}) = \frac{1}{2^{n_c^{(k)}}} \Pr(\boldsymbol{C}^{(k)}).$$

According to Algorithm 4, within the $n_D^{(k)}$ rows in the truth table where $\boldsymbol{d}_i^{(k)} = \boldsymbol{D}_i^{(k)}$, the number of rows with secret value $\boldsymbol{c}^{(k)}$ equal to each possible $\boldsymbol{C}^{(k)}$ is $n_D^{(k)}/2^{n_c^{(k)}}$. This implies that:

$$\Pr(\boldsymbol{D}_i^{(k)}, \boldsymbol{C}^{(k)}) = \frac{n_D^{(k)}}{2^{n_c^{(k)}}} \cdot \frac{1}{2^{n_c^{(k)}}} \Pr(\boldsymbol{C}^{(k)}) = \frac{n_D^{(k)}}{2^{2n_c^{(k)}}} \Pr(\boldsymbol{C}^{(k)})$$

$$\Pr(\boldsymbol{D}_i^{(k)}) = \sum_{\boldsymbol{C}^{(k)}} \Pr(\boldsymbol{D}_i^{(k)}, \boldsymbol{C}^{(k)}) = \sum_{\boldsymbol{C}^{(k)}} \frac{n_D^{(k)}}{2^{2n_c^{(k)}}} \Pr(\boldsymbol{C}^{(k)}) = \frac{n_D^{(k)}}{2^{2n_c^{(k)}}}.$$

This proves that for each subset, the register values are independent of the secret values, i.e., $\Pr(\boldsymbol{D}_i^{(k)}, \boldsymbol{C}^{(k)}) = \Pr(\boldsymbol{D}_i^{(k)}) \Pr(\boldsymbol{C}^{(k)})$. Next, we merge the truth tables of all the subsets into a single large truth table, where each row corresponds to each possible input value $\hat{\boldsymbol{C}} = (\hat{\boldsymbol{C}}^{(1)}, \dots, \hat{\boldsymbol{C}}^{(m)})$ and columns correspond to register values $\boldsymbol{d}_i = (\boldsymbol{d}_i^{(1)}, \dots, \boldsymbol{d}_i^{(m)})$ and secret values $\boldsymbol{c} = (\boldsymbol{c}^{(1)}, \dots, \boldsymbol{c}^{(m)})$. According to Property 3, the probability associated with each row in the large truth table is:

$$\Pr(\hat{\boldsymbol{C}}) = \frac{1}{2^{n_c}} \Pr(\boldsymbol{C}),$$

where $n_c = \sum_{k=1}^{m} n_c^{(k)}$. Due to the independence among the calculations for distinct subsets, a large truth table is formed by taking the direct product of the truth tables of these subsets. Therefore, the number of rows in the large table where $\boldsymbol{d}_i = \boldsymbol{D}_i$ is equal to $\prod_{k=1}^{m} n_D^{(k)}$. Similarly, the number of rows in the large table where $\boldsymbol{d}_i = \boldsymbol{D}_i$ and $\boldsymbol{c} = \boldsymbol{C}$ is equal to $\prod_{k=1}^{m} \frac{n_D^{(k)}}{2^{n_c^{(k)}}}$. Thus, we obtain:

$$\Pr(\boldsymbol{D}_i, \boldsymbol{C}) = \prod_{k=1}^{m} \frac{n_D^{(k)}}{2^{n_c^{(k)}}} \cdot \frac{1}{2^{n_c}} \Pr(\boldsymbol{C}) = \frac{\prod_{k=1}^{m} n_D^{(k)}}{2^{n_c}} \cdot \frac{1}{2^{n_c}} \Pr(\boldsymbol{C}) = \frac{\prod_{k=1}^{m} n_D^{(k)}}{2^{2n_c}} \Pr(\boldsymbol{C})$$

$$\Pr(\boldsymbol{D}_i) = \sum_{\boldsymbol{C}} \Pr(\boldsymbol{D}_i, \boldsymbol{C}) = \frac{\prod_{k=1}^{m} n_D^{(k)}}{2^{2n_c}}.$$

Therefore, $\Pr(\boldsymbol{D}_i, \boldsymbol{C}) = \Pr(\boldsymbol{D}_i) \Pr(\boldsymbol{C})$, indicating that the register values and secret values corresponding to a single glitch-extended probe are independent. □

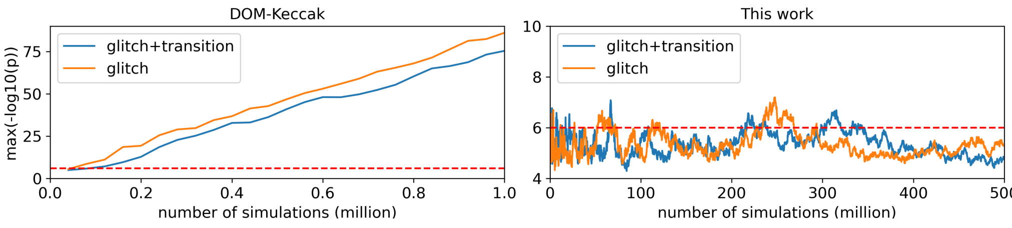


Figure 11: PROLEAD Results for DOM-Keccak and Our Design.

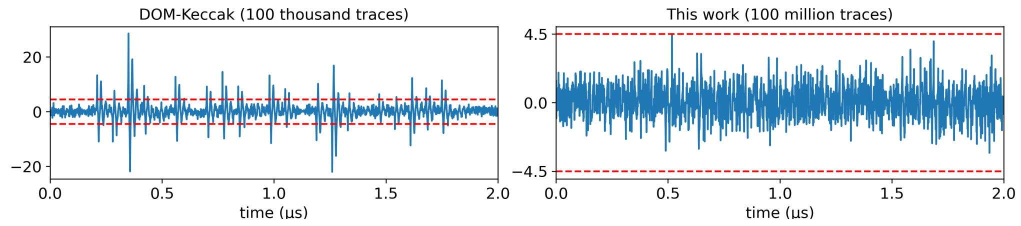


Figure 12: First-Order TVLA Results for DOM-Keccak and Our Design.

5.3 Glitch+Register-Transition-Robust Probing Security

In this section, we additionally consider the potential leakage caused by register transitions, namely, the combined glitch+register-transition-robust probing security. In this model, the attacker can additionally gain knowledge of the consecutive two-cycle values of a register. The security proof under this model can be easily integrated into the proof process in Section 5.2 as follows. For a glitch-extended probe placed at $d_i^n[x, y, z]$, Algorithm 1 generates an index list of corresponding inputs $\boldsymbol{d}_k^{n-1}[\dots]$. For the combined model, we append $d_i^{n-1}[x, y, z]$ to the list, which represents the value in the previous cycle of the probed register. Subsequently, we executed the remaining security proof components outlined in Section 5.2, which establish the first-order combined glitch+register-transition-robust probing security.

To the best of our knowledge, existing round-based masked Keccak implementations only guarantee glitch-robust probing security. Although no transition-related leakage has been found in these implementations yet, they did not provide formal proofs of transition-robust security. Our design additionally ensures first-order combined security with register transitions. Meeting the more conservative glitch+wire-transition-robust probing security is highly challenging for low-latency round-based implementations. Furthermore, the impact of this combination on security is believed to be negligible under sufficient noise levels [FGP+18]. To assess the potential leakage in the glitch+wire-transition-robust model, we use the simulation-based verification tool PROLEAD to evaluate our design in Section 6.

6 Leakage Evaluation

In this section, we evaluate the security of our design with experimental leakage assessments, including the PROLEAD verification tool [MM22], Test Vector Leakage Assessment (TVLA) experiments [BCD+13], and the VerMI verification tool [ANR18], serving as complementary evidence to support our theoretical security proofs.

PROLEAD. PROLEAD is a simulation-based leakage detection tool that evaluates the security of masking designs by directly analyzing the distribution of intermediate variables. PROLEAD can be used to assess both glitch-robust [FGP+18] and glitch+wire-transition-

Table 6: KECCAK-f[1600] Round-Based First-Order Masking Implementations' Performance Results and Comparison Using NanGate 45 nm Library.

Designs	Area (kGE)						Rand.	Clock	Freq.
	θ	χ	State	Other	Total	Ratio	(bits)	cycles	(MHz)
Plain[BDN+13]	6.4	5.6	9.0	7.1	28.1	1.0×	-	24	690
This work	**13.2**	**26.1**	**36.3**	**7.9**	**83.5**	**3.0×**	**2**	**24**	**763**
[BDN+13]-3sh	19.2	40.6	27.2	29.6	116.6	4.1×	4	25	592
[BDN+13]-4sh	25.6	48.7	36.3	28.8	139.4	5.0×	0	24	588
[ZSS+21][a]	50.3	21.9	36.3	6.5	115.0	4.1×	1600	24	877
[GSM17a][b,c]	15.0	38.4	32.2	-	85.7	3.0×	0	48	891
[SM21b][d]	-	-	-	-	129.3	4.2×	0	72	775
[ABP+18]-6sh[e]	11.3	44.6	8.0	6.2	70.1	25×	0	9	437

[a] Results synthesized using the open-source codes of original works.
[b] Results synthesized with library UMC 90 nm.
[c] Implementation lacking uniformity and thus theoretically insecure.
[d] Results synthesized with library UMC 130 nm.
[e] Implementation of KECCAK-f[200].

robust [CS21] probing models. The assessment results are expressed as p values, with the authors suggesting a typical threshold of 10^{-5}. However, due to the large number of probing sets in our case (2.26×10^5), there is a significant probability (approximately 90%) of encountering false positives [MM22]. Therefore, we adjusted the threshold to 10^{-6}, corresponding to a false-positive probability of approximately 20%. Additionally, the decisive factor for leakage does not exceed the threshold but rather is the continuous increase in the p value with an increasing number of simulations [Mül23].

As an illustration, we conducted 1 million simulations on DOM-KECCAK using PROLEAD's `compact` mode, and the relationship between the minimum p value and the number of simulations is shown in Figure 11. In the presence of leakage, the p value quickly increases with the number of simulations. We conducted 500 million simulations for our implementation under both glitch-robust and glitch+wire-transition-robust probing models using PROLEAD's `compact` mode, and the results are shown in Figure 11. Notably, our p values do not increase with the number of simulations for either the glitch-robust or glitch+wire-transition-robust probing models. Furthermore, all our final p values are below the threshold of 10^{-6}. This indicates that even when considering the glitch+wire-transition-robust probing model and testing with noise-free PROLEAD, there is no observable leakage in our implementation.

TVLA. We conducted the fixed-versus-random t-test on the power traces of a Kintex-7 FPGA-based evaluation platform (SAKURA-X) following the strategies and procedures explained in [SM15]. Using a WaveRunner 8404M oscilloscope, we measured the AC-amplified voltage drop over a 10 mΩ shunt resistor placed on the VDD path of the target FPGA. The KECCAK design was clocked at a frequency of 14.3 MHz, and the power traces were sampled at 1 GS/s. To reduce noise, all fresh random numbers used in the operations were generated in advance by a TRIVIUM-based pseudorandom number generator [De 06].

We evaluated our design using 100 million traces and simultaneously tested the one-cycle-per-round DOM-KECCAK design in the same experimental setup for comparison. The test results are presented in Figure 12, where the red dashed line represents the threshold for leakage detection. Our design exhibited no leakage in the test, with 100 million traces. In contrast, significant leakage was detected for the DOM-KECCAK design, with only 100 thousand traces.

VerMI. VerMI is a verification tool designed for masked implementations to detect whether they satisfy non-completeness and uniformity. Since our design is too large for

VerMI to check uniformity, we only assessed non-completeness. Our design successfully passed the VerMI non-completeness test. Notably, this validation cannot replace all manual proofs. The test results demonstrate that there is no leakage in the first round, and any glitch-extended probe in subsequent rounds will not extend to all shares of a variable.

7 Results and Comparison

This section presents the ASIC implementation results of the proposed design and a comparison with related works. We used the Synopsys Design Compiler with the NanGate 45 nm Open Cell Library to synthesize our design. The options `-exact_map` and `-no_autoungroup` were set during the synthesis to maintain the hierarchy and prevent any optimization that could affect the security. The implementation results and state-of-the-art round-based first-order masking implementations of Keccak are presented in Table 6. To highlight the area cost of the masking, we also present the plain implementation without countermeasures proposed by [BDN+13], along with the relative area ratio compared to the area of the plain implementation for each design. The results reported in [ZSS+21] are only for Keccak-$f[200]$, and we synthesized the $f[1600]$ version based on the authors' open-source code [ZSM21] for comparison. The first-order design of [GSM17a] does not satisfy uniformity conditions and thus is not theoretically first-order secure. The results reported in [ABP+18] are only for Keccak-$f[200]$, and the source code is not available for resynthesis.

Bilgin et al. [BDN+13] proposed first-order TI designs of Keccak, including a 3-share version with 4-bit randomness requirements per cycle and a 4-share version without randomness requirements. Our design uses only two shares instead of at least three in TIs, which enables us to achieve lower area costs for θ, χ, and MUXs. Compared to their implementations, ours reduces area requirements by at least 28%, translating to just three times the area of the plain implementation, rather than over four times for theirs.

Zarei et al. [ZSS+21] proposed DOM-based masking implementations of Keccak for arbitrary orders. To prevent leakage caused by glitches, they deferred the compression step of DOM until the completion of the θ operation in the next round, resulting in a fourfold increase in the θ area and an eightfold increase in the MUX area. To achieve uniformity, their first-order implementation uses 1600 bits of random numbers per cycle. In contrast, our leakage countermeasure adds only a small number of XOR gates compared to the original DOM implementation, reducing the number of θ circuits by half and requiring many fewer MUXs, resulting in a 27% reduction in area consumption. Furthermore, our carefully designed COTG-like method for achieving uniformity reduces random number consumption by 1598 bits per cycle.

Gross et al. [GSM17a] proposed DOM-based implementations of Keccak. To avoid leakage caused by glitches, they require two cycles to execute one round. Additionally, they attempted to remove the randomness requirements in the first-order implementation but failed to achieve uniformity. In contrast, we prevent glitch-induced leakage through a new structure rather than adding an extra stage of registers, resulting in a halved delay compared to that in their design. Our design meets the theoretical uniformity requirements, thereby achieving provable security. By using 1600 bits of fresh randomness per cycle, their design can also be adapted to address uniformity. However, even with a high-throughput stream cipher-based PRNG such as Trivium, this results in at least 48 kGE in area with the 65 nm library [CMM+23].

Shahmirzadi et al. [SM21b] designed a 2-share masked χ function that achieves uniformity without fresh randomness requirements. However, the new χ function is much more complex than the original, introducing new data dependencies and a larger area cost. To avoid leakage caused by glitches, they require three cycles to execute one round. In contrast, we achieve uniformity through a COTG-like approach, maintaining the simplicity of the χ function, resulting in a smaller area. Moreover, our new structure for preventing glitch-induced leakage reduces our delay to only one-third of theirs.

Arribas et al. [ABP+18] proposed first-order masked Keccak implementations where two rounds are executed in one cycle. These implementations are based on TIs with more than five shares, achieving half the clock cycles of the plain implementation but at the cost of more than 25 times the area. In comparison, we achieve the same clock cycles as the plain implementation with only three times the area.

8 Conclusions and Discussions

In this work, we conducted a detailed analysis of potential leaks in two-share first-order masked SHA-3, paving the way for further reducing the area cost of masking SHA-3. Future research could explore additional optimization techniques to achieve more efficient implementations. We present a compact and low-latency first-order masked SHA-3 implementation, which significantly reduces the area cost from more than four times that of unprotected implementations to only three times that of unprotected implementations. A complete theoretical proof of security under the glitch+register-transition-robust probing model and the results of experimental leakage detection are reported for this implementation. The RTL design codes, leakage analysis programs, and security proof algorithms used are available at https://github.com/zck15/CLLFO-SHA-3. The presented leakage analysis for first-order masked SHA-3 can guide the design of additional masking protections for SHA-3. The proposed implementation can be employed in crafting compact and efficient implementations of higher-level cryptographic schemes, such as post-quantum cryptography algorithms.

When incorporating our design as a constituent within larger masking circuits, it is important to note that our design does not satisfy Non-Interference (NI) [BBD+16], Strong Non-Interference (SNI) [BBD+16], or Probe-Isolating Non-Interference (PINI) [CS20]. Nevertheless, due to first-order security allowing only one probe at a time, our design can be trivially integrated to construct a large first-order secure circuit with the following considerations. Prior to feeding inputs into our design or utilizing its outputs, signals should be stored in registers to prevent glitch-robust probes from extending to unexpected locations. Uniformity conditions should be satisfied for the entire 2×1600-bit input state. The outputs of a single squeeze operation ensure uniformity, as corroborated by Lemma 2. Due to constraints on the total entropy of the inputs, the aggregate outputs of multiple squeezes cannot achieve uniformity. However, since the outputs of multiple squeezes are spaced by 24 rounds of diffusion, we believe that exploiting the relationships between these outputs would be challenging.

Our achievement in significant area reduction is based on optimizations tailored for the round-based Keccak-f[1600] version. Therefore, our circuit structure cannot be directly applied to other Keccak-$f[b]$ versions with smaller b values or higher orders. However, our leakage analysis can provide insights for protecting other versions of Keccak. The technique of utilizing θ inputs as random numbers can be utilized by other masking designs. Moreover, our security proof methodology and scripts can serve as references for other first-order masking designs.

Acknowledgments

The authors would like to express their gratitude to the anonymous reviewers for their valuable comments. Special thanks go to Victor Arribas, Svetla Nikova, and Vincent Rijmen for their invaluable assistance in utilizing VerMI to evaluate our design. This work is supported in part by the National Key R&D Program of China (Grant No.

2023YFB4403500), and in part by the National Natural Science Foundation of China (Grant No. 62274102), and in part by the National Key R&D Program of China (Grant No.2021YFB2701201).

References

[ABC$^+$23] Melissa Azouaoui, Olivier Bronchain, Gaëtan Cassiers, Clément Hoffmann, Yulia Kuzovkova, Joost Renes, Tobias Schneider, Markus Schönauer, François-Xavier Standaert, and Christine van Vredendaal. Protecting dilithium against leakage revisited sensitivity analysis and improved implementations. *IACR TCHES*, 2023(4):58–79, 2023.

[ABCG20] Erdem Alkim, Yusuf Alper Bilgin, Murat Cenk, and François Gérard. Cortex-M4 optimizations for {R,M}LWE schemes. *IACR TCHES*, 2020(3):336–357, 2020. https://tches.iacr.org/index.php/TCHES/article/view/8593.

[ABP$^+$18] Victor Arribas, Begül Bilgin, George Petrides, Svetla Nikova, and Vincent Rijmen. Rhythmic Keccak: SCA security and low latency in HW. *IACR TCHES*, 2018(1):269–290, 2018. https://tches.iacr.org/index.php/TCHES/article/view/840.

[AMI$^+$23] Aikata Aikata, Ahmet Can Mert, Malik Imran, Samuel Pagliarini, and Sujoy Sinha Roy. Kali: A crystal for post-quantum security using kyber and dilithium. *IEEE Trans. Circuits Syst. I Regul. Pap.*, 70(2):747–758, 2023.

[ANR18] Victor Arribas, Svetla Nikova, and Vincent Rijmen. Vermi: Verification tool for masked implementations. In *25th IEEE International Conference on Electronics, Circuits and Systems, ICECS 2018, Bordeaux, France, December 9-12, 2018*, pages 381–384. IEEE, 2018.

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

[BCD$^+$13] George Becker, Jennifer Cooper, Elke DeMulder, Gilbert Goodwill, Joshua Jaffe, Gary Kenworthy, Timofei Kouzminov, Andrew Leiserson, Matthew Marson, Pankaj Rohatgi, and Sami Saab. Test vector leakage assessment (tvla) methodology in practice. In *International Cryptographic Module Conference*, volume 20, 2013.

[BDF$^+$17] Gilles Barthe, François Dupressoir, Sebastian Faust, Benjamin Grégoire, François-Xavier Standaert, and Pierre-Yves Strub. Parallel implementations of masking schemes and the bounded moment leakage model. In Jean-Sébastien Coron and Jesper Buus Nielsen, editors, *EUROCRYPT 2017, Part I*, volume 10210 of *LNCS*, pages 535–566. Springer, Heidelberg, April / May 2017.

[BDN$^+$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 Aurélien Francillon and Pankaj Rohatgi, editors, *Smart Card Research and Advanced Applications - 12th International Conference, CARDIS 2013, Berlin, Germany, November 27-29, 2013. Revised Selected Papers*, volume 8419 of *Lecture Notes in Computer Science*, pages 187–199. Springer, 2013.

[BDPA10a] Guido Bertoni, Joan Daemen, Michaël Peeters, and Gilles Van Assche. Building power analysis resistant implementations of keccak. In *Second SHA-3 Candidate Conference*, 2010.

[BDPA10b] Guido Bertoni, Joan Daemen, Michaël Peeters, and Gilles Van Assche. Keccak sponge function family main document. Technical report, June 2010. available at https://keccak.team/archives.html.

[BGR$^+$21] Joppe W. Bos, Marc Gourjon, Joost Renes, Tobias Schneider, and Christine van Vredendaal. Masking kyber: First- and higher-order implementations. *IACR TCHES*, 2021(4):173–214, 2021. https://tches.iacr.org/index.php/TCHES/article/view/9064.

[BNN$^+$12] Begül Bilgin, Svetla Nikova, Ventzislav Nikov, Vincent Rijmen, and Georg Stütz. Threshold implementations of all 3×3 and 4×4 S-boxes. In Emmanuel Prouff and Patrick Schaumont, editors, *CHES 2012*, volume 7428 of *LNCS*, pages 76–91. Springer, Heidelberg, September 2012.

[CJRR99] Suresh Chari, Charanjit S. Jutla, Josyula R. Rao, and Pankaj Rohatgi. Towards sound approaches to counteract power-analysis attacks. In Michael J. Wiener, editor, *CRYPTO'99*, volume 1666 of *LNCS*, pages 398–412. Springer, Heidelberg, August 1999.

[CMM$^+$23] Gaëtan Cassiers, Loïc Masure, Charles Momin, Thorben Moos, Amir Moradi, and François-Xavier Standaert. Randomness generation for secure hardware masking - unrolled trivium to the rescue. Cryptology ePrint Archive, Report 2023/1134, 2023. https://eprint.iacr.org/2023/1134.

[CS20] Gaëtan Cassiers and François-Xavier Standaert. Trivially and efficiently composing masked gadgets with probe isolating non-interference. *IEEE Trans. Inf. Forensics Secur.*, 15:2542–2555, 2020.

[CS21] Gaëtan Cassiers and François-Xavier Standaert. Provably secure hardware masking in the transition- and glitch-robust probing model: Better safe than sorry. *IACR TCHES*, 2021(2):136–158, 2021. https://tches.iacr.org/index.php/TCHES/article/view/8790.

[Dae17] Joan Daemen. Changing of the guards: A simple and efficient method for achieving uniformity in threshold sharing. In Wieland Fischer and Naofumi Homma, editors, *CHES 2017*, volume 10529 of *LNCS*, pages 137–153. Springer, Heidelberg, September 2017.

[DDF14] Alexandre Duc, Stefan Dziembowski, and Sebastian Faust. Unifying leakage models: From probing attacks to noisy leakage. In Phong Q. Nguyen and Elisabeth Oswald, editors, *EUROCRYPT 2014*, volume 8441 of *LNCS*, pages 423–440. Springer, Heidelberg, May 2014.

[De 06] Christophe De Cannière. Trivium: A stream cipher construction inspired by block cipher design principles. In Sokratis K. Katsikas, Javier Lopez, Michael Backes, Stefanos Gritzalis, and Bart Preneel, editors, *ISC 2006*, volume 4176 of *LNCS*, pages 171–186. Springer, Heidelberg, August / September 2006.

[FGP$^+$18] Sebastian Faust, Vincent Grosso, Santos Merino Del Pozo, Clara Paglialonga, and François-Xavier Standaert. Composable masking schemes in the presence of physical defaults & the robust probing model. *IACR TCHES*, 2018(3):89–120, 2018. https://tches.iacr.org/index.php/TCHES/article/view/7270.

[GKS21] Denisa O. C. Greconici, Matthias J. Kannwischer, and Amber Sprenkels. Compact dilithium implementations on cortex-M3 and cortex-M4. *IACR TCHES*, 2021(1):1–24, 2021. https://tches.iacr.org/index.php/TCHES/article/view/8725.

[GMK16] Hannes Groß, Stefan Mangard, and Thomas Korak. Domain-oriented masking: Compact masked hardware implementations with arbitrary protection order. In Begül Bilgin, Svetla Nikova, and Vincent Rijmen, editors, *Proceedings of the ACM Workshop on Theory of Implementation Security, TIS@CCS 2016 Vienna, Austria, October, 2016*, page 3. ACM, 2016.

[GS18] Vincent Grosso and François-Xavier Standaert. Masking proofs are tight and how to exploit it in security evaluations. In Jesper Buus Nielsen and Vincent Rijmen, editors, *EUROCRYPT 2018, Part II*, volume 10821 of *LNCS*, pages 385–412. Springer, Heidelberg, April / May 2018.

[GSM17a] Hannes Gross, David Schaffenrath, and Stefan Mangard. Higher-order side-channel protected implementations of Keccak. Cryptology ePrint Archive, Report 2017/395, 2017. https://eprint.iacr.org/2017/395.

[GSM17b] Hannes Groß, David Schaffenrath, and Stefan Mangard. Higher-order side-channel protected implementations of KECCAK. In Hana Kubátová, Martin Novotný, and Amund Skavhaug, editors, *Euromicro Conference on Digital System Design, DSD 2017, Vienna, Austria, August 30 - Sept. 1, 2017*, pages 205–212. IEEE Computer Society, 2017.

[HBD$^+$22] Andreas Hülsing, Daniel J. Bernstein, Christoph Dobraunig, Maria Eichlseder, Scott Fluhrer, Stefan-Lukas Gazdag, Panos Kampanakis, Stefan Kölbl, Tanja Lange, Martin M. Lauridsen, Florian Mendel, Ruben Niederhagen, Christian Rechberger, Joost Rijneveld, Peter Schwabe, Jean-Philippe Aumasson, Bas Westerbaan, and Ward Beullens. SPHINCS$^+$. Technical report, National Institute of Standards and Technology, 2022. available at https://csrc.nist.gov/Projects/post-quantum-cryptography/selected-algorithms-2022.

[ISW03] Yuval Ishai, Amit Sahai, and David Wagner. Private circuits: Securing hardware against probing attacks. In Dan Boneh, editor, *CRYPTO 2003*, volume 2729 of *LNCS*, pages 463–481. Springer, Heidelberg, August 2003.

[KjCP16] John Kelsey, Shu jen Chang, and Ray Perlner. Nist special publication 800-185: SHA-3 derived functions: cshake, kmac, tuplehash and parallelhash. Technical report, National Institute of Standards and Technology, 2016. available at https://csrc.nist.gov/pubs/sp/800/185/final.

[KM22] David Knichel and Amir Moradi. Composable gadgets with reused fresh masks first-order probing-secure hardware circuits with only 6 fresh masks. *IACR TCHES*, 2022(3):114–140, 2022.

[KPP20] Matthias J. Kannwischer, Peter Pessl, and Robert Primas. Single-trace attacks on Keccak. *IACR TCHES*, 2020(3):243–268, 2020. https://tches.iacr.org/index.php/TCHES/article/view/8590.

[LDK$^+$22] Vadim Lyubashevsky, Léo Ducas, Eike Kiltz, Tancrède Lepoint, Peter Schwabe, Gregor Seiler, Damien Stehlé, and Shi Bai. CRYSTALS-DILITHIUM. Technical report, National Institute of Standards and Technology, 2022. available at https://csrc.nist.gov/Projects/post-quantum-cryptography/selected-algorithms-2022.

[LFF+14] Pei Luo, Yunsi Fei, Xin Fang, A. Adam Ding, Miriam Leeser, and David R. Kaeli. Power analysis attack on hardware implementation of mac-keccak on fpgas. In *2014 International Conference on ReConFigurable Computing and FPGAs, ReConFig14, Cancun, Mexico, December 8-10, 2014*, pages 1–7. IEEE, 2014.

[LFF+15] Pei Luo, Yunsi Fei, Xin Fang, A. Adam Ding, David R. Kaeli, and Miriam Leeser. Side-channel analysis of mac-keccak hardware implementations. In Ruby B. Lee, Weidong Shi, and Jakub Szefer, editors, *Proceedings of the Fourth Workshop on Hardware and Architectural Support for Security and Privacy, HASP@ISCA 2015, Portland, OR, USA, June 14, 2015*, pages 1:1–1:8. ACM, 2015.

[MM22] Nicolai Müller and Amir Moradi. PROLEAD A probing-based hardware leakage detection tool. *IACR TCHES*, 2022(4):311–348, 2022.

[MPG05] Stefan Mangard, Thomas Popp, and Berndt M. Gammel. Side-channel leakage of masked CMOS gates. In Alfred Menezes, editor, *CT-RSA 2005*, volume 3376 of *LNCS*, pages 351–365. Springer, Heidelberg, February 2005.

[MPO05] Stefan Mangard, Norbert Pramstaller, and Elisabeth Oswald. Successfully attacking masked AES hardware implementations. In Josyula R. Rao and Berk Sunar, editors, *CHES 2005*, volume 3659 of *LNCS*, pages 157–171. Springer, Heidelberg, August / September 2005.

[Mül23] Nicolai Müller. Prolead - wiki - results. https://github.com/ChairImpSec/PROLEAD/wiki/Results, 2023. Accessed: December 16, 2023.

[NIS15] NIST. Fips pub 202: SHA-3 Standard: Permutation-Based Hash and Extendable-Output Functions. Technical report, U.S. Department of Commerce, 2015. available at https://csrc.nist.gov/pubs/fips/202/final.

[NRR06] Svetla Nikova, Christian Rechberger, and Vincent Rijmen. Threshold implementations against side-channel attacks and glitches. In Peng Ning, Sihan Qing, and Ninghui Li, editors, *ICICS 06*, volume 4307 of *LNCS*, pages 529–545. Springer, Heidelberg, December 2006.

[PMSN23] Srinidhi Hari Prasad, Florian Mendel, Martin Schläffer, and Rishub Nagpal. Efficient low-latency masking of ascon without fresh randomness. Cryptology ePrint Archive, Report 2023/1914, 2023. https://eprint.iacr.org/2023/1914.

[PR13] Emmanuel Prouff and Matthieu Rivain. Masking against side-channel attacks: A formal security proof. In Thomas Johansson and Phong Q. Nguyen, editors, *EUROCRYPT 2013*, volume 7881 of *LNCS*, pages 142–159. Springer, Heidelberg, May 2013.

[PYY+23] Shuohang Peng, Bohan Yang, Shuying Yin, Hang Zhao, Cankun Zhao, Shaojun Wei, and Leibo Liu. A low-randomness first-order masked xoodyak. In *IEEE International Symposium on Hardware Oriented Security and Trust, HOST 2023, San Jose, CA, USA, May 1-4, 2023*, pages 48–56. IEEE, 2023.

[SAB+22] Peter Schwabe, Roberto Avanzi, Joppe Bos, Léo Ducas, Eike Kiltz, Tancrède Lepoint, Vadim Lyubashevsky, John M. Schanck, Gregor Seiler, Damien Stehlé, and Jintai Ding. CRYSTALS-KYBER. Technical report, National Institute of Standards and Technology, 2022. available at https://csrc.nist.gov/Projects/post-quantum-cryptography/selected-algorithms-2022.

[SM15] Tobias Schneider and Amir Moradi. Leakage assessment methodology - A clear roadmap for side-channel evaluations. In Tim Güneysu and Helena Handschuh, editors, *CHES 2015*, volume 9293 of *LNCS*, pages 495–513. Springer, Heidelberg, September 2015.

[SM21a] Aein Rezaei Shahmirzadi and Amir Moradi. Re-consolidating first-order masking schemes. *IACR TCHES*, 2021(1):305–342, 2021. https://tches.iacr.org/index.php/TCHES/article/view/8736.

[SM21b] Aein Rezaei Shahmirzadi and Amir Moradi. Second-order SCA security with almost no fresh randomness. *IACR TCHES*, 2021(3):708–755, 2021. https://tches.iacr.org/index.php/TCHES/article/view/8990.

[TS13a] Mostafa M. I. Taha and Patrick Schaumont. Differential power analysis of MAC-Keccak at any key-length. In Kazuo Sakiyama and Masayuki Terada, editors, *IWSEC 13*, volume 8231 of *LNCS*, pages 68–82. Springer, Heidelberg, 2013.

[TS13b] Mostafa M. I. Taha and Patrick Schaumont. Side-channel analysis of mac-keccak. In *2013 IEEE International Symposium on Hardware-Oriented Security and Trust, HOST 2013, Austin, TX, USA, June 2-3, 2013*, pages 125–130. IEEE Computer Society, 2013.

[YK20] Shih-Chun You and Markus G. Kuhn. A template attack to reconstruct the input of SHA-3 on an 8-bit device. In Guido Marco Bertoni and Francesco Regazzoni, editors, *COSADE 2020*, volume 12244 of *LNCS*, pages 25–42. Springer, Heidelberg, April 2020.

[YK21] Shih-Chun You and Markus G. Kuhn. Single-trace fragment template attack on a 32-bit implementation of keccak. In Vincent Grosso and Thomas Pöppelmann, editors, *Smart Card Research and Advanced Applications - 20th International Conference, CARDIS 2021, Lübeck, Germany, November 11-12, 2021, Revised Selected Papers*, volume 13173 of *Lecture Notes in Computer Science*, pages 3–23. Springer, 2021.

[ZSM21] Sara Zarei, Aein Rezaei Shahmirzadi, and Amir Moradi. Low-latency keccak. https://github.com/Chair-for-Security-Engineering/Low-Latency_Keccak, 2021. Accessed: December 13, 2023.

[ZSS+21] Sara Zarei, Aein Rezaei Shahmirzadi, Hadi Soleimany, Raziyeh Salarifard, and Amir Moradi. Low-latency keccak at any arbitrary order. *IACR TCHES*, 2021(4):388–411, 2021. https://tches.iacr.org/index.php/TCHES/article/view/9070.

[ZZW+22] Cankun Zhao, Neng Zhang, Hanning Wang, Bohan Yang, Wenping Zhu, Zhengdong Li, Min Zhu, Shouyi Yin, Shaojun Wei, and Leibo Liu. A compact and high-performance hardware architecture for CRYSTALS-dilithium. *IACR TCHES*, 2022(1):270–295, 2022.