

Uniform Sharing in Multiple Stages: Towards Second-Order Security

Aein Rezaei Shahmirzadi¹ , Artemii Ovchinnikov^{2*} , and Siemen Dhooghe² 

¹ PQShield, Oxford, UK

aein.shahmirzadi@pqshield.com

² COSIC, KU Leuven, Leuven, Belgium

firstname.lastname@esat.kuleuven.be

Abstract. Mitigating Side-Channel Analysis (SCA) of cryptographic implementations relies heavily on masking schemes; however, achieving efficient and straightforward hardware realizations remains challenging. While Threshold Implementation (TI) provides provable security against glitches in hardware, it incurs significant area overhead due to the large number of shares required to protect non-linear functions. Successor schemes achieve the same security order with a minimal number of shares, but often rely on a high amount of per-cycle randomness, especially at higher orders. Recent optimizations such as NULLFRESH and MULTISTAGE attain minimal share counts without additional randomness; however, they lack a generalized methodology for higher-order security. In particular, although MULTISTAGE enables automated generation of low-randomness masked designs, it is currently limited to first-order protection.

In this work, we bridge this gap by introducing an automated masking technique for arbitrary functions that guarantees second-order security in the glitch-extended probing model. Building upon the MULTISTAGE approach, our method preserves the efficiency of NULLFRESH by minimizing requirements for fresh masks, while overcoming structural limitations of previous schemes. We demonstrate the feasibility of the approach on the lightweight block ciphers PRINCE, MIDORI, and SKINNY. The resulting designs are formally verified using SILVER and PROLEAD, and experimentally evaluated on FPGA using TVLA.

Keywords: Masking · Side-Channel Analysis · Higher-Order Security · Uniformity · Hardware

1 Introduction

Following Kocher et al.’s seminal work on power side-channel attacks [25], extensive effort has focused on developing countermeasures. Masking schemes are the most common approach due to their solid theoretical foundation and well-understood security guarantees. They randomize sensitive values during cipher execution, breaking the link between secret-dependent data and the device’s physical behavior. Typically, each sensitive variable is split into multiple shares, and all

* Corresponding author.

computations occur on these shares, effectively implementing the cryptographic function in the secret-sharing domain.

Implementing masking schemes on hardware is challenging, mainly due to unintended gate switching activity, called glitches. Threshold Implementation (TI) [28] was the first approach to offer provable security in the presence of glitches, with the required number of shares depending on the function’s algebraic degree and the target security order. This dependence often makes high-order implementations too costly. In addition, although TI can secure against probing adversaries [22,18], constructing a uniform sharing remains difficult. Later, Domain Oriented Masking (DOM) [21] broke the link between the number of input shares and the algebraic degree, reducing the requirement to $d + 1$ input shares for security order d . However, achieving probing security and uniformity typically requires large amounts of fresh randomness, especially at higher orders, making the design inefficient for many applications.

There is a considerable body of work aiming to make these schemes more efficient, typically targeting specific cryptographic algorithms. Notably, a methodology, called NULLFRESH, presented in [34] achieves first-order security with two shares and no fresh randomness. This approach was later expanded in [35] for second-order security, although it works only for certain quadratic functions. While both techniques reduce implementation costs, they rely on a search algorithm that does not guarantee a solution (e.g., failing for the first-order secure PRINCE S-box [34]).

A uniform sharing with three shares has been presented in [33] but requires fresh randomness for probing security. Hence, applying these techniques to arbitrary functions is challenging, as none of them offers a generic approach. Recently, the MULTISTAGE technique was introduced in [29], offering a generic method to achieve first-order secure and uniform sharing for arbitrary functions using two input shares and no extra randomness, at the cost of increased latency in terms of clock cycles.

It is important to highlight another line of research based on composability notions, such as Strong Non-Interference (SNI) [2] and Probe Isolating Non-Interference (PINI) [14]. The idea behind these approaches is to decompose the circuit into smaller elementary functions (like AND and XOR) and construct secure gadgets for them that are trivially composable. While this automates security, the conservative design assumptions result in costly gadgets with high area overhead and randomness requirements, often making them less suitable for real-world scenarios compared to handcrafted designs.

Conversely, manually optimized designs are susceptible to subtle flaws that are difficult to detect. Although state-of-the-art verification and statistical testing frameworks such as SILVER [24] and PROLEAD [27] enable automated leakage detection, they face significant scalability limitations. Consequently, their high computational complexity often restricts their applicability to full-scale cryptographic cores, which must be analyzed in their entirety when masking is applied in a hand-crafted manner. Although the MULTISTAGE technique [29] makes it possible to effectively avoid possible design flaws without relying on fresh randomness, there is a distinct need to improve existing approaches and develop a

generic method that ensures second-order security while maintaining low randomness requirements and a minimal number of shares.

Contributions. In this work, we address the downsides outlined above by proposing a novel technique to achieve a second-order probing-secure uniform masking for arbitrary functions that can be applied coordinate function by coordinate function, provided that the number of output bits does not exceed the number of input bits. To this end, we extend MULTISTAGE technique to three shares to automatically generate a first-order masked variant using three shares and no extra fresh randomness. Then we add fresh randomness to make the circuit second-order secure. Since these random bits are only needed to maintain second-order security of non-linear functions, the vast majority of this randomness can be reused in other S-boxes and remain constant during the execution of the cipher, reducing the randomness complexity of the designs.

While this technique does not offer arbitrary composable security, resilient uniformity [16], a stronger notion of uniformity, in addition to probing security, still guarantees sequential composability [17, Lemma 2]. Thus, when applying the automatic masking technique, implying correct, non-complete and uniform shares, on each S-box of a substitution-permutation cipher, the circuit is claimed to be second-order robust probing secure in case of the sequential composability. The original work on MULTISTAGE uniformity shows that sequential composability suffices to claim security for symmetric-key ciphers [29, Section 2.4].

This work represents a step forward in the generation of automated masked designs by avoiding trivially composable gadgets, thereby reducing implementation costs and minimizing the potential for errors typically associated with handcrafted designs. The technique has been proven secure in the glitch-extended probing model and has been applied to several ciphers such as MIDORI, PRINCE, and SKINNY. Implementation costs for each of these applications have been calculated and compared with previous studies. The masking schemes have been evaluated using SILVER [24] and the PROLEAD tool [27], and they have also been tested on FPGA using TVLA [19]. All RTL codes are provided online ³.

2 Preliminaries

2.1 Notation

We denote stochastic variables by capital letters, *e.g.*, X , and deterministic Boolean variables by lowercase letters, *e.g.*, x . Share indices are denoted by subscripts, *e.g.*, x_i , while stage indices are denoted by superscripts, *e.g.*, x^i . By a *stage*, we refer to a block of combinational logic between two registers. A vector of shares is denoted in bold as $\mathbf{x} = (x_0, x_1, \dots, x_{s-1}) \in \mathbb{F}_2^s$, where s is the total number of shares. The same notation applies to shared functions: $\mathbf{g}(\cdot)$ refers to the shared representation of a Boolean function g . A function that depends on all shares except the i -th is denoted by:

$$\tilde{\mathbf{g}}_i(\cdot) = \mathbf{g}(x_0, \dots, x_{i-1}, x_{i+1}, \dots, x_{s-1})$$

³ https://github.com/KULeuven-COSIC/MultiStage_Uniformity

The Boolean negation of a bit x is denoted by a bar, $\bar{x}$. The XOR operation is denoted by $\oplus$ or $+$; we employ these notations interchangeably throughout this paper. The AND operation is denoted by $\wedge$, but is omitted for simplicity in our equations. If a value y is a multi-bit variable, such as a nibble or a byte, we represent it as a vector over the field $\mathbb{F}_2$. To avoid confusion with a vector of shares, we write $y = (y[n-1], \dots, y[1], y[0]) \in \mathbb{F}_2^n$, where n denotes the number of bits. A masked n -bit value with s shares is then represented as:

$$\mathbf{y} = (y_0[n-1], \dots, y_0[0], \dots, y_{s-1}[n-1], \dots, y_{s-1}[0]) \in \mathbb{F}_2^{ns}$$

Bit vectors are enumerated in big-endian order, which corresponds to the standard ordering in hardware implementations. Functions acting on multi-bit values can be expressed similarly. The values that are drawn uniformly at random are denoted as $r \xleftarrow{\$} \mathbb{F}_2$. Algorithms are described using pseudo-code notation.

2.2 Security Models

The d -probing model (or ISW probing model), introduced in [22], was a significant first step toward formally defining side-channel adversaries. In this model, an adversary is allowed to probe up to d intermediate values of a circuit. Yet, it became clear that this abstract model does not align with physical hardware behavior. Specifically, it fails to capture the impact of *glitches*, meaning “secure” designs under this model can still leak information in practice [26]. *Glitches* are physical phenomena that cause additional unwanted transitions in combinatorial gates, affecting the overall power consumption and potentially leaking sensitive information. In hardware, they are normally stopped by registers.

To accommodate these characteristics, the “glitch-extended probing model” was introduced [31] and formally defined in [18]. Under this model, any probe in a combinatorial circuit is extended backward to the nearest registers or primary inputs of the circuit. This means that by probing a single wire, an adversary effectively observes all the preceding signals that contribute to that wire’s calculation. This approach is intentionally conservative and consequently robust. To the best of our knowledge, it establishes the most rigorous adversary model to date for capturing the effects of glitches. The robust glitch-extended probing model is sufficiently strong to imply security in both the practical noisy leakage model [30] and the bounded moment leakage model, as shown by Barthe *et al.* [3]. By restricting the adversary to the d -th statistical moment of the leakage distribution, the latter model connects naturally to practical leakage assessment techniques, including TVLA [19], commonly employed to evaluate the SCA resistance of masked implementations. In the following, we restate definitions of probing adversaries from the original works.

Definition 1 (d -probing adversary [22]). *Consider a masked algorithm represented as a circuit, a directed acyclic graph where each edge (wire) corresponds to a digital value (bit) and each vertex is a binary operation. In the d^{th} -order probing model, the adversary is allowed to probe up to d intermediate wires of the circuit during the algorithm’s execution.*

Definition 2 (d -robust glitch-extended probing adversary [18]). *Consider the same setting as defined for the ISW probing adversary. A robust glitch-extended probing adversary of order d may observe all wires on which the d probed wires depend, tracing back to the register outputs of the current stage, or to the primary inputs if the stage is the first.*

Based on Definition 2, the security of an algorithm is defined as follows:

Definition 3 (d -robust glitch-extended probing security [18]). *Consider a masked algorithm represented as a circuit, modeled as a directed acyclic graph where each edge (wire) carries a digital value (bit) and each vertex represents a binary operation. We assume the presence of a d -glitch-extended probing adversary with access to this circuit. If the observed values are simulatable by a simulator that is not given any of the secret values, the masked algorithm is considered secure under the d^{th} glitch-extended probing model.*

In other words, it suffices to show that the probed values follow a fixed distribution independent of the secret values. The introduction of this simple model facilitated the development of formal verification tools for small circuits [24] and statistical tools for larger circuits [27].

Using the security model from Definition 3, the authors of [29] argue that the security of an entire block cipher can be established if it is composed of uniformly masked blocks, based on the notion of sequential composability [11]:

Definition 4 (Sequential Composability [11]). *Two blocks are considered sequentially composed if:*

- *the second block uses the full output of the first block;*
- *the second block operates on inputs independent of the first block, which also covers parallel composition.*

Even though the primitives considered in Sections 3 and 4 are glitch-extended probing secure, they do not necessarily meet the requirements of trivial composable gadgets like SNI and PINI. Consequently, the goal of the remainder of this paper is to provide sufficient evidence that block ciphers masked using the generalized MULTISTAGE technique remain glitch-extended probing secure. Since the block ciphers considered in that work are composed of permutation blocks that preserve uniformity given a uniform input and are composed sequentially, the uniformity of the output of the entire construction can be established. Section 2.4 describes how to generate a uniform masking with two shares for an arbitrary function. Moreover, it was shown in [17] that a threshold circuit composed of layers that are uniform, correct, and non-complete is first-order robust-probing secure. Building on these results, we extend their application to second-order ($d = 2$) security in Section 3.

2.3 Boolean Masking

One of the most widely used and easily applicable masking techniques for symmetric-key cryptography is Boolean masking [15,20]. In s -share Boolean masking, a secret

bit $x \in \mathbb{F}_2$ is shared as a vector $\mathbf{x} = (x_0, \dots, x_{s-1}) \in \mathbb{F}_2^s$ satisfying:

$$\bigoplus_{i=0}^{s-1} x_i = x$$

The sharing is said to be uniform if $(X_0, \dots, X_{s-2})$ are uniformly distributed over $\mathbb{F}_2^{s-1}$. Typically, the shares are generated as follows:

$$(x_0, \dots, x_{s-2}) = (r_0, \dots, r_{s-2}) \xleftarrow{\$} \mathbb{F}_2^{s-1}, \quad x_{s-1} = x \oplus \bigoplus_{i=0}^{s-2} r_i$$

Boolean masking is particularly well-suited for functions composed of Boolean operations, offering a natural and efficient protection strategy. For linear functions, the application of Boolean masking is straightforward: due to the linearity property $L(x \oplus y) = L(x) \oplus L(y)$, the function can simply be applied to each share independently without requiring interaction between them. However, securing non-linear functions presents a significant challenge, especially under the glitch-extended probing model. As the input and output bit-widths increase and the algebraic degree of the function rises, constructing a glitch-robust masked realization becomes exponentially more complex, often requiring sophisticated gadgets or substantial randomness to maintain security. One of the first techniques to formally describe the masking of both linear and non-linear functions and proven to be secure under the glitch-extended probing model is Threshold Implementations (TI), which we briefly explain below:

Threshold Implementations The original work [28] relies on three properties to provide security against first-order attacks, extended later for higher orders in [7].

Definition 5 (Properties of Threshold Implementations [28,7]). *Let $F : \mathbb{F}_2^n \rightarrow \mathbb{F}_2^m$ be a function and $\mathbf{F} : \mathbb{F}_2^{n s_x} \rightarrow \mathbb{F}_2^{m s_y}$ be a sharing of F . The sharing $\mathbf{F}$ must satisfy the properties described below:*

- **Correctness:** *The masking scheme must function correctly, meaning that the recombination of the output shares must yield the correct result. Formally, if a function F is shared into s_y component functions $f_1, \dots, f_{s_y}$ operating on input shares $\mathbf{x} = (x_1, \dots, x_{s_x})$, then:*

$$\bigoplus_{i=1}^{s_y} f_i(\mathbf{x}) = F \left(\bigoplus_{j=1}^{s_x} x_j \right)$$

- **d^{th} -order non-completeness:** *This is the core property that provides glitch resilience. It requires that each component function f_i be independent of at least one input share. This ensures that a single probe cannot access all the shares necessary to reconstruct the secret value.*
- **Uniformity:** *To prevent leakage from cascading when functions are composed, the output shares must follow a uniform distribution. This means that for every*

valid unmasked input, the shared representation (the tuple of shares) produced by the function must be uniformly distributed over all valid sharings. More formally, $\mathbf{F}$ should map a uniform random sharing of any $x \in \mathbb{F}_2^n$ to a uniform random sharing of $F(x) \in \mathbb{F}_2^m$.

TI is applied at the functional level, *i.e.*, to entire Boolean functions rather than individual Boolean operations, and can therefore be costly, mainly due to the large number of required input shares. Typically, the minimum number of input shares is $td+1$, where t denotes the algebraic degree of the target function and d is the desired security order. This proliferation of shares inevitably results in increased area overhead and higher consumption of fresh randomness, particularly when composing functions or enforcing uniformity. To reduce implementation costs, some works propose decomposing a function into several lower-degree functions [8]; however, guaranteeing uniformity for each sub-function remains challenging. In addition, the integration of non-linear functions often requires register layers to preserve non-completeness, which may increase latency, especially when a large number of small functions is used.

Domain-Oriented Masking Two studies have explored decoupling the number of input shares from the algebraic degree of a masked function [31,21]. They propose approaches that achieve d^{th} -order security using only $d+1$ input shares, while also providing glitch resistance in hardware, potentially reducing implementation costs due to the smaller number of shares. Typically, these methodologies require additional randomness to ensure non-completeness, in contrast to threshold implementations, which preserve non-completeness by construction. To ensure that non-completeness is preserved after re-masking, even in the presence of glitches, a $(d+1)$ -shared Boolean function is divided into two *stages* by a register layer.

We focus on the example of the approach presented by Groß *et al.* [21], known as Domain-Oriented Masking (DOM). To illustrate its application to non-linear Boolean functions, we consider a Boolean AND gate, $y = f(a, b) = ab$. The two-share implementation is organized as a two-stage logic: first, each shared monomial is evaluated separately, and then the resulting pairs are combined so that one variable remains independent of one of its shares, forming the so-called *domains*. The dashed vertical line represents a register, while the solid horizontal line highlights the domains.

$$\begin{array}{rclcl}
 f_{00}(a_0, b_0) & = & a_0 b_0 & \rightarrow & x_{00}^0 \mid \\
 f_{01}(a_0, b_1, r) & = & a_0 b_1 + r & \rightarrow & x_{01}^0 \mid & x_{00}^0 + x_{01}^0 = y_0^1 \\
 \hline
 f_{10}(a_1, b_0, r) & = & a_1 b_0 + r & \rightarrow & x_{10}^0 \mid & x_{10}^0 + x_{11}^0 = y_1^1 \\
 f_{11}(a_1, b_1) & = & a_1 b_1 & \rightarrow & x_{11}^0 \mid
 \end{array} \tag{1}$$

A fresh random bit $r \xleftarrow{\$} \mathbb{F}_2$ is applied before the register stage to ensure probing security of y_0^1 and y_1^1 . This refreshing step is applied before the register stage to prevent glitches from revealing information about the unmasked secret bits. The functions f_{**} are referred to as *component functions*, and their outcomes are

meant to be stored in registers x_{**}^0 . The product $a_i b_j$, where $i \neq j$, is referred to as a *cross-domain term*. The segment that produces output shares by XOR-ing registered values is called the *compression layer*.

NULLFRESH masking To avoid the use of fresh randomness, a technique for realizing first-order $d + 1$ hardware implementations has been presented in [34], which has been informally named NULLFRESH masking. We demonstrate its application to the same AND gate, considered above:

$$\begin{array}{lll}
 f_{00}(a_0, b_0) = a_0 b_0 + b_0 & \rightarrow x_{00}^0 & \\
 f_{01}(a_0, b_1) = a_0 b_1 & \rightarrow x_{01}^0 & x_{00}^0 + x_{10}^0 = y_0^1 \\
 f_{10}(a_1, b_0) = a_1 b_0 + b_0 & \rightarrow x_{10}^0 & x_{10}^0 + x_{11}^0 = y_1^1 \\
 f_{11}(a_1, b_1) = a_1 b_1 & \rightarrow x_{11}^0 &
 \end{array} \quad (2)$$

NULLFRESH achieved probing security by introducing a redundant share b_0 . Depending on the values of a_0 and a_1 , functions (f_{00}, f_{01}) and (f_{10}, f_{11}) , respectively, take either values $(0, b_1)$ or $(b_0, 0)$. Consequently, the second stage in Equation 2 depends on a single share of both a and b , and shares b_0, b_1 are not mixed due to the register layer, stopping glitches. The resulting values y_0^1, y_1^1 , although they formally depend on both shares of b , are non-complete by design, and are therefore robust-probing secure for $d = 1$. Using a brute-force approach, it is possible to uncover sharings for more complex functions while minimizing the number of input and output shares. However, such implementations necessitate multiple stages, and not every function masked using this approach can preserve uniformity [9]. Recently, another approach based on NULLFRESH multi-stage masking of non-linear functions has been introduced, providing a more straightforward and efficient method for generating masked implementations without using fresh randomness. This technique is known as MULTISTAGE masking.

2.4 MULTISTAGE Masking Technique

The MULTISTAGE technique [29] is a masking methodology that guarantees first-order security and output uniformity using two shares, without requiring fresh randomness. This holds provided that the output bit-width of the target function does not exceed its input size. Let us define the target function as $z = f(x)$, where $x \in \mathbb{F}_2^n$ and $z \in \mathbb{F}_2^k$, with the condition that $k \leq n$. The uniformity of the masked realization $\mathbf{F}$, operating on two input shares (x_0, x_1) , relies on the property that the mapping from (x_0, x_1) to $(z_0 = f(x) + x_0, z_1 = x_0)$ is always uniform, as established in [29, Theorem 1]. Consequently, the implementation challenge lies solely in generating the first output share, $y_0 = f(x) + x_0$, since the second output share is simply the first input share x_0 and requires no computation.

In the MULTISTAGE technique, constructing a first-order secure masked variant begins by expanding the target coordinate functions $f(x)$ into 2^n component functions, where n is the width of the input x . This is achieved via an *Expand* function, which utilizes a core function called *DirectShare_{FO}* [29, Algorithms 1 and 2]. Crucially, each resulting component function is first-order non-complete.

To prevent glitch propagation, the output of every component function is immediately registered. Subsequently, these outputs are compressed over several stages, with the intermediate results of each stage being stored in registers—a structural characteristic that gives the “MULTISTAGE” technique its name. Each component function f_i , where $i \in \{0, \dots, k-1\}$, is masked using distinct bits from the zeroth input share, denoted as $x_0[j]$ with $j \in \{0, \dots, n-1\}$.

Applying this technique to the same Boolean AND operation, using a slightly changed notation $f(x) = x[0]x[1]$ with $x[0] = a, x[1] = b$, yields the following masking where vertical dashed lines are register layers.

$$\begin{array}{lll}
 f_{00}^0 = x_0[0]x_0[1] + x_0[0] + x_0[1] & & \\
 f_{01}^0 = x_0[0]x_1[1] & | & f_{0*}^1 = x_0[0]x[1] + x_0[0] + x_0[1] \\
 f_{10}^0 = x_1[0]x_0[1] + x_0[1] & | & f_{1*}^1 = x_1[0]x[1] + x_0[1] \\
 f_{11}^0 = x_1[0]x_1[1] & | & f_0^2 = x[0]x[1] + x_0[0] \\
 \hline
 f_1^0 = x_0[0] & | & f_1^1 = x_0[0] \\
 & | & f_1^2 = x_0[0]
 \end{array} \tag{3}$$

This technique has been shown to be first-order secure and comparable to other automatic masking generation approaches in benchmark evaluations [29]. Since algorithms from [29], restated in Appendix A, generate trivial uniform maskings with two shares for arbitrary functions, it is natural to investigate whether this approach can be extended to three shares in order to achieve second-order security.

3 Second-Order Uniform Sharing in Multiple Stages

In this section, we extend the MULTISTAGE technique, briefly described in Section 2.4, to generate second-order secure masked realizations of arbitrary functions, provided the output bit-width does not exceed the input bit-width. To achieve this, we first adapt the MULTISTAGE framework to operate with three shares instead of two. Specifically, we present the algorithm to produce first-order secure designs using three shares without requiring fresh randomness.

The foundational step is the *DirectShare_{SO}* function, which expands the target function based on the i^{th} input bit. In essence, this direct sharing algorithm provides a non-complete sharing of the function into multiple output terms with respect to a single input variable. The procedure outlined in Algorithm 1. It accepts an n -bit function f and an index i , which specifies the input bit used for sharing. We designate this algorithm as *DirectShare_{SO}* (Second-Order) as it utilizes three shares, the minimum number of shares for second-order security. The underlying principle relies on the property that, for Boolean functions, any monomial in the ANF is linear with respect to any individual variable $x[i]$. Consequently, terms involving $x[i]$ can be split trivially by substituting the shared representation of $x[i]$.

To illustrate the operation of *DirectShare_{SO}*, consider the target function $f(x) = x[0]x[1]$ (an AND gate) and the splitting index $i = 1$. Our objective is to decompose $f(x)$ into three component functions f_0, f_1, f_2 based on the shares of the input bit $x[1]$.

Algorithm 1 *DirectShare_{SO}*: Direct sharing with *three* shares

Input: A function $f(x[0], \dots, x[n-1]) \in \mathbb{F}_2$ and an arbitrary index $i \in \{0, \dots, n-1\}$

Output: Three shares (z_0, z_1, z_2) over a chosen index variable $x[i]$

- 1: Consider $f = c + x[i]d = c + (x_0[i] + x_1[i] + x_2[i])d$
 - 2: Set $z_0 = c + x_0[i]d + x_0[i-1 \bmod n] + x_1[i-1 \bmod n]$
 - 3: Set $z_1 = x_1[i]d + x_1[i-1 \bmod n] + x_2[i-1 \bmod n]$
 - 4: Set $z_2 = x_2[i]d + x_2[i-1 \bmod n] + x_0[i-1 \bmod n]$
 - 5: **return** (z_0, z_1, z_2)
-

Let the three Boolean shares of $x[1]$ be denoted as $x_0[1]$, $x_1[1]$, and $x_2[1]$. We expand the function by substituting $x[1]$ with the sum of its shares:

$$f(x) = x[0]x[1] = x[0](x_0[1] + x_1[1] + x_2[1]) = \underbrace{x[0]x_0[1]}_{f_0} + \underbrace{x[0]x_1[1]}_{f_1} + \underbrace{x[0]x_2[1]}_{f_2}$$

From this expansion, we define the three resulting component functions as:

$$f_0 = x[0]x_0[1], \quad f_1 = x[0]x_1[1], \quad f_2 = x[0]x_2[1]$$

In summary, *DirectShare_{SO}* accepts a target function and an input index i , producing three shared functions. Crucially, each resulting function f_j depends on only a single share of the target input bit $x[i]$, thereby ensuring non-completeness with respect to that specific input. To guarantee second-order non-completeness and ensure secure composition of the resulting gadget, the output of Algorithm 1 is augmented with two shares of the next input variable, allocated in a cyclic manner. This approach can be generalized to an arbitrary number of shares.

Based on the algorithms from Appendix A and Algorithm 1, it becomes clear that the original algorithms can be adapted to operate with an increased number of shares. Building upon that, we present Algorithm 2 for generating first-order secure and uniform masked realizations of arbitrary functions with three shares. Fundamentally, the adjusted procedure iteratively applies *DirectShare_{SO}* to the target function across all n input bits via the updated *Expand* function (Algorithm 6). This process generates 3^n component functions (each satisfying second-order non-completeness) while strategically incorporating linear terms to ensure the compression layer remains first-order secure.

To establish the uniformity of the output generated by Algorithm 2, we generalize the corresponding theorem from [29], extending it to the case of three or more shares:

Lemma 1. *The masked function $\tilde{F} : \mathbb{F}_2^{sn} \rightarrow \mathbb{F}_2^{sk}, : \tilde{F}(x_0, \dots, x_{s-1})$, where the shared functions are*

$$f_0 = f(x) + x_0[0, \dots, k-1] + \dots + x_{s-2}[0, \dots, k-1],$$

$$f_1 = x_0[0, \dots, k-1], f_2 = x_1[0, \dots, k-1], \dots, f_{s-1} = x_{s-2}[0, \dots, k-1]$$

of the function $f : \mathbb{F}_2^n \rightarrow \mathbb{F}_2^k$, with $k \leq n$, is uniform, where $x = x_0 + \dots + x_{s-1}$ and s denotes the number of shares.

Algorithm 2 MULTISTAGE uniform sharing with *three* shares**Input:** A function $g(x[0], \dots, x[n-1]) \in \mathbb{F}_2$ and an arbitrary index $j \in \{0, \dots, n-1\}$ **Output:** A masked gadget $\mathcal{G}$ performing compression of $(f_l)_{0 \leq l < 3^n}$ over n stages for every $f_{3k}, f_{3k+1}, f_{3k+2}$, where $k \in 0, \dots, 3^{n-l}$. **Security order is by default $d = 1$, can be extended to $d = 2$ after applying refreshing.**

- 1: Set $\mathcal{R}$ a register layer tuple (single bit memory)
- 2: Set $\mathcal{F}$ a tuple of component functions (empty at first)
- 3: Set $\Gamma : (f_{3k}, f_{3k+1}, f_{3k+2}) \mapsto f'_k$
- 4: Set $(z_l^0)_{0 \leq l < 3^n} \leftarrow \text{Expand}(g + x_0[j] + x_1[j], j, j)$
- 5: Set $z_{3^n}^0 = x_0[j]$
- 6: Set $z_{3^n+1}^0 = x_1[j]$
- 7: Append $(z_l^0)_{0 \leq l \leq 3^n}$ to $\mathcal{F}$
- 8: Set $c = 1$
- 9: **for** $c \leq n$ **do**
- 10: Set $(z_k^c)_{0 \leq k < 3^{n-c}} = \mathcal{R}(\Gamma(z_{3k}^c, z_{3k+1}^c, z_{3k+2}^c))_{0 \leq k < 3^{n-c}}$
- 11: Set $z_{3^{n-c}}^c = z_{3^{n-c}}^{c-1}$
- 12: Set $z_{3^{n-c}+1}^c = z_{3^{n-c}+1}^{c-1}$
- 13: Append $(z_k^c)_{0 \leq k \leq 3^{n-c}+1}$ to $\mathcal{F}$
- 14: **end for**
- 15: Set $(z_{ij}^0)_{i \neq j} \leftarrow \text{Ring_Refresh}((z_{ij}^0)_{i \neq j})$ $\triangleright$ Required for $d=2$
- 16: Set $(z_l^n)_{l \in [0,2]} \leftarrow \text{Ring_Refresh}(z_l^n)_{l \in [0,2]}$ $\triangleright$ Required for $d=2$
- 17: **return** $\mathcal{G} = (\mathcal{F}, \Gamma, \mathcal{R}, r)$ $\triangleright r$ from Ring_Refresh is required for $d=2$

Proof. Each separate output share is uniformly randomly distributed given that the input shares $x_0, \dots, x_{s-2}$ are uniformly randomly distributed. Since $\tilde{F}$ maps a uniform input sharing to a uniform output sharing, the function is uniform. $\square$

Applying Algorithm 2 to the AND function $f(x[0], x[1]) = x[0]x[1]$ yields the following result:

$$\begin{array}{lll}
 z_0^0 = \overline{x_0[0]}x_0[1] + x_0[0] & & \\
 z_1^0 = \overline{x_0[0]}x_1[1] & , z_0^1 = z_0^0 + z_1^0 + z_2^0 & \\
 z_2^0 = \overline{x_0[0]}x_2[1] & & \\
 z_3^0 = \overline{x_1[0]}x_0[1] + x_1[0] & & \\
 z_4^0 = \overline{x_1[0]}x_1[1] & , z_1^1 = z_3^0 + z_4^0 + z_5^0 & , z_0^2 = z_0^1 + z_1^1 + z_2^1 \\
 z_5^0 = \overline{x_1[0]}x_2[1] & & \\
 z_6^0 = \overline{x_2[0]}x_0[1] & & \\
 z_7^0 = \overline{x_2[0]}x_1[1] & , z_2^1 = z_6^0 + z_7^0 + z_8^0 & \\
 z_8^0 = \overline{x_2[0]}x_2[1] & & \\
 \hline
 z_9^0 = x_0[0] & , z_3^1 = z_9^0 & , z_1^2 = z_3^1 \\
 z_{10}^0 = x_1[0] & , z_4^1 = z_{10}^0 & , z_2^2 = z_4^1
 \end{array} \tag{4}$$

Regarding the first-order security of Algorithm 2, it is straightforward to verify that any probe placed on a component function z_i^0 is statistically independent of the secret, as these functions are by design second-order non-complete. Therefore, the proof reduces to demonstrating that these components can be safely combined in the compression layer, where groups of three component functions are XORed

at each stage. For an informal argument, consider three component functions (y , z , and w) whose outputs are stored in registers. In the subsequent stage, these values can be safely XORed because they are individually masked by $x_0[\ell - 1]$, $x_1[\ell - 1]$, and $x_2[\ell - 1]$, ensuring that each is uniformly distributed regardless of the secret. Furthermore, we enforce a specific structure where w contains the term $ax_0[\ell]$ and both y and z contain $\bar{a}x_0[\ell]$ (for some control bit a , denoted as $f(\Omega')$ in the lemma). This creates a “multiplexer-like” mechanism, guaranteeing that one of the values among y , z , and w is uniquely masked by $x_0[\ell]$, which effectively blinds one share at all times. Since all component functions are second-order non-complete and masked by $x_i[\ell - 1]$, and the “multiplexer-like” structure ensures constant blinding of one share, the set of three probes on y , z , and w remains independent of the secret. The formalization of this argument is provided in Lemma 4 in Appendix B.

Based on Lemma 4, the first-order glitch-extended security of Algorithm 2 is established, notably without the requirement for fresh masks.

Theorem 1. *Algorithm 2 is 1st-order glitch-extended probing secure given a uniform input sharing.*

Proof. The robust probing adversary can probe any $x_i[j]$, but it leads to observing only one share out of three shares of each input bit, which is insufficient to reveal any information about the secret. Alternatively, a probe can be placed on the values in z^c for $c \in \{1, \dots, n\}$. Two shares z_1^n, z_2^n of the final stage and any value z^0 in the initial stage are glitch-extended probing secure since they are non-complete with the uniform input. Security proof for such constructions was given in [17, Theorem 1]. For the remaining values z_i^c , each variable is itself uniformly distributed no matter the input secret since it can be written as

$$z_i^c = h(\{x_0, x_1, x_2\} \setminus \{x_0[c], x_1[c], x_2[c]\}) + x_0[c] + x_1[c],$$

meaning that it can be written as a function that does not depend on the shares of the variable $x[c]$ while being re-masked by the uniform shares $x_0[c], x_1[c]$. We now demonstrate that $z_i^c = z_{3i}^{c-1} + z_{3i+1}^{c-1} + z_{3i+2}^{c-1}$ is glitch-extended probing secure.

From Algorithm 2, we see that $z_{3i}^{c-1}, z_{3i+1}^{c-1}, z_{3i+2}^{c-1}$ is the direct sharing of z_i^c in the variable $j + n - c$ and that z_{3i}^{c-1} is added by $x_0[j + n - c], x_1[j + n - c]$. As a result, the addition follows the form of Lemma 4, proving the security of the compression. $\square$

To transform the provided first-order secure solution into second-order secure masking, only fresh randomness is required, since the component functions z_i^0 of Algorithm 2 are second-order non-complete by design. Consolidating Masking Schemes [31] was one of the first works to demonstrate that a function satisfying TI properties (Definition 5) and second-order non-completeness remains secure in practice against second-order attacks if its output is re-masked using a *ring-refreshing* approach. This approach was later proven to be theoretically secure under a more general notion of uniformity, called *resilient uniformity*, which states that a resilient uniform function that is second-order non-complete is secure against a second-order robust probing adversary [16].

We begin with a definition of the ring-refreshing gadget used to re-mask bits of z^0 and z^n of our MULTISTAGE construction using three shares. We extend it with a definition of resilient uniformity and a proof that a function re-masked by ring-refreshing approach is resilient uniform. Finally, we prove that Algorithm 2, extended with Lines 15 and 16, achieves second-order security against the robust probing adversary.

Definition 6 (Ring-refreshing gadget [31]). *We call the ring-refreshing gadget $\text{Ring_Refresh}_2^\rho(\{x_0, \dots, x_{\rho-1}\})$ a gadget re-masking a set of input bits:*

$$\{z_0, \dots, z_{\rho-1}\} = \text{Ring_Refresh}_2^\rho(\{x_0, \dots, x_{\rho-1}\}) :$$

$$\{r_0, r_1, \dots, r_{\rho-1}\} \xleftarrow{\$} \mathbb{F}_2^\rho, \quad z_i = x_i \oplus r_i \oplus r_{(i+1) \bmod \rho} \quad \text{for } i = 0, \dots, \rho - 1,$$

where $r_0 \oplus r_1 \oplus \dots \oplus r_{\rho-1} = 0$; and $\rho = 3^n$ for $d + 1 = 3$

This gadget ensures that any single probe observes at most two random bits, *e.g.*, r_i and r_{i+1} , and that an additional probe does not reveal further information due to the cyclic structure of the refreshing. To obtain a second-order secure masking, we apply the ring-refreshing gadget $\text{Ring_Refresh}_2^\rho$ to the cross-domain outputs of the initial stage $(z_{ij}^0)_{i \neq j}$ and to the output shares obtained after the final compression, namely $(z_l^n)_{l \in [0, 2]}$.

Before presenting the security proof of this construction, we apply Algorithm 2 ($d = 2$) to the Boolean AND operation, whose first-order masked variant with three shares is given in Equation 4.

$$\begin{array}{rcl}
 z_0^0 & & \\
 z_1^0 + r_0 + r_1 & | & z_0^1 = z_0^0 + z_1^0 + z_2^0 \\
 z_2^0 + r_1 + r_2 & | & \\
 z_3^0 + r_2 + r_3 & | & \\
 z_4^0 & | & z_1^1 = z_3^0 + z_4^0 + z_5^0 | z_0^2 = z_0^1 + z_1^1 + z_2^1 + r_0^* + r_1^* \\
 z_5^0 + r_3 + r_4 & | & \\
 z_6^0 + r_4 + r_5 & | & \\
 z_7^0 + r_5 + r_0 & | & z_2^1 = z_6^0 + z_7^0 + z_8^0 \\
 z_8^0 & | & \\
 \hline
 z_9^0 & | & z_3^1 = z_9^0 | z_1^2 = z_3^1 + r_1^* + r_2^* \\
 z_{10}^0 & | & z_4^1 = z_{10}^0 | z_2^2 = z_4^1 + r_2^* + r_0^*
 \end{array} \tag{5}$$

In Equation 5, any product of the form $x_i[0]x_j[1]$ with $i \neq j$ is called a cross-domain product and needs to be refreshed using **blue** randomness. In general, for any arbitrary function if a term mixes different indices, then it needs to be refreshed by **blue** randomness. In total, we need at most $k^n - k$ random bits, where n is the number of input bits and k is the number of shares. The number may vary for different non-linear functions, because after the initial expansion step, some component functions may lack shared monomials and remain empty, thus not needing to be re-masked. Finally, additional randomness (in **red**) is added on the last register stage refreshing the output shares. While the **blue** randomness can be reused across shared functions operating on different inputs [6] and also

cancels out during compression, the red randomness must be unique for each share of each coordinate function.

Definition 7 (Resilient uniformity [16]). *A function $F(x) = y$ is resilient uniform if for any probe in the function observing inputs with indices in a set I , there is an output bit with index j , such that $\forall x_I^* \in \mathbb{F}_2^{|I|}, y_j^* \in \mathbb{F}_2, \exists c \in \mathbb{N}$, such that $\forall x \in \mathbb{F}_2^n$ and $y \in Sh(F(x))$ with $y_j = y_j^*$:*

$$|\{x \in Sh(x) \text{ with } x_I = x_I^* \mid F(x) = y\}| = c.$$

In simple terms, after placing a probe, the output distribution is still uniformly distributed except for one bit.

We use the above definition because it allows for a straightforward security proof of our constructions. Namely, [16] first shows that any function shared by a ring refreshing is resilient uniform.

Lemma 2. *Any function refreshed by a ring refreshing from Definition 6 is resilient uniform as defined in [16, Def. 7].*

Proof. The proof is given in [16, Theorem 2]. In words, placing a single probe on a function that has been re-masked by a ring-refreshing gadget reveals at most a single output bit of that function. The remaining output bits are refreshed with fresh randomness, making them independent of what the placed probe observes. $\square$

The same work by Dhooghe and Nikova [16] also highlights that the resilient uniformity property can propagate through sequentially composed blocks (Lemma 4). Finally, we conclude by applying [16, Theorem 1], which asserts that any composition of resilient uniform functions is second-order probing secure. In short, we show that any two probes are always separated by a ring-refreshing step, allowing us to leverage the resilient uniformity framework to justify the security argument.

Theorem 2. *Algorithm 2 (with $d = 2$) is second-order glitch-extended probing secure when given a uniform input sharing.*

Proof. Consider two probes; we make the distinction of their placement where either the two probes are placed in the same or in different stages. In case the probes are placed in the same stage, the second-order non-completeness provides the security argument. In case the two probes are placed in separate stages, we can consider the ring refreshing in between the two probes in which case the security is leveraged by the resilient uniformity of the ring refreshing itself and Theorem 1 of [16]. $\square$

To ensure the independence of sequential applications of the masking technique, e.g., over several rounds of a cipher, additional randomness is added at the last stage. Since this randomness is again provided by the ring-refreshing gadget, the resulting output is resilient uniform. Moreover, because the cipher blocks considered in the next section are sequentially composed, the resilient uniformity

property propagates across the entire cipher, guaranteeing second-order security even when probes are placed between different blocks. Importantly, this final-stage randomness cannot be reused.

As a closing remark for this section, we highlight that we have proven the security of our algorithm under the glitch-extended probing model. We make no claims regarding the security of our designs in the presence of transitional leakage or cross-talk effects.

4 Case Studies

In this section, we present several case studies that illustrate the applicability of the extended MULTISTAGE masking technique in achieving second-order security. We focus on the lightweight block ciphers PRINCE, MIDORI, and SKINNY, whose 4-bit S-boxes constitute particularly suitable candidates for FPGA-based evaluation. We begin with a brief introduction to each cipher selected for assessing the security properties of the proposed technique. Block diagrams of the implemented designs are provided in Appendix D. Next, we give a comparative summary of the performance benchmarks obtained for each implementation. Finally, we conclude the section with a generic proof of glitch-extended probing security that applies uniformly to all considered block ciphers.

Table 1. Performance figures of different implementations.

Cipher	Design	Latency (cycles)	Rand. (bits/Enc)	Delay (ns)	Area (kGE)			Throughput (Gbps)	Mask. Type
						†	‡		
PRINCE	[10]	72	5568	4.00	13.4 ⁵	14.96	15.76	0.22	[DOM]
	[10]	72	4224	4.10	18.7 ⁵	19.75	20.26	0.22	[TI]
	[10]	24	10752	3.42	32.4 ⁵	40.14	44.4	0.78	[DOM]
	[10]	24	9792	3.54	177.6 ⁵	183.96	187.46	0.75	[TI]
	[35]	91	11136	3.11	19.4 ³	21.92	23.27	1.58	[NullFresh2]
	<i>this work</i>	78	2256	1.46	111.2 ⁴	118.6	120.5	3.37	[MultiStage2]
MIDORI-64	[35]	68	8576	2.86	15.5 ³	18.02	19.37	1.32	[NullFresh2]
	[23]*	153	117888	0.67	105.2 ⁴	118.77	126.23	5.62	[HPC2]
	<i>this work</i>	102	2784	1.18	105.2 ⁴	112.4	114.4	3.19	[MultiStage2]
SKINNY-64-64	[35]	132	16640	1.22	10.6 ³	13.12	14.47	1.59	[NullFresh2]
	[23]*	165	31936	0.59	36.2 ⁴	39.83	41.76	3.29	[HPC2]
	[37]*	132	42496	0.50	23.0 ⁴	28.83	32.0	3.88	[AND-XOR1]
	[37]*	132	25600	0.56	29.5 ⁴	33.13	35.06	3.46	[AND-XOR2]
	<i>this work</i>	198	4608	0.57	92.1 ⁴	101.1	99.1	3.40	[MultiStage2]

¹ UMC 180 library, ² UMC 130 library, ³ UMC 90 library, ⁴ NANGATE 45 library,

⁵ TSMC 90 library; ⁶ Nibble-serial implementation; * Auto-generated design

| w/o PRNG, † with Bivium PRNG, ‡ with Trivium PRNG.

4.1 Ciphers

PRINCE [9] is a lightweight block cipher designed for low-latency applications. It has a 64-bit state, represented as a 4×4 nibble matrix. It accepts a 128-bit key,

which is expanded into a 192-bit schedule comprising two 64-bit whitening keys and a 64-bit round key. The round function follows a Substitution-Permutation Network (SPN) structure, featuring a 4-bit S-box, `ShiftRows`, and a column-wise MC operation. Uniquely, the cipher’s 12 rounds are symmetric: the first six execute forward operations, while the final six mirror them using inverse transformations. Consequently, the implementation requires two S-boxes (where one represents an inverse of another one). As noted in [29], the PRINCE S-box and its inverse are affine equivalent. We leverage this property to optimize our implementation.

MIDORI is a block cipher, originally proposed by Banik *et al.* [1]. To stick with lightweight 4-bit S-boxes, we choose to implement a version with 64-bit state size organized into 16 nibbles. The key schedule is linear, deriving round keys by selecting either the most or least significant 64 bits or their XOR sum, combined with a round-dependent constant. The round function comprises a 4-bit S-box, a nibble permutation, and a column-mixing matrix operation. While we acknowledge the cryptanalytic weaknesses identified in [36], our choice of MIDORI is driven by the broader relevance of its components. Specifically, its S-box is shared by other prominent lightweight ciphers, like MANTIS [4], making it an excellent candidate for demonstrating our masking methodology.

SKINNY is a tweakable block cipher introduced by Beierle *et al.* [4]. For this study, we target the variant featuring a 64-bit state and a 64-bit tweakkey. The internal state comprises 16 nibbles. The cipher employs a straightforward affine tweakkey schedule, while the round function integrates a nibble-wise cubic S-box with a lightweight mixing matrix characterized by a branch number of two.

4.2 Second-order secure designs

We developed an automated tool implementing Algorithms 1, 2 and 6 to generate second-order secure masked S-boxes for all three ciphers. The tool first constructs a three-shared, first-order secure design for each component function, and subsequently re-masks the outputs of the first and last stages using fresh randomness, following the uniform sharing rules of Section 3. Since each cipher employs 4-bit S-boxes operating on 16-nibble states, every shared output nibble requires 8 fresh bits of **red** non-reusable randomness for secure sequential composition to maintain glitch-extended probing security. In addition, up to $4 \cdot (3^4 - 3) = 312$ bits of reusable, **blue** randomness are required to re-mask the outputs of the initial stage. As discussed in Section 3, these reusable bits can be utilized across all S-boxes and rounds. Their exact number can be further optimized by exploiting the monomial structure of the underlying Boolean coordinate functions, allowing cipher-specific reduction of this **blue** randomness. Moreover, pipelining inherent in our round-based architectures allows more efficient use of lightweight PRNGs such as Bivium or Trivium [13], explained in the following paragraph.

The round-based architectures used for PRINCE, MIDORI, and SKINNY are depicted in Appendix D. In all cases, the cipher state is registered both before S-box expansion and after S-box compression, resulting in six clock cycles per round. All

linear layers are trivially masked by triplicating their computations, and the key schedules of all three ciphers are linear, making masked key handling straightforward. To provide the correct amount of fresh randomness, the PRNG must generate $\lceil \frac{16 \times 8}{6} \rceil = 22$ bits per cycle, since each of the 16 S-boxes requires 8 fresh bits per round. The required throughput for [reusable](#) randomness depends on the structure of each S-box. Given the total latency of the corresponding architectures, the PRNG must supply at most $\lceil \frac{286}{72} \rceil = 4$ [blue](#) random bits per cycle for PRINCE, $\lceil \frac{278}{96} \rceil = 3$ for MIDORI, and $\lceil \frac{256}{192} \rceil = 2$ for Skinny. These bits are stored in dedicated registers so they can be reused across all S-boxes and rounds. We take the area allocated for these registers into consideration when we estimate the area with PRNGs included. This way PRNGs manage to generate enough randomness for the next round or cipher execution without the need to generate all random bits in a single cycle. Performance metrics, including area, latency, randomness consumption, throughput, and critical path delay, are summarized in Table 1. Each design is labeled according to the masking technique used to obtain the corresponding second-order SCA-secure implementation. Latency values are reported under pipelined execution assumptions, when available.

Overall, the auto-generated MULTISTAGE ($d = 2$) designs achieve the lowest randomness consumption per encryption among the considered implementations. The latency of the designs presented in this work depends on the number of bits processed by the S-boxes and scales proportionally with the number of cipher rounds. Consequently, SKINNY-64-64 exhibits higher latency due to its larger number of rounds. The critical-path delays and throughput of our implementations outperform those of handcrafted designs due to the pipelined nature of the proposed architecture. Furthermore, in some cases, the proposed designs achieve throughput comparable to that of auto-generated implementations, as observed for SKINNY, owing to the relatively simple gate structure of the cipher. On the other hand, MULTISTAGE ($d = 2$) designs incur a noticeable area overhead. This overhead is directly related to the complexity of the S-box coordinate functions, i.e., the number of involved monomials, as well as to the aggressive use of register cells resulting from the unrolling of up to 3^n shared component functions per coordinate. Therefore, the primary advantages of the generalized MULTISTAGE masking approach are the straightforward generation of non-linear masked gadgets directly from lookup-table representations and the reduced randomness cost.

4.3 Glitch-extended probing security

We discuss the glitch-extended probing security of the designs described in Section 4.2. The security of the masked S-boxes is already discussed in Section 3. In this section, we discuss the security of the entire masked cipher (applicable to PRINCE, MIDORI, and SKINNY).

Lemma 3. *Consider a masked block cipher supplied with a uniformly distributed three-shared encoding of the plaintext and the secret key. The linear functions are masked share-wise, and the non-linear layer is shared following Section 3. Then the implementation is secure in the (glitch-extended) second-order probing model,*

i.e., any two probes placed within the circuit reveal no information about the secret variables.

Proof. **Case ①:** Consider two probes placed in masked S-boxes. If both probes are placed within the same S-box, probing security follows directly from Theorem 2. If the two probes are placed in distinct masked S-boxes, these S-boxes either belong to the same round or to different rounds. In the former case, the S-boxes operate on independently masked inputs, while sharing only a limited amount of randomness (illustrated in blue in Equation 5). Since each S-box is second-order secure, a probe on the first S-box reveals at most one share of each input bit together with some randomness. Due to the independence of the S-box inputs, such observations are insufficient to reconstruct or infer information about the secret inputs of the second S-box. In the latter case, where the two S-boxes belong to different rounds, their inputs are independently masked as a result of the refreshing applied after each S-box (*i.e.*, the red randomness in Equation 5). Here, resilient uniformity of the refreshing ensures second-order probing security. Consequently, two probes located in different rounds yield observations that are statistically independent of each other and of the secret.

Case ②: Consider two probes placed within linear layers. As before, we distinguish whether the probes belong to the same round or to different rounds. If both probes are located in the same round, security follows directly from the share-wise application of all linear operations, which prevents any combination of shares. If the probes are located in linear layers of different rounds, security is again guaranteed by the state-wide refreshing performed by the masked S-boxes between the rounds, and an argument analogous to Case ① applies.

Case ③: Consider one probe placed in a linear layer and one probe placed in a masked S-box. In this scenario, probing security is ensured by the refreshing separating the non-linear and linear components. As a result, the joint distribution of the probed values remains independent of the secret.

Case ④: Finally, we consider probes placed within the key schedule. All ciphers considered in our case studies employ linear key schedules. The key state is therefore updated, stored, and combined with the data path in a share-wise manner at each round. Consequently, placing two probes within the key schedule reveals at most two shares of the secret key, which are uniformly distributed and statistically independent of the key itself. Moreover, placing one probe in the key schedule and another elsewhere in the cipher can reveal at most one share of the plaintext and one share of the key, which again conveys no information about the secrets. $\square$

5 Practical Evaluation

5.1 SILVER and PROLEAD Simulations

SILVER [24] serves as a formal verification framework for masked hardware implementations. It operates on the gate-level netlist to provide a rigorous, exhaustive proof of security. It models the joint probability distributions of intermediate signals and secret values, effectively transforming the security check into a problem

of verifying statistical independence. This approach allows SILVER to analyze circuits under the glitch-extended probing model without the risk of false negatives. Furthermore, the tool is capable of verifying the uniformity of output sharings, making it a powerful standard for validating the theoretical soundness of masked gadgets.

PROLEAD [27] is a simulation-based verification framework designed to evaluate the side-channel security of hardware implementations. Unlike formal verification tools (such as SILVER) that rely on exhaustive Boolean proofs and often face scalability bottlenecks, PROLEAD employs a statistical approach to analyze large-scale designs, including full cipher implementations. The tool operates on the gate-level netlist and evaluates security under the robust probing model, effectively capturing physical effects such as glitches and transitions. It applies statistical hypothesis testing (such as the G -test) to the probed signals, determining whether the distribution of intermediate values exhibits a statistical dependence on secret values. This methodology allows it to detect complex, high-order multivariate leakages in designs that are computationally too expensive for formal verification methods to handle.

We initially verified the first-order probing security and uniformity of all S-box designs, deliberately setting all fresh randomness signals to zero to confirm first-order security in the absence of randomness. Although it is theoretically possible to verify second-order security using SILVER, the computational complexity proves prohibitive for these designs, even on high-performance servers. Consequently, we validated the second-order security of all S-box implementations using PROLEAD.

Table 2 in Appendix E summarizes the cumulative verification results obtained via PROLEAD. All simulations were conducted on a workstation equipped with an 8-core Intel® Xeon® Gold 6244 processor (supporting 16 threads) and 256 GB of RAM. Due to the combinatorial explosion of probing sets associated with full-cipher implementations, we restricted our second-order verification to the non-linear S-box components. The analysis of a single S-box required over 50 hours of computation; consequently, verifying a complete round (comprising 16 S-boxes and linear layers) was computationally infeasible with the available resources. We further validate these findings through practical TVLA experiments in the subsequent section.

5.2 FPGA Experiments

We utilize Test Vector Leakage Assessment (TVLA) [19], a statistical methodology for detecting potential side-channel leakage. Rather than attempting to recover a secret directly, TVLA evaluates whether two sets of side-channel measurements are statistically distinguishable. It does not require the evaluator to specify a particular leakage model or implement a concrete key-recovery attack. A commonly used TVLA configuration employs Welch’s t-test to compare measurements obtained from two input sets. In the “fixed-versus-random” test, one set of traces is acquired while the device processes a fixed input (plaintext), whereas the other set is acquired using random inputs. The t-statistic is computed separately at each time sample, and values exceeding a predefined threshold, commonly $|t| > 4.5$, are treated as evidence of a statistically significant difference between the two trace

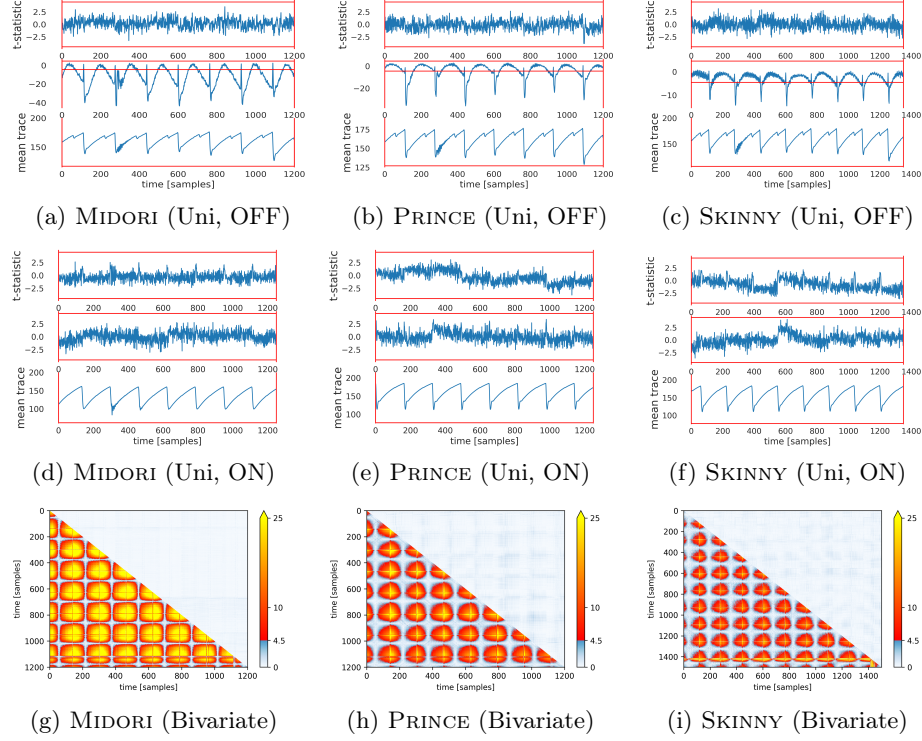


Fig. 1. Comprehensive TVLA results.

Top Row: Univariate, PRNG OFF. **Middle Row:** Univariate, PRNG ON.
Bottom Row: Bivariate (bottom-left: 5M traces/OFF, top-right: 100M traces/ON).

populations and, consequently, as an indication of potential leakage. Conversely, the absence of a threshold crossing means only that no leakage was detected under the considered measurement conditions, test configuration, and trace budget; it does not establish the absence of leakage. TVLA is therefore widely used as an efficient leakage-detection methodology and as one component of a broader side-channel security evaluation.

The experimental evaluation was performed on the Sakura-G side-channel evaluation board [32], utilizing a Tektronix DPO7254C oscilloscope for data acquisition. The setup partitions tasks between the platform’s two FPGAs: the control FPGA generates the randomness required for the initial input sharing, while the main FPGA executes the target ciphers alongside their associated PRNGs. Both devices operate synchronously at a clock frequency of 6.144 MHz. Power consumption traces were captured at a sampling rate of 1 GS/s, using an amplifier coupled to the main power line via a 1Ω shunt resistor.

We evaluated our second-order secure MIDORI, PRINCE, and SKINNY S-boxes. Since masked full-cipher designs exceeded our FPGA capacity, we restricted the analysis to the S-box level. Fresh randomness was supplied by multiple KECCAK-

p[800] instances [5]. The results of the evaluation performed with 100 million traces are shown in Figures 1d, 1e, and 1f. In addition to second-order univariate tests, we also implemented bivariate tests using the open-source SCALib library [12], relying on the same data sets as those used for the univariate analysis. As illustrated in the top-right triangles of Figures 1g, 1h, and 1i, the results demonstrate the absence of detectable leakage for up to 100 million traces. To verify the soundness of the tests and, at the same time, highlight the first-order security of Algorithm 2, we adopt an unconventional approach. Instead of disabling the initial masking of the inputs, we disable the PRNGs, thereby removing the fresh randomness required for second-order security and effectively aligning our S-boxes with first-order implementations using three shares. Although leakage in both univariate and bivariate second-order tests appears almost immediately, with only several thousand traces, we collect a total of 5 million traces to demonstrate that no leakage is observed in the corresponding first-order experiments. The bottom-left parts of Figures 1g, 1h, and 1i illustrate the resulting bivariate leakage, while Figures 1a, 1b, and 1c confirm the first-order security of the MULTISTAGE masking with three shares.

6 Conclusions

In this work, we propose a masking technique designed for the second-order sharing of arbitrary functions over multiple stages. This method ensures uniform sharing while minimizing randomness complexity by enabling the reuse of a significant portion of the random bits across all S-boxes. The approach represents an evolution of the MULTISTAGE technique. We applied this methodology to several prominent cryptographic primitives and validated their security using the formal verification tool SILVER and the simulation framework PROLEAD. Furthermore, we evaluated the side-channel leakage of our FPGA implementations using TVLA. No statistically significant leakage was detected under the considered conditions.

Acknowledgments. This work was supported by the Flemish Government through the Cybersecurity Research Program with grant number: VOEWICS02. In addition, this work was partially supported by the Flemish Government through FWO Project “MatSec”: G0ABK25N, and FWO mobility project: VS07724N.

References

1. Banik, S., Bogdanov, A., Isobe, T., Shibutani, K., Hiwatari, H., Akishita, T., Regazzoni, F.: Midori: A Block Cipher for Low Energy. In: ASIACRYPT 2015. Lecture Notes in Computer Science, vol. 9453, pp. 411–436. Springer (2015)
2. Barthe, G., Belaïd, S., Dupressoir, F., Fouque, P., Grégoire, B., Strub, P., Zucchini, R.: Strong Non-Interference and Type-Directed Higher-Order Masking. In: CCS 2016. pp. 116–129. ACM (2016)
3. Barthe, G., Dupressoir, F., Faust, S., Grégoire, B., Standaert, F., Strub, P.: Parallel implementations of masking schemes and the bounded moment leakage model. In: Coron, J., Nielsen, J.B. (eds.) Advances in Cryptology - EUROCRYPT 2017 - 36th

- Annual International Conference on the Theory and Applications of Cryptographic Techniques, Paris, France, April 30 - May 4, 2017, Proceedings, Part I. Lecture Notes in Computer Science, vol. 10210, pp. 535–566 (2017)
4. Beierle, C., Jean, J., Kölbl, S., Leander, G., Moradi, A., Peyrin, T., Sasaki, Y., Sasdrich, P., Sim, S.M.: The SKINNY family of block ciphers and its low-latency variant MANTIS. In: CRYPTO 2016. Lecture Notes in Computer Science, vol. 9815, pp. 123–153. Springer (2016)
 5. Bertoni, G., Daemen, J., Peeters, M., Assche, G.V.: The keccak reference. <http://keccak.noekeon.org>
 6. Beyne, T., Dhooghe, S., Moradi, A., Shahmirzadi, A.R.: Cryptanalysis of efficient masked ciphers: Applications to low latency. *IACR Trans. Cryptogr. Hardw. Embed. Syst.* **2022**(1), 679–721 (2022)
 7. Bilgin, B., Gierlichs, B., Nikova, S., Nikov, V., Rijmen, V.: Higher-Order Threshold Implementations. In: ASIACRYPT 2014. Lecture Notes in Computer Science, vol. 8874, pp. 326–343. Springer (2014)
 8. Bilgin, B., Nikova, S., Nikov, V., Rijmen, V., Tokareva, N.N., Vitkup, V.: Threshold implementations of small S-boxes. *Cryptogr. Commun.* **7**(1), 3–33 (2015)
 9. Borghoff, J., Canteaut, A., Güneysu, T., Kavun, E.B., Knezevic, M., Knudsen, L.R., Leander, G., Nikov, V., Paar, C., Rechberger, C., Rombouts, P., Thomsen, S.S., Yalçin, T.: PRINCE - A Low-Latency Block Cipher for Pervasive Computing Applications - Extended Abstract. In: ASIACRYPT 2012. Lecture Notes in Computer Science, vol. 7658, pp. 208–225. Springer (2012)
 10. Bozilog, D., Knezevic, M., Nikov, V.: Optimized Threshold Implementations: Minimizing the Latency of Secure Cryptographic Accelerators. In: CARDIS 2019. Lecture Notes in Computer Science, vol. 11833, pp. 20–39. Springer (2019)
 11. Canetti, R.: Universally composable security. *J. ACM* **67**(5), 28:1–28:94 (2020)
 12. Cassiers, G., Bronchain, O.: Scalib: A side-channel analysis library. *J. Open Source Softw.* **8**(86), 5196 (2023)
 13. Cassiers, G., Masure, L., Momin, C., Moos, T., Moradi, A., Standaert, F.: Randomness generation for secure hardware masking - unrolled trivium to the rescue. *IACR Commun. Cryptol.* **1**(2), 4 (2024)
 14. Cassiers, G., Standaert, F.: Trivially and Efficiently Composing Masked Gadgets With Probe Isolating Non-Interference. *IEEE Trans. Information Forensics and Security* **15**, 2542–2555 (2020)
 15. Chari, S., Jutla, C.S., Rao, J.R., Rohatgi, P.: Towards sound approaches to counteract power-analysis attacks. In: Wiener, M.J. (ed.) *Advances in Cryptology - CRYPTO '99*, 19th Annual International Cryptology Conference, Santa Barbara, California, USA, August 15–19, 1999, Proceedings. Lecture Notes in Computer Science, vol. 1666, pp. 398–412. Springer (1999)
 16. Dhooghe, S., Nikova, S.: Resilient uniformity: applying resiliency in masking. *Cryptogr. Commun.* **14**(1), 41–58 (2022)
 17. Dhooghe, S., Nikova, S., Rijmen, V.: Threshold implementations in the robust probing model. In: Bilgin, B., Petkova-Nikova, S., Rijmen, V. (eds.) *Proceedings of ACM Workshop on Theory of Implementation Security, TIS@CCS 2019*, London, UK, November 11, 2019. pp. 30–37. ACM (2019)
 18. Faust, S., Grosso, V., Pozo, S.M.D., Paglialonga, C., Standaert, F.: Composable Masking Schemes in the Presence of Physical Defaults & the Robust Probing Model. *IACR Trans. Cryptogr. Hardw. Embed. Syst.* **2018**(3), 89–120 (2018)
 19. Goodwill, G., Jun, B., Jaffe, J., Rohatgi, P.: A testing methodology for side-channel resistance validation. In: *NIST non-invasive attack testing workshop* (2011)

20. Goubin, L., Patarin, J.: DES and differential power analysis (the "duplication" method). In: Koç, Ç.K., Paar, C. (eds.) *Cryptographic Hardware and Embedded Systems, First International Workshop, CHES'99, Worcester, MA, USA, August 12-13, 1999, Proceedings. Lecture Notes in Computer Science*, vol. 1717, pp. 158–172. Springer (1999)
21. Groß, H., Mangard, S., Korak, T.: Domain-Oriented Masking: Compact Masked Hardware Implementations with Arbitrary Protection Order. In: *Theory of Implementation Security - TIS@CCS 2016*. p. 3. ACM (2016)
22. Ishai, Y., Sahai, A., Wagner, D.A.: Private circuits: Securing hardware against probing attacks. In: Boneh, D. (ed.) *Advances in Cryptology - CRYPTO 2003, 23rd Annual International Cryptology Conference, Santa Barbara, California, USA, August 17-21, 2003, Proceedings. Lecture Notes in Computer Science*, vol. 2729, pp. 463–481. Springer (2003)
23. Knichel, D., Moradi, A., Müller, N., Sasdrich, P.: Automated generation of masked hardware. *IACR Trans. Cryptogr. Hardw. Embed. Syst.* **2022**(1), 589–629 (2022)
24. Knichel, D., Sasdrich, P., Moradi, A.: SILVER - Statistical Independence and Leakage Verification. In: *ASIACRYPT 2020. Lecture Notes in Computer Science*, vol. 12491, pp. 787–816. Springer (2020)
25. Kocher, P.C., Jaffe, J., Jun, B.: Differential Power Analysis. In: *CRYPTO 1999. Lecture Notes in Computer Science*, vol. 1666, pp. 388–397. Springer (1999)
26. Mangard, S., Pramstaller, N., Oswald, E.: Successfully Attacking Masked AES Hardware Implementations. In: *CHES 2005. Lecture Notes in Computer Science*, vol. 3659, pp. 157–171. Springer (2005)
27. Müller, N., Moradi, A.: PROLEAD A probing-based hardware leakage detection tool. *IACR Trans. Cryptogr. Hardw. Embed. Syst.* **2022**(4), 311–348 (2022)
28. Nikova, S., Rechberger, C., Rijmen, V.: Threshold Implementations Against Side-Channel Attacks and Glitches. In: *ICICS 2006. Lecture Notes in Computer Science*, vol. 4307, pp. 529–545. Springer (2006)
29. Ovchinnikov, A., Shahmirzadi, A.R., Dhooghe, S.: Uniform sharing in multiple stages: Nullfresh for arbitrary functions. *IACR Transactions on Cryptographic Hardware and Embedded Systems* **2026**(2), 893–927 (Apr 2026)
30. Prouff, E., Rivain, M.: Masking against side-channel attacks: A formal security proof. In: Johansson, T., Nguyen, P.Q. (eds.) *Advances in Cryptology - EUROCRYPT 2013, 32nd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Athens, Greece, May 26-30, 2013. Proceedings. Lecture Notes in Computer Science*, vol. 7881, pp. 142–159. Springer (2013)
31. Reparaz, O., Bilgin, B., Nikova, S., Gierlichs, B., Verbauwhede, I.: Consolidating masking schemes. In: *CRYPTO 2015. Lecture Notes in Computer Science*, vol. 9215, pp. 764–783. Springer (2015)
32. SAKURA: Side-channel Attack User Reference Architecture. <http://sato.h.cs.uec.ac.jp/SAKURA/index.html>
33. Shahmirzadi, A.R., Dhooghe, S., Moradi, A.: Low-latency and low-randomness second-order masked cubic functions. *IACR Trans. Cryptogr. Hardw. Embed. Syst.* **2023**(1), 113–152 (2023)
34. Shahmirzadi, A.R., Moradi, A.: Re-Consolidating First-Order Masking Schemes - Nullifying Fresh Randomness. *IACR Trans. Cryptogr. Hardw. Embed. Syst.* **2021**(1), 305–342 (2020)
35. Shahmirzadi, A.R., Moradi, A.: Second-order SCA security with almost no fresh randomness. *IACR Trans. Cryptogr. Hardw. Embed. Syst.* **2021**(3), 708–755 (2021)
36. Todo, Y., Leander, G., Sasaki, Y.: Nonlinear invariant attack - practical attack on full scream, iscream, and midori64. In: *ASIACRYPT 2016. Lecture Notes in Computer Science*, vol. 10032, pp. 3–33 (2016)

37. Wu, L., Fan, Y., Preneel, B., Wang, W., Wang, M.: Automated generation of masked nonlinear components: - from lookup tables to private circuits. In: Andreoni, M. (ed.) Applied Cryptography and Network Security Workshops - ACNS 2024 Satellite Workshops, AIBlock, AIHWS, AIoTS, SCI, AAC, SiMLA, LLE, and CIMSS, Abu Dhabi, United Arab Emirates, March 5-8, 2024, Proceedings, Part I. Lecture Notes in Computer Science, vol. 14586, pp. 319–339. Springer (2024)

A MULTISTAGE Algorithms for $d = 1$ [29]

Algorithm 3 *DirectShare_{FO}*: Direct sharing with *two* shares

Input: A function $f(x[0], \dots, x[n-1]) \in \mathbb{F}_2$ and an arbitrary index $i \in \{0, \dots, n-1\}$

Output: Two shares (z_0, z_1) over a chosen index variable $x[i]$

- 1: Consider $f = c + x[i]d = c + (x_0[i] + x_1[i])d$
 - 2: Set $f_0 = c + x_0[i]d$
 - 3: Set $f_1 = x_1[i]d$
 - 4: Set $z_0 = f_0 + x_0[i - 1 \bmod n]$
 - 5: Set $z_1 = f_1 + x_0[i - 1 \bmod n]$
 - 6: **return** (z_0, z_1)
-

Algorithm 4 Recursive *Expand* function operating on *two* shares

Input: A function $f(x[0], \dots, x[n-1]) \in \mathbb{F}_2$, indices $i, j \in \{0, \dots, n-1\}$

Output: A tuple of $(f_i)_{0 \leq i < 2^n}$ functions

- 1: Set $f_0, f_1 \leftarrow \text{DirectShare}_{FO}(f, i)$
 - 2: Set $i = i - 1 \bmod n$
 - 3: **if** $i = j$ **then**
 - 4: **return** $(f_0 + x_0[i], f_1 + x_0[i])$
 - 5: **else**
 - 6: **return** $(\text{Expand}(f_0, i, j), \text{Expand}(f_1, i, j))$
 - 7: **end if**
-

Algorithm 5 MULTISTAGE *first-order* uniform sharing

Input: A function $g(x[0], \dots, x[n-1]) \in \mathbb{F}_2$ and an arbitrary index $j \in \{0, \dots, n-1\}$

Output: A masked gadget $\mathcal{G}$ performing compression of $(f_i)_{0 \leq i < 2^n}$ over n stages for every f_{2k}, f_{2k+1} , where $k \in 0, \dots, 2^{n-l}$

- 1: Set $\mathcal{R}$ a register layer tuple (single bit memory)
 - 2: Set $\mathcal{F}$ a tuple of component functions (empty at first)
 - 3: Set $\Gamma : (f_{2k}, f_{2k+1}) \mapsto f'_k$
 - 4: Set $(z_l^0)_{0 \leq l < 2^n} \leftarrow \text{Expand}(g + x_0[j], j, j)$
 - 5: Set $z_{2^n}^0 = x_0[i]$
 - 6: Append $(z_l^0)_{0 \leq l \leq 2^n}$ to $\mathcal{F}$
 - 7: Set $c = 1$
 - 8: **for** $c \leq n$ **do**
 - 9: Set $(z_k^c)_{0 \leq k < 2^{n-c}} = \mathcal{R}(\Gamma(z_{2k}^c, z_{2k+1}^c))_{0 \leq k < 2^{n-c}}$
 - 10: Set $z_{2^{n-c}}^c = z_{2^{n-c}}^{c-1}$
 - 11: Append $(z_k^c)_{0 \leq k \leq 2^{n-c}}$ to $\mathcal{F}$
 - 12: **end for**
 - 13: **return** $\mathcal{G} = (\mathcal{F}, \Gamma, \mathcal{R})$
-

B First-order security of directly shared compression with three shares

Lemma 4. *Consider the variables $(x[0], \dots, x[n-1]) \in \mathbb{F}_2^n$ and consider its uniform three-sharing $\Omega = ((x_0[0], x_1[0], x_2[0]), \dots, (x_0[n-1], x_1[n-1], x_2[n-1])) \in \mathbb{F}_2^{3n}$, then any probe placed within the operation $y + z + w \in \mathbb{F}_2$ only views information independent of the secret variable, with*

$$\begin{aligned} y &= g(\Omega') + \overline{f(\Omega')}x_0[\ell] + x_0[\ell-1] + x_1[\ell-1] \\ z &= h(\Omega') + \overline{f(\Omega')}x_1[\ell] + x_1[\ell-1] + x_2[\ell-1] \\ w &= l(\Omega') + f(\Omega')x_2[\ell] + x_0[\ell-1] + x_2[\ell-1], \end{aligned}$$

for

$$\Omega' = \Omega \setminus \{x_0[\ell], x_1[\ell], x_2[\ell], x_0[\ell-1], x_1[\ell-1], x_2[\ell-1]\} \cup \{x[\ell], x[\ell-1]\},$$

some $\ell \in \{0, \dots, n-1\}$ and some functions $f, g, h, l : \mathbb{F}_2^{3n-3} \rightarrow \mathbb{F}_2$.

Proof. For the joint uniformity of the distribution of the three values, the distributions of y and z are independent since they can be rewritten as

$$\begin{aligned} y &= g(\Omega') + \overline{f(\Omega')}x_0[\ell] + x[\ell-1] + x_2[\ell-1] \\ z &= h(\Omega') + \overline{f(\Omega')}x_1[\ell] + x[\ell-1] + x_0[\ell-1]. \end{aligned}$$

We note that $x[\ell-1] = x_0[\ell-1] + x_1[\ell-1] + x_2[\ell-1]$. Thus, y and z are masked by the unique uniform random values $x_1[\ell-1], x_2[\ell-1]$. The variable w is then masked by the unique value $x_2[\ell]$ since its $f(\Omega')$ is negated compared to the one in y, z .

Having established that y and z can be safely combined, the next step is to demonstrate that the distributions of $v = y + z$ and w are statistically independent. Proving this independence guarantees that the values are jointly probing-secure.

$$\begin{aligned} v &= y + z = g(\Omega') + h(\Omega') + \overline{f(\Omega')}(x[\ell] + x_2[\ell]) + x_0[\ell-1] + x_2[\ell-1] \\ w &= l(\Omega') + f(\Omega')x_2[\ell] + x_0[\ell-1] + x_2[\ell-1]. \end{aligned}$$

It is straightforward to verify that v and w are individually probing-secure. Since both are masked by the variable $x_0[\ell-1] + x_2[\ell-1]$, their distributions are uniform individually and independent of the secret. We now proceed to demonstrate that the distributions of v and w are statistically independent by proving that $\Pr[V, W] = \Pr[V] \Pr[W]$:

$$\begin{aligned} \Pr[V, W] &= \Pr[V, W | f(\Omega') = 0] \Pr[f(\Omega') = 0] + \Pr[V, W | f(\Omega') = 1] \Pr[f(\Omega') = 1] \\ &= \Pr[V | f(\Omega') = 0] \Pr[W | f(\Omega') = 0] \Pr[f(\Omega') = 0] + \\ &\quad \Pr[V | f(\Omega') = 1] \Pr[W | f(\Omega') = 1] \Pr[f(\Omega') = 1] \\ &= \Pr[f(\Omega') = 0]/4 + \Pr[f(\Omega') = 1]/4 = 1/4 \\ &= 1/4(\Pr[f(\Omega') = 0] + \Pr[f(\Omega') = 1])(\Pr[f(\Omega') = 0] + \Pr[f(\Omega') = 1]) \\ &= \Pr[V] \Pr[W], \end{aligned}$$

The second equality holds regardless of the value of the Boolean function $f(\Omega') \in \{0, 1\}$.

Case 1: If $f(\Omega') = 1$, we have:

$$\begin{aligned} v &= g(\Omega') + h(\Omega') + x_0[\ell - 1] + x_2[\ell - 1] \\ w &= l(\Omega') + x_2[\ell] + x_0[\ell - 1] + x_2[\ell - 1]. \end{aligned}$$

Here, w is masked by $x_2[\ell]$, whereas v does not depend on $x_0[\ell]$, $x_1[\ell]$, or $x_2[\ell]$. This structure demonstrates that v and w are statistically independent.

Case 2: If $f(\Omega') = 0$, the expressions become:

$$\begin{aligned} v &= g(\Omega') + h(\Omega') + (x[\ell] + x_2[\ell]) + x_0[\ell - 1] + x_2[\ell - 1] \\ w &= l(\Omega') + x_0[\ell - 1] + x_2[\ell - 1]. \end{aligned}$$

In this case, v is masked by $x[\ell] + x_2[\ell]$ (which is equivalent to $x_0[\ell] + x_1[\ell]$), while w does not depend on $x_0[\ell]$, $x_1[\ell]$, or $x_2[\ell]$. Thus, statistical independence is maintained. The third equality holds because the random variables V and W are uniform regardless of the value of $f(\Omega')$. $\square$

C EXPAND function for MULTISTAGE with $d = 2$

Algorithm 6 Recursive *Expand* function operating on *three* shares

Input: A function $f(x[0], \dots, x[n-1]) \in \mathbb{F}_2$, indices $i, j \in \{0, \dots, n-1\}$

Output: A tuple of $(f_i)_{0 \leq i < 3^n}$ functions

- 1: Set $f_0, f_1, f_2 \leftarrow \text{DirectShare}_{SO}(f, i)$
 - 2: Set $i = i - 1 \bmod n$
 - 3: **if** $i = j$ **then**
 - 4: **return** $(f_0 + x_0[i] + x_1[i], f_1 + x_1[i] + x_2[i], f_2 + x_2[i] + x_0[i])$
 - 5: **else**
 - 6: **return** $(\text{Expand}(f_0, i, j), \text{Expand}(f_1, i, j), \text{Expand}(f_2, i, j))$
 - 7: **end if**
-

D Design Figures

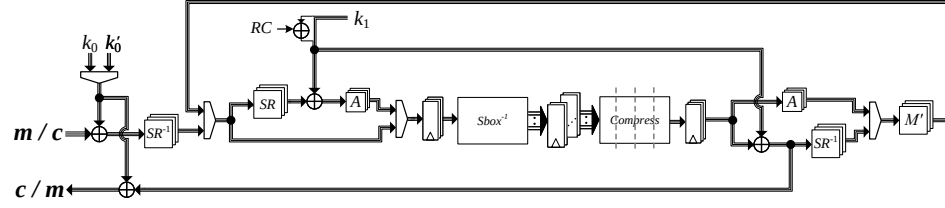


Fig. 2. Second-order secure round-based design of PRINCE.

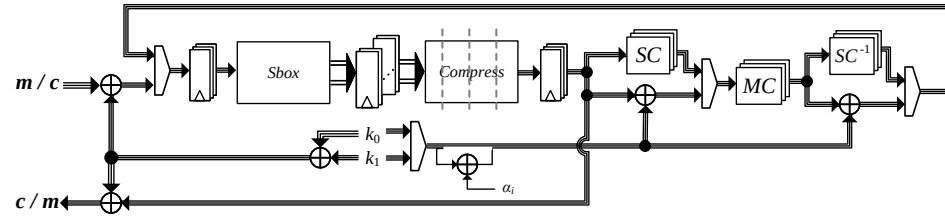


Fig. 3. Second-order secure round-based design of MIDORI-64.

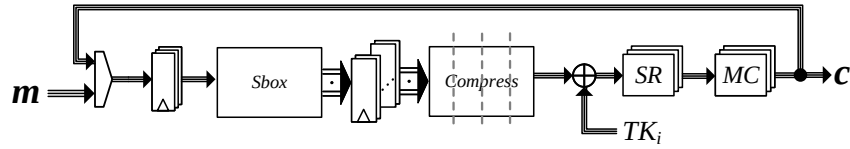


Fig. 4. Second-order secure round-based design of SKINNY.

E PROLEAD Results

Table 2. PROLEAD Simulations Results

Cipher	Cycles	RAM (GB)	Time (Hours)	#Traces (M)	Result
PRINCE	6	2.6	65.4	40	OKAY
MIDORI-64	6	2.3	55.1	40	OKAY
SKINNY-64-64	6	2.12	50.0	40	OKAY