Low-Latency Low-Randomness First-Order OPINI Gadgets and Their Formal Verification

Lixuan Wu[1,2,3], Yanhong Fan[1,2,3](✉), Guowei Liu[1,2,3], Chaoran Wang[1,2,3] and Meiqin Wang[1,2,3]

[1] School of Cyber Science and Technology, Shandong University, Qingdao, China,
{lixuanwu,yanhongfan,mqwang}@sdu.edu.cn, {guoweiliu,chaoranwang}@mail.sdu.edu.cn
[2] State Key Laboratory of Cryptography and Digital Economy Security, Shandong University, Qingdao, 266237, China,
[3] Key Laboratory of Cryptologic Technology and Information Security, Ministry of Education, Shandong University, Jinan, China

Abstract. Masking is an essential countermeasure against side-channel attacks, yet implementing secure and low-latency hardware masking remains challenging. In particular, although OPINI provides strong composability guarantees for single-cycle iterative architectures, the prior low-latency OPINI gadget, HPC4, is limited to two-input multiplication. In this work, we present a low-latency, low-randomness, first-order OPINI gadget applicable to arbitrary Boolean functions, denoted as GOM. Independent and concurrent work by Rahimi and Moradi proposes OTSM, which is also a generic, low-latency first-order OPINI gadget. Our construction involves two new techniques: (i) extending the HPC4 idea—originally masking each share of one secret input with two bits of randomness—to masking each monomial derived from the input shares accordingly, and (ii) a randomness-reassignment technique that enables the two circuits generating the output shares to reuse the same set of randomness while preserving the first-order OPINI security. To validate OPINI security, we propose a formal verification technique based on three symbolic reduction rules, and use it to verify multiple low-latency OPINI gadgets (i.e., HPC4, GOM and OTSM). Leveraging the generality of our gadget, we instantiate several OPINI-secure S-boxes across different algebraic degrees. For the algebraic-degree-2 Ascon S-box, our gadget achieves a 21% reduction in area and a 28% reduction in randomness compared to the HPC4-based implementation. We further construct higher-degree S-boxes from the PRESENT, PRINCE and AES ciphers, report their hardware performance, and provide a comparison with the concurrent work OTSM. The first-order OPINI security of all masked S-boxes is successfully verified within 20 minutes using our formal verification method. Finally, FPGA-based experiments confirm the practical security of the masked implementations.

Keywords: Hardware · Low-Latency · OPINI Gadgets · Formal Verification

1 Introduction

Masking [CJRR99, GP99], rooted in secret sharing [Sha79], is one of the most popular countermeasures against Side-Channel Analysis (SCA) attacks [Koc96, KJJ99]. In a masking scheme, computations operate on multiple randomized shares of sensitive data rather than on the data itself, thereby removing exploitable statistical dependencies between intermediate values and side-channel information. Threshold Implementation (TI) [NRR06] is the first masking strategy to avoid leakage in hardware implementations. Subsequent schemes, including Consolidating Masking Schemes (CMS) [RBN+15] and

Domain-Oriented Masking (DOM) [GMK16], further reduce hardware overhead without compromising security.

Despite these advances in masking strategies, constructing and proving secure masking for arbitrary Boolean functions remains challenging due to the prohibitive computational complexity involved. To address this, Cassiers et al. [CS20] introduced Probe-Isolating Non-Interference (PINI), an elegant notion that enables a divide-and-conquer approach for constructing large-scale masked circuits. Small PINI gadgets (e.g., HPC1 [CGLS21], HPC2 [CGLS21] and HPC3 [KM22]) can be modularly composed into larger secure circuits. Since these gadgets primarily focus on two-input multiplication, circuits with high algebraic degree typically require multi-cycle latency (e.g., a 6-cycle AES S-box constructed from HPC2 gadgets [MCS22]).

To mitigate such latency overhead, Knichel et al. [KSM22] introduced the Low-Latency Generic Hardware Private Circuits (GHPC_{LL}), a first-order PINI construction for arbitrary Boolean functions using only a single layer of registers. However, masking schemes based on GHPC_{LL} have a large implementation area and many random bits. More recently, Time Sharing Masking (TSM) [VDB+24] and Mirrored-Circuits Masking (MCM) [WFH+26] have been introduced to reduce hardware area and fresh randomness costs further while maintaining single-cycle latency. The low-latency feature has also been extended to higher-order scenarios with minimal additional latency [VDB+25, LMDM25]

However, as emphasized in [CS21], satisfying PINI is not enough to guarantee security in single-cycle iterative architectures, where transitions introduce additional leakage. This limitation motivated the authors to introduce a stronger composability notion, Output-Probe-Isolating Non-Interference (OPINI), which enables trivial composition even in a single-cycle-per-round implementation. A representative low-latency OPINI gadget is HPC4 [CSV24], which supports two-input multiplication across arbitrary security order. Independent and concurrent with our work, Rahimi and Moradi [RM26] propose OTSM, a low-latency first-order OPINI gadget that can be extended to arbitrary Boolean functions.

1.1 Contributions

This paper proposes a construction method for the low-latency, first-order OPINI gadget applicable to arbitrary Boolean functions. The proposed Generic OPINI Masking (GOM)[1] extends the capability of OPINI-secure constructions beyond the two-input multiplication while maintaining practical efficiency in terms of latency, randomness, and area. In addition, we propose a symbolic formal verification technique to validate the OPINI security of GOM. Our contributions are summarized below.

A Generic OPINI Gadget for Arbitrary Boolean Functions. Our gadget construction extends the HPC4 idea by masking each monomial derived from the input shares with two random bits to achieve OPINI security under the single-cycle latency. We further introduce a randomness-reassignment technique that enables the two circuits producing the output shares to reuse the same randomness, thereby reducing the total randomness without compromising security. Based on these techniques, we construct a generic first-order OPINI gadget that achieves both low latency and low randomness.

A Symbolic Formal Verification for the GOM **Gadget.** To validate the OPINI security, we propose a symbolic verification technique. The technique introduces three symbolic reduction rules to derive the information required to simulate each probe configuration in a target gadget. If the resulting information satisfies the OPINI definition, the gadget is considered OPINI-secure. Our technique verifies multiple single-cycle OPINI gadgets, including HPC4, GOM, OTSM.

[1] Since the construction method serves as an OPINI-secure gadget for generic Boolean functions, we refer to it as Generic OPINI Masking (GOM).

Improved Hardware Performance. Leveraging the generality of GOM, we instantiate several masked S-boxes across different algebraic degrees. For the algebraic-degree-2 Ascon S-box, our gadget achieves a 21% area reduction and a 28% randomness reduction compared to the HPC4-based implementation. We further instantiate higher-algebraic-degree (> 2) S-boxes, including the 4-bit PRINCE S-box and the 8-bit AES S-box, report their hardware performance, and provide a comparison with the concurrent work OTSM. All instantiated S-boxes are successfully verified within 20 minutes using our symbolic verification technique. The source code for all implementations is available at https://github.com/dfhsdkjc243/GOM-Gadgets-and-Their-Formal-Verification.git.

1.2 Outline

Section 2 introduces the necessary notations and background for understanding this work. In Section 3, we present the low-latency OPINI gadget for arbitrary Boolean functions, and the symbolic formal verification technique used to validate its security. Section 4 instantiates several masked S-boxes using our gadget, reports their hardware performance, and verifies their security using our formal technique. In Section 5, we conduct FPGA-based experiments to confirm the practical security of our gadgets.

2 Preliminaries

2.1 Notations

We denote a Boolean variable by a lowercase letter $x \in \mathbb{F}_2$ and a vector of Boolean variables by an uppercase letter $X \in \mathbb{F}_2^n$. The i-th share of the vector X is written using a subscript, i.e., X_i. The superscript notation X^j denotes the j-th element of the vector X. The set of all shares of all elements of vector X is denoted as $Sh(X)$. In Boolean masking, a Boolean variable x is split into $d+1$ uniform random shares, denoted as $(x_0, x_1, \ldots, x_d)$. To ensure correctness, these shares satisfy the condition $x = \bigoplus_{i=0}^{d} x_i$.

2.2 Probing Model

Several adversary models have been introduced to evaluate the security of masking schemes and to guide the construction of more masking schemes. Among them, the d-probing model is widely used for analyzing and designing masking schemes in software implementations. In this model, the parameter d, representing the number of probes, determines the order of the attack. To capture information leakage caused by physical defaults such as glitches, transitions, and couplings, Faust et al. [FGP+18] introduced the robust probing model. In this model, each physical default is abstracted using a probe extension, which increases the number of values observable through a single probe.

In this work, we focus on glitches and transitions. Glitches refer to the unpredictable signal transition activities on combinational wires, caused by the different signal delays of logic gates. Transitions are the unpredictable recombinations of register contents across consecutive invocations (e.g., clock cycles). A glitch-extended probe placed on a combinational wire observes all intermediate values up to the most recent synchronization point, such as registers. A transition-extended probe placed on a register observes a pair of values that correspond to consecutive invocations.

2.3 Trivially Composable Masking

Constructing and proving secure masking schemes for complex functions or high security orders remains challenging, due to the prohibitive computational complexity involved. For this, researchers have introduced some composability notions that enable the construction

of large circuits by combining smaller sub-circuits (often referred to as gadgets) with particular properties. To support trivial composability, Barthe et al. [BBD+16] proposed the notion of Strong Non-Interference (SNI), which ensures secure composability among single-output gadgets. This concept was later extended to multiple-output gadgets by Cassierrs et al. [CS20]. In the same work, the authors also introduced a new composable notion, Probe-Isolating Non-Interference (PINI), which further reduces the implementation overhead of gadgets while preserving trivial composability. As elaborated in [CS21], satisfying PINI may not be sufficient to guarantee security in an iterative architecture, where the input of one iteration depends on the output of a preceding one, introducing leakage from transitions. To solve this issue, the authors proposed OPINI, a stronger composability notion that accounts for both glitches and transitions. It guarantees security in such scenarios. As a necessary concept, we first recall the concept of simulability before introducing PINI and OPINI.

Definition 1 (Simulatability [CS20])**.** Let $P = \{p_0, p_1, \ldots, p_{l-1}\}$ be a set of l probes placed on a gadget G, and let G_P denote the tuple of probe values obtained during an execution of G. We say that P is simulatable by a set of input shares S if there exists a randomized simulator $SIM : \mathbb{F}_2^{|S|} \to \mathbb{F}_2^l$ such that, for any shared input of G, the distributions of G_P and $SIM(S)$ are identical, where the probability is taken over the random bits in G and SIM.

Definition 2 (d-Probe-Isolating Non-Interference [CS20])**.** Let G be a gadget and let P be a set of t_i probes placed on internal wires of G. Let A be a set of t_o output-share indices. The gadget G is said to satisfy d-PINI if, for every configuration of P and A such that $t_i + t_o \leq d$, there exists a set B of at most t_i share indices such that the probes on the wires set $P \cup Y_A$ can be simulated by the input shares $X_{A\cup B}$. Here, $X_{A\cup B}$ denotes all inputs of share indices in set $A \cup B$, and Y_A denotes all outputs of share indices in set A.

Definition 3 (d-Output-Probe-Isolating Non-Interference [CS21])**.** Let G be a gadget, and let P be a set of t_i probes placed on internal wires of G. Let A be a set of t_o output-share indices. The gadget G is said to satisfy d-OPINI if, for every configuration of P and A such that $t_i + t_o \leq d$, there exists a set B of at most t_i share indices such that the probes on the set wire $P \cup Y_A \cup Y_B$ can be simulated by the input shares $X_{A\cup B}$. Here, $X_{A\cup B}$ denotes all inputs of share indices in set $A \cup B$, Y_A denotes all outputs of share indices in set A, and Y_B denotes all outputs of share indices in set B.

2.4 Related Work

At CHES 2024, Cassiers et al. [CSV24] introduced the first low-latency OPINI gadget, called HPC4. This gadget realized a 2-input multiplication using only a single layer of registers. Owing to its elegant structure, HPC4 can be extended to arbitrary security orders, requiring $\frac{5d(d+1)}{2}$ random bits and $4d^2 + 5d + 1$ registers for security order d. Its first-order instance is shown in Equation (1). Specifically, to achieve OPINI security, a shared term is masked with two random bits inside the HPC4 gadget (e.g., the component function f_2, where the shared term b_1 is masked with r_0 and r_1 in Equation (1)). These two random bits produces redundant monomials (e.g., a_0r_0, a_0r_1) during the computation of cross-domain products (e.g., $\mathrm{w}_1\mathrm{w}_2$). These redundant monomials are eliminated by two extra redundant operations (e.g., f_3 and f_4 in Equation (1)).

In contrast to the HPC4 gadget, our work focuses on constructing low-latency, first-order OPINI gadgets applicable to arbitrary Boolean functions. In other words, we extend the capability of OPINI gadgets beyond two-input multiplication to general Boolean functions while preserving a single register layer.

$$
\begin{array}{llllll}
f_0(a_0,b_0) & = & a_0b_0 & \rightarrow & \mathrm{w}_0 & \\
f_1(a_0) & = & a_0 & \rightarrow & \mathrm{w}_1 & \\
f_2(b_1,r_0,r_1) & = & b_1+r_0+r_1 & \rightarrow & \mathrm{w}_2 & \\
f_3(a_0,r_0,r_2) & = & a_0r_0+r_2 & \rightarrow & \mathrm{w}_3 & \\
f_4(a_0,r_1,r_3) & = & a_0r_1+r_3 & \rightarrow & \mathrm{w}_4 & \mathrm{w}_1\mathrm{w}_2+\mathrm{w}_0+\mathrm{w}_3+\mathrm{w}_4=z_0 \\
\hline
f_5(a_1,b_1) & = & a_1b_1 & \rightarrow & \mathrm{w}_5 & \mathrm{w}_6\mathrm{w}_7+\mathrm{w}_5+\mathrm{w}_8+\mathrm{w}_9=z_1 \\
f_6(a_1) & = & a_1 & \rightarrow & \mathrm{w}_6 & \\
f_7(b_0,r_0,r_4) & = & b_0+r_0+r_4 & \rightarrow & \mathrm{w}_7 & \\
f_8(a_1,r_0,r_2) & = & a_1r_0+r_2 & \rightarrow & \mathrm{w}_8 & \\
f_9(a_1,r_3,r_4) & = & a_1r_4+r_3 & \rightarrow & \mathrm{w}_9 &
\end{array}
\tag{1}
$$

Independent and concurrent with our work, Rahimi and Moradi [RM26] proposed a low-latency first-order OPINI gadget (called OTSM), and extended it to arbitrary Boolean functions. In their construction, only the monomials derived from the second input shares are masked (two random bits per monomial). In contrast, our construction follows a different construction principle by masking all monomials derived from both the first and second input shares and introducing a randomness-reassignment technique that enables reusing randomness across the two output-share computations. Beyond our gadget, we also introduce a symbolic formal verification method and apply it to validate multiple low-latency OPINI gadgets, including OTSM.

3 New Techniques

This section presents two new techniques: (i) a method for constructing low-latency first-order OPINI-secure composable gadgets GOM, and (ii) a formal verification technique for validating the security of our gadgets. Section 3.1 presents a construction method for low-latency OPINI gadgets applicable to arbitrary Boolean functions. In this construction, each monomial derived from both the first and second input shares is masked using two random bits, and a randomness-reassignment technique is employed to further reduce the overall random bits without compromising security. To verify the security of GOM, Section 3.2 presents a symbolic formal verification technique whose key idea is several symbolic reduction rules. Leveraging these rules, the verification technique is applicable to multiple low-latency OPINI gadgets, including HPC4, GOM and OTSM.

3.1 Low-Latency First-Order OPINI Gadgets

To enhance clarity, Section 3.1.1 discusses the necessity of OPINI in a single-cycle iterative architecture. Section 3.1.2 then illustrates our technique through a first-order OPINI-secure AND3 gadget. Section 3.1.3 further extends our techniques to construct low-latency first-order OPINI gadgets for arbitrary Boolean functions, accompanied by security proofs.

3.1.1 The Necessity of First-Order OPINI in a Single-Cycle Iterative Architecture

In the single-cycle iterative architecture considered in this work, first-order OPINI is necessary to establish the security of the overall design. Although OPINI is closely related to PINI, it strengthens PINI by incorporating the additional simulability condition needed for trivial composition when a gadget is iteratively employed across consecutive clock cycles. The necessity of OPINI is developed from three perspectives, namely theoretical analysis, reported counterexamples, and practical countermeasures.

Theoretical Analysis. In a single-cycle iterative architecture, the shared outputs generated in one clock cycle are fed back as the shared inputs of the next cycle. The same combinational circuit is repeatedly evaluated on successive masked intermediate states, and

the security of the overall design should be analyzed across consecutive executions rather than within each execution in isolation. In this setting, it is necessary to consider not only the leakage caused by glitches, but also the effect of transitions, which capture register recombinations across adjacent clock cycles. PINI provides compositional guarantees in the presence of glitches, but does not account for the transition default. OPINI addresses this gap by incorporating the additional simulability condition required for trivial composition under both glitches and transitions. Without this notion, the security of iterative architecture has to be argued in an architecture-dependent and case-by-case manner. First-order OPINI therefore provides the appropriate formal foundation for the single-cycle iterative architecture considered in this work.

Reported Counterexamples. The necessity of first-order OPINI is further supported by previously reported leakage in iterative masked implementations. In particular, a single-cycle iterative implementation based on first-order HPC3 gadgets has been shown to exhibit first-order leakage [CSV24]. Similar first-order leakage has been reported for iterative implementations based on MCM gadgets [WFH+26] and TSM gadgets [RM26]. These results show that, although the underlying gadgets satisfy first-order PINI, this property alone does not prevent transition leakage when they are deployed in iterative architectures. Consequently, first-order PINI alone does not provide a generic security guarantee for iterative implementations, motivating the additional simulability condition formalized by first-order OPINI.

Practical Countermeasures. There are currently two approaches for achieving OPINI security. The first approach is to enhance an existing PINI gadget by appending an additional refresh layer, such as HPC3$^+$ [KM22] and MCM$^+$ [WFH+26]. By decorrelating the shared values processed in adjacent cycles, the refresh layer mitigates exploitable cross-cycle dependencies. However, it introduces an additional register layer and therefore cannot preserve single-cycle latency. The second approach is to construct new gadgets that inherently satisfy OPINI, such as HPC4 [CSV24]. In this approach, the required security condition is incorporated directly into the gadget construction. Although this approach is more challenging, it is more attractive for constructing single-cycle OPINI gadgets. This work adopts the second approach and proposes new low-latency first-order OPINI gadgets.

3.1.2 A First-Order OPINI Gadget for the AND3 Gate

The first-order OPINI gadget for the AND3 gate (i.e., $z = f(a, b, c) = abc$) is expressed using Boolean expressions, as shown in Equations (2) and (3). Equation (2) specifies the operations before the register layer (i.e., w_0–w_{15}), while Equation (3) describes the operations after the register layer. Specifically, the Boolean functions f_0–f_7 generate the first output share z_0, and are therefore said to operate in domain 0. Similarly, f_8–f_{15} are performed in domain 1. In domain 0, the monomials derived from the input shares with index 1 (i.e., b_1, c_1, b_1c_1) are masked using (r_0, r_1), (r_2, r_3), and (r_4, r_5), respectively. To ensure the overall correctness of the first-order OPINI-secure AND3 gadget, two extra redundant operations (i.e., f_6 and f_7) are needed in domain 0. Similarly, in domain 1, the monomials b_0, c_0, b_0c_0 are masked with (r_0, r_8), (r_2, r_9), (r_4, r_{10}), respectively, and the redundant operations f_{14} and f_{15} are needed for the output share z_1.

The above example illustrates the first technique used to construct the OPINI gadget, i.e., masking each monomial (i.e., $b_{0/1}, c_{0/1}, b_0c_0, b_1c_1$) with two random bits. To guarantee OPINI security, the above random bits (i.e., r_0, r_1, r_2, r_3, r_4, r_5, r_8, r_9, r_{10}) can be organized into two categories. The first category consists of the common random bits r_0, r_2, r_4, which are shared between the two circuits producing the two output shares. The second category contains distinct random bits (i.e., r_1, r_3, r_5, r_8, r_9, r_{10}), which are used independently to mask monomials. Noted that the idea of masking each monomial using two random bits is an extension of the HPC4 idea. In the HPC4 gadget, each share of one secret input is masked using two random bits (e.g., the share b_1 (resp., b_0) is masked

with r_0 and r_1 (resp., r_0 and r_4) in Equation (1)). To apply to the AND3 gadget, we extend the masked objective from input shares to monomials (e.g., the $b_{0/1}$, $c_{0/1}$, b_0c_0, b_1c_1). By extending the HPC4 idea, this work aims to construct OPINI-secure gadgets for the AND3 gate and even for arbitrary Boolean functions.

$$
\begin{array}{llll}
f_0(a_0) & = & a_0 & \rightarrow \ \mathrm{w}_0 \\
f_1(b_0) & = & b_0 & \rightarrow \ \mathrm{w}_1 \\
f_2(c_0) & = & c_0 & \rightarrow \ \mathrm{w}_2 \\
f_3(b_1, r_0, r_1) & = & b_1 + r_0 + r_1 & \rightarrow \ \mathrm{w}_3 \\
f_4(c_1, r_2, r_3) & = & c_1 + r_2 + r_3 & \rightarrow \ \mathrm{w}_4 \\
f_5(b_1, c_1, r_4, r_5) & = & b_1c_1 + r_4 + r_5 & \rightarrow \ \mathrm{w}_5 \\
f_6(a_0, b_0, c_0, r_0, r_2, r_4, r_6) & = & a_0b_0c_0 + a_0c_0r_0 + a_0b_0r_2 + a_0r_4 + r_6 & \rightarrow \ \mathrm{w}_6 \\
f_7(a_0, b_0, c_0, r_1, r_3, r_5, r_7) & = & a_0c_0r_1 + a_0b_0r_3 + a_0r_5 + r_7 & \rightarrow \ \mathrm{w}_7 \\
\hline
f_8(a_1) & = & a_1 & \rightarrow \ \mathrm{w}_8 \\
f_9(b_1) & = & b_1 & \rightarrow \ \mathrm{w}_9 \\
f_{10}(c_1) & = & c_1 & \rightarrow \ \mathrm{w}_{10} \\
f_{11}(b_0, r_0, r_8) & = & b_0 + r_0 + r_8 & \rightarrow \ \mathrm{w}_{11} \\
f_{12}(c_0, r_2, r_9) & = & c_0 + r_2 + r_9 & \rightarrow \ \mathrm{w}_{12} \\
f_{13}(b_0, c_0, r_4, r_{10}) & = & b_0c_0 + r_4 + r_{10} & \rightarrow \ \mathrm{w}_{13} \\
f_{14}(a_1, b_1, c_1, r_0, r_2, r_4, r_6) & = & a_1b_1c_1 + a_1c_1r_0 + a_1b_1r_2 + a_1r_4 + r_6 & \rightarrow \ \mathrm{w}_{14} \\
f_{15}(a_1, b_1, c_1, r_7, r_8, r_9, r_{10}) & = & a_1c_1r_8 + a_1b_1r_9 + a_1r_{10} + r_7 & \rightarrow \ \mathrm{w}_{15}
\end{array}
\tag{2}
$$

$$
\begin{array}{lll}
\mathrm{w}_0\mathrm{w}_1\mathrm{w}_4 + \mathrm{w}_0\mathrm{w}_3\mathrm{w}_2 + \mathrm{w}_0\mathrm{w}_5 + \mathrm{w}_6 + \mathrm{w}_7 & = & z_0 \\
\hline
\mathrm{w}_8\mathrm{w}_9\mathrm{w}_{12} + \mathrm{w}_8\mathrm{w}_{11}\mathrm{w}_{10} + \mathrm{w}_8\mathrm{w}_{13} + \mathrm{w}_{14} + \mathrm{w}_{15} & = & z_1
\end{array}
\tag{3}
$$

$$
\begin{array}{llll}
f_0(a_0) & = & a_0 & \rightarrow \ \mathrm{w}_0 \\
f_1(b_0) & = & b_0 & \rightarrow \ \mathrm{w}_1 \\
f_2(c_0) & = & c_0 & \rightarrow \ \mathrm{w}_2 \\
f_3(b_1, r_0, r_1) & = & b_1 + r_0 + r_1 & \rightarrow \ \mathrm{w}_3 \\
f_4(c_1, r_2, r_3) & = & c_1 + r_2 + r_3 & \rightarrow \ \mathrm{w}_4 \\
f_5(b_1, c_1, r_4, r_5) & = & b_1c_1 + r_4 + r_5 & \rightarrow \ \mathrm{w}_5 \\
f_6(a_0, b_0, c_0, r_0, r_2, r_4, r_6) & = & a_0b_0c_0 + a_0c_0r_0 + a_0b_0r_2 + a_0r_4 + r_6 & \rightarrow \ \mathrm{w}_6 \\
f_7(a_0, b_0, c_0, r_1, r_3, r_5, r_7) & = & a_0c_0r_1 + a_0b_0r_3 + a_0r_5 + r_7 & \rightarrow \ \mathrm{w}_7 \\
\hline
f_8(a_1) & = & a_1 & \rightarrow \ \mathrm{w}_8 \\
f_9(b_1) & = & b_1 & \rightarrow \ \mathrm{w}_9 \\
f_{10}(c_1) & = & c_1 & \rightarrow \ \mathrm{w}_{10} \\
f_{11}(b_0, r_1, r_2) & = & b_0 + r_1 + r_2 & \rightarrow \ \mathrm{w}_{11} \\
f_{12}(c_0, r_3, r_4) & = & c_0 + r_3 + r_4 & \rightarrow \ \mathrm{w}_{12} \\
f_{13}(b_0, c_0, r_0, r_5) & = & b_0c_0 + r_5 + r_0 & \rightarrow \ \mathrm{w}_{13} \\
f_{14}(a_1, b_1, c_1, r_1, r_3, r_5, r_6) & = & a_1b_1c_1 + a_1c_1r_1 + a_1b_1r_3 + a_1r_5 + r_6 & \rightarrow \ \mathrm{w}_{14} \\
f_{15}(a_1, b_1, c_1, r_0, r_2, r_4, r_7) & = & a_1c_1r_2 + a_1b_1r_4 + a_1r_0 + r_7 & \rightarrow \ \mathrm{w}_{15}
\end{array}
\tag{4}
$$

On the premise of achieving OPINI security[2], we further introduce an optimized gadget that employs a randomness-reassignment technique to reduce the number of random bits. In the optimized construction, the random bits required by the two output-share computations are reused, resulting in an AND3 gadget that requires only 8 random bits. The optimized scheme is shown in Equations (4) and (3). Compared with Equation (2), the main differences in Equation (4) lie in the operations of f_{11}–f_{13}. Specifically, we modify the ordering of random bits in f_3–f_5 to mask f_{11}–f_{13}. To elaborate, if the random bits masking f_3–f_5 are concatenated into a vector $R = r_0\|r_1\|r_2\|r_3\|r_4\|r_5$, then the random bits masking f_{11}–f_{13} are obtained by rotating R left by one position, yielding $R' = r_1\|r_2\|r_3\|r_4\|r_5\|r_0$.

[2]The security proof is omitted here and will be given in Section 3.1.3.

Consequently, the monomials b_0, c_0, and b_0c_0 are masked with (r_1, r_2), (r_3, r_4), and (r_5, r_0), respectively. Following this modification, the two redundant functions f_{14} and f_{15} are adjusted accordingly. The operations after the register layer remain unchanged and still use Equation (3).

After optimization, the low-latency first-order OPINI-secure AND3 gadget requires only 8 random bits and 16 registers. In the next subsection, we will demonstrate the generality of our techniques by extending them to arbitrary Boolean functions.

3.1.3 The First-Order OPINI Gadget for an Arbitrary Boolean Function

An arbitrary Boolean function can be represented in Algebraic Normal Form (ANF) as an XOR sum of multiple monomials. Since XOR operations can be trivially implemented within the OPINI masking framework, we apply our technique to construct OPINI-secure monomials. By integrating these monomials, we obtain a complete masking scheme for any Boolean function. We describe the construction guidelines, construction method, security proof, and the method of integrating into an arbitrary Boolean function as follows.

Construction Guidelines. To construct a low-latency, first-order OPINI gadget for an arbitrary monomial, we propose several key guidelines, outlined below.

- Low-Latency. This requires the masking scheme to contain only a single layer of registers. In addition, the critical path—defined as the maximum delay between the paths from the primary inputs to the registers and from the registers to the final outputs—should be as short as possible.
- Low-Randomness. To minimize randomness usage, we introduce a randomness-reassignment technique that allows the two computations producing output shares to reuse the same set of random bits. For example, in the AND3 gadget of Equation (4), the random bits $(r_0, r_1, \ldots, r_7)$ are shared between the computations of z_0 and z_1.
- Correctness. The XOR sum of all output shares equals the correct secret output. For the AND3 gadget $z = f(a, b, c) = abc$ in Equations (4) and (3), the two output shares are $z_0 = \mathrm{w}_0\mathrm{w}_1\mathrm{w}_4 + \mathrm{w}_0\mathrm{w}_3\mathrm{w}_2 + \mathrm{w}_0\mathrm{w}_5 + \mathrm{w}_6 + \mathrm{w}_7 = a_0bc + r_6 + r_7$ and $z_1 = \mathrm{w}_8\mathrm{w}_9\mathrm{w}_{12} + \mathrm{w}_8\mathrm{w}_{11}\mathrm{w}_{10} + \mathrm{w}_8\mathrm{w}_{13} + \mathrm{w}_{14} + \mathrm{w}_{15} = a_1bc + r_6 + r_7$. Their XOR sum satisfies $z = z_0 + z_1$, ensuring the correctness of the overall computation.
- Uniformity. This property requires that each output share be masked with at least one random bit. We ensure this by assigning one random bit to each redundant function. For example, in the AND3 gadget of Equation (4), we use r_6 for f_6 and r_7 for f_7.
- OPINI security. According to the definition of first-order OPINI, we consider two types of probing: internal probing (i.e., probing the inputs of registers) and output probing (i.e., probing the output shares). When a probe is placed at the input of a register, we can observe certain input shares with a single index—denoted as index B—along with some random bits. Based on this known information, all registers contributing to the output share with index B should be simulated using only input shares with index B, and some random bits. To satisfy this requirement, we introduce an additional dedicated random bit to remask each monomial, preventing the leakage of monomials involving input shares whose index is different from B. For example, in the AND3 gadget of Equation (4), a probe at the input of register w_6 reveals a_0, b_0, c_0, and r_0, r_2, r_4, r_6, i.e., some input shares with index 0 and some random bits. The registers w_0–w_7 generating z_0 can be simulated using the above known information along with the additional random bits r_1, r_3, r_5, r_7. When some probes are placed at output shares with a certain index, denoted as index A, this case should be simulated using input shares with index A and some random bits. To satisfy this requirement, each monomial derived from the input shares should be

masked with at least one distinct random bit. For instance, a probe at the output share z_0 reveals a_0, b_0, c_0, and r_0/r_1, r_2/r_3, r_4/r_5, r_6, r_7.

The above guidelines establish the core requirements for the low-latency OPINI gadget. Building upon them, we present the construction method of the first-order OPINI gadget for an arbitrary monomial in the following.

Construction Method. To clearly describe the method for constructing a first-order OPINI gadget for an arbitrary monomial, we present the construction steps in Algorithm 1.

Algorithm 1 The first-order OPINI gadget for a monomial of degree three or higher

Input: the target monomial $\prod_{i=0}^{m-1} x^i$ with $m > 2$,
input share $X_0 = x_0^{m-1}||x_0^{m-2}||\cdots||x_0^0$,
input share $X_1 = x_1^{m-1}||x_1^{m-2}||\cdots||x_1^0$,
a randomness vector R of length $2^m - 2$,
2-bit randomness r', r''.
Output: the circuit with output shares z_0, z_1, such that $z_0 + z_1 = \prod_{i=0}^{m-1} x^i$.

1: $\Omega \leftarrow \{\, U \subseteq \{1, 2, \ldots, m{-}1\} \mid U \neq \varnothing \,\}$
2: $\ell \leftarrow |\Omega|\ (= 2^{m-1} - 1)$ ▷ $\Omega \leftarrow \{U_t\}, 0 \le t \le \ell - 1$
3: $\left\langle r^{(0,t,1)},\ r^{(0,t,2)} \right\rangle_{0\le t\le \ell-1} \leftarrow R$
▷ $\left\langle r^{(0,t,1)},\ r^{(0,t,2)} \right\rangle_{0\le t\le \ell-1}$ cascades all 2-bit blocks $(r^{(0,t,1)}, r^{(0,t,2)})$.
4: $\left\langle r^{(1,t,1)},\ r^{(1,t,2)} \right\rangle_{0\le t\le \ell-1} \leftarrow \textsc{RotateLeftByOne}(R)$
▷ $\textsc{RotateLeftByOne}(R)$ denotes a one-bit cyclic left rotation of R.
5: $r^{(0,1)} = r^{(1,1)} \leftarrow r'$; $r^{(0,2)} = r^{(1,2)} \leftarrow r''$ ▷ Low-randomness
6: **for** $i = 0$ to 1 **do**
7: **for** $j = 0$ to $m{-}1$ **do**
8: $\mathcal{S}^{(i,j)} \leftarrow \mathrm{Reg}_{\mathrm{pipe}}[x_i^j]$ ▷ Low-Latency, OPINI security
9: **end for**
10: **for** $t = 0$ to $\ell - 1$ **do**
11: $V_t \leftarrow \{1, 2, \ldots, m{-}1\} \setminus U_t$
12: $\mathcal{P}^{(i,t)} \leftarrow \mathrm{Reg}[\,\prod_{u\in U_t} x_{1-i}^u + r^{(i,t,1)} + r^{(i,t,2)}\,]$ ▷ Low-Latency, OPINI security
13: $n^{(i,t,1)} \leftarrow x_i^0 \cdot \prod_{v\in V_t} x_i^v \cdot r^{(i,t,1)}$
14: $n^{(i,t,2)} \leftarrow x_i^0 \cdot \prod_{v\in V_t} x_i^v \cdot r^{(i,t,2)}$
15: **end for**
16: $\mathcal{N}^{(i,1)} \leftarrow \mathrm{Reg}[\,\prod_{j=0}^{m-1} x_i^j + \sum_{t=0}^{\ell-1} n^{(i,t,1)} + r^{(i,1)}\,]$
▷ Low-Latency, Uniformity, OPINI security
17: $\mathcal{N}^{(i,2)} \leftarrow \mathrm{Reg}[\,\sum_{t=0}^{\ell-1} n^{(i,t,2)} + r^{(i,2)}\,]$
18: **end for**
19: **for** $i = 0$ to 1 **do**
20: **for** $t = 0$ to $\ell - 1$ **do**
21: $V_t \leftarrow \{1, 2, \ldots, m{-}1\} \setminus U_t$
22: $w^{(i,t)} \leftarrow \mathcal{S}^{(i,0)} \cdot \prod_{v\in V_t} \mathcal{S}^{(i,v)} \cdot \mathcal{P}^{(i,t)}$
23: **end for**
24: $z_i = \sum_{t=0}^{\ell-1} w^{(i,t)} + \mathcal{N}^{(i,1)} + \mathcal{N}^{(i,2)}$ ▷ Correctness
25: **end for**

Algorithm 1 takes five inputs: the target monomial $\prod_{i=0}^{m-1} x^i$ with $m > 2$, input shares X_0 and X_1, a randomness vector R, and 2-bit randomness r' and r''. The algorithm outputs the desired circuit with output shares z_0 and z_1. The first two lines define the set Ω as the collection of all non-empty subsets of $\{1, 2, \ldots, m-1\}$. The element number of Ω is $\ell = 2^{m-1} - 1$. To reference its elements in the following, we take an enumeration $\{U_t\}$ of Ω and use t as the index, where $0 \le t \le \ell - 1$. Lines 3 to 5 specify the usage of randomness. Specifically, the randomness vector R is assigned to $r^{(0,t,1)}$ and $(r^{(0,t,2)})$, which are involved

in operations in domain 0, and used to mask the monomials derived from the input shares with index 1. The function RotateLeftByOne(R) denotes a one-position left rotation of R. The rotated result is assigned to $r^{(1,t,1)}$ and $(r^{(1,t,2)})$, which participate in operations in domain 1, and mask the monomials from the input shares with index 1. The random bit r' (resp., r'') is assigned to $r^{(0,1)}$ and $r^{(1,1)}$ (resp., $r^{(0,2)}$ and $r^{(1,2)}$), and is used to mask the redundant functions in domain 0 (resp., domain 1). Lines 6 to 8 synchronize each shared term by storing it into register $\mathcal{S}^{(i,j)}$. In lines 10 to 14, the symbol U_t represents the t-th element of the set Ω, while V_t denotes the complement of U_t. The monomials derived from the input shares with index $1-i$ are represented as $\prod_{u\in U_t} x^u_{1-i}$, which are masked with two random bits $r^{(i,t,1)}$ and $r^{(i,t,2)}$, respectively. The masked result is then stored into register $\mathcal{P}^{(i,t)}$. The variables $n^{(i,t,1)}$ and $n^{(i,t,2)}$ are auxiliary intermediate variables used to generate the following two redundant registers. In the computations of $n^{(i,t,1)}$ and $n^{(i,t,2)}$, we interpret $\prod_{v\in V_t} x^v_i$ as 1 when $V_t = \emptyset$. Lines 16 to 17 generate two redundant registers $\mathcal{N}^{(i,1)}$ and $\mathcal{N}^{(i,2)}$, whose stored values are already masked with $r^{(i,1)}$ and $r^{(i,2)}$, respectively. Lines 19 to 22 generate the required cross-domain product $w^{(i,t)}$. The output share z_i is obtained in line 24 by summing all cross-domain products and the two redundant registers.

The construction guidelines are also annotated in Algorithm 1. Lines 8, 12, 16 and 17 ensure the low-latency guideline, meaning that this scheme contains only a single layer of registers. The low-randomness guideline is achieved in lines 3 to 5, where one set of random bits is reused in the two circuits producing output shares z_0 and z_1. In lines 16, 17 and 24, two random bits $r^{(i,1)}$ and $r^{(i,2)}$, are involved in the output share z_i, thereby satisfying the uniformity. Lines 8, 12, 16 and 17 jointly guarantee the OPINI security. Line 24 eliminates the redundant monomials introduced during the computation of cross-domain product $w^{(i,t)}$ by performing XOR operations on $\mathcal{N}^{(i,1)}$ and $\mathcal{N}^{(i,2)}$. This ensures correctness, meaning the XOR sum of two output shares, z_0 and z_1, equals the target monomial.

As for the performance of an arbitrary masked monomial, the number of required random bits and registers is given below. According to line 12 of Algorithm 1, the number of U_t is $\ell = 2^{m-1} - 1$, which requires $2 \times (2^{m-1} - 1) = 2^m - 2$ random bits. In addition, two random bits r' and r'' are used in lines 3–5 of Algorithm 1. Therefore, for a masked monomial consisting of m terms ($m > 2$), a total of $2^m - 2 + 2 = 2^m$ random bits is required. Since the registers are used not only to store variables masked with the random bits described above, but also to store the shared terms in line 8 of Algorithm 1, the number of registers contributing to one output share is $2^{m-1} - 1 + 2 + m = 2^{m-1} + m + 1$. Therefore, the total number of registers required for the complete masking construction is $2 \times (2^{m-1} + m + 1) = 2^m + 2m + 2$.

Since Algorithm 1 is defined under the constraint $m > 2$, we also provide the $m = 2$ case for completeness, as shown in Equation (5). The main difference from Algorithm 1 is that the length of the randomness vector R is extended to three bits, $R = r_0 \| r_1 \| r_2$, instead of $2^2 - 2 = 2$ bits. The $m = 2$ case requires 5 random bits and 8 registers. Compare with the first-order instance of the HPC4 gadget, our gadget reduces two registers by merging the monomial a_0b_0 (resp., a_1b_1) into the operation f_2 (resp., f_6). This merging yields a 18% reduction in hardware area, as shown in Table 1.

$$
\begin{array}{llcll}
f_0(a_0) & = & a_0 & \to & \mathrm{w}_0 & \\
f_1(b_1, r_0, r_1) & = & b_1 + r_0 + r_1 & \to & \mathrm{w}_1 & \\
f_2(a_0, b_0, r_0, r_3) & = & a_0b_0 + a_0r_0 + r_3 & \to & \mathrm{w}_2 & \\
f_3(a_0, r_1, r_4) & = & a_0r_1 + r_4 & \to & \mathrm{w}_3 & \mathrm{w}_0\mathrm{w}_1 + \mathrm{w}_2 + \mathrm{w}_3 = z_0 \\
\hline
f_4(a_1) & = & a_1 & \to & \mathrm{w}_4 & \mathrm{w}_4\mathrm{w}_5 + \mathrm{w}_6 + \mathrm{w}_7 = z_1 \\
f_5(b_0, r_1, r_2) & = & b_0 + r_1 + r_2 & \to & \mathrm{w}_5 & \\
f_6(a_1, b_1, r_1, r_3) & = & a_1b_1 + a_1r_1 + r_3 & \to & \mathrm{w}_6 & \\
f_7(a_1, r_2, r_4) & = & a_1r_2 + r_4 & \to & \mathrm{w}_7 &
\end{array}
\tag{5}
$$

Security Proof. For any masked monomial generated by Algorithm 1, the OPINI

security proof is presented below.

Theorem 1. *Assuming that all randomness bits involved in Algorithm 1 are statistically independent of each input share, then the masked monomial generated by Algorithm 1 is iterated glitch+transition-robust 1-OPINI.*

Proof.

i. When a probe is placed at the input of $\mathcal{S}^{(i,j)}$, it observes the variable x_i^j, which can be directly simulated by x_i^j. There are four types of registers contributing to the output share z_i, namely $\mathcal{S}^{(i,j)}$, $\mathcal{P}^{(i,t)}$, $\mathcal{N}^{(i,1)}$, and $\mathcal{N}^{(i,2)}$. Since the registers $\mathcal{P}^{(i,t)}$, $\mathcal{N}^{(i,1)}$, and $\mathcal{N}^{(i,2)}$ are all masked with independent random bits, they can be simulated using $r^{(i,t,1)}/r^{(i,t,2)}$, $r^{(i,1)}$, and $r^{(i,2)}$, respectively. Therefore, the input of $\mathcal{S}^{(i,j)}$ and z_i can be jointly simulated with x_i^j, $r^{(i,t,1)}/r^{(i,t,2)}$, $r^{(i,1)}$, and $r^{(i,2)}$, which is consistent with the OPINI security.

ii. A probe at the input of $\mathcal{P}^{(i,t)}$ directly reveals variables $\{x_{1-i}^u \mid u \in U_t\}$ together with $r^{(i,t,1)}$ and $r^{(i,t,2)}$. We analyze the four types of registers contributing to z_{1-i}, namely $\mathcal{S}^{(1-i,j)}$, $\mathcal{P}^{(1-i,t)}$, $\mathcal{N}^{(1-i,1)}$, and $\mathcal{N}^{(1-i,2)}$. The value stored in $\mathcal{S}^{(1-i,j)}$ can be simulated using x_{1-i}^j. By the randomness-reassignment technique, the two known random bits $r^{(i,t,1)}$ and $r^{(i,t,2)}$ are reused in two different instances of $\mathcal{P}^{(1-i,t)}$, denoted as $\mathcal{P}^{(1-i,t_\alpha)}$ and $\mathcal{P}^{(1-i,t_\beta)}$, such that $r^{(1-i,t_\alpha,2)} = r^{(i,t,1)}$ and $r^{(1-i,t_\beta,1)} = r^{(i,t,2)}$. Since these two instances still contains unknown random bits, namely $r^{(1-i,t_\alpha,1)}$ and $r^{(1-i,t_\beta,2)}$, the values stored in $\mathcal{P}^{(1-i,t_\alpha)}$ and $\mathcal{P}^{(1-i,t_\beta)}$ can be simulated using these unknown random bits, respectively. For all other instances $\mathcal{P}^{(1-i,t)}$ with $t \notin \{t_\alpha, t_\beta\}$, which do not involve $r^{(i,t,1)}$ and $r^{(i,t,2)}$, simulation can be performed using $r^{(1-i,t,1)}/r^{(1-i,t,2)}$. The values stored in $\mathcal{N}^{(1-i,1)}$ and $\mathcal{N}^{(1-i,2)}$ are masked with two unknown random bits $r^{(1-i,1)}$ and $r^{(1-i,2)}$, respectively. They can be simulated using these two random bits, respectively. Therefore, this case can be jointly simulated with multiple x_{1-i}^u, $r^{(i,t,1)}$, $r^{(i,t,2)}$, x_{1-i}^j, $r^{(1-i,t_\alpha,1)}$, $r^{(1-i,t_\beta,2)}$, $r^{(1-i,t,1)}/r^{(1-i,t,2)}$ (for $t \notin \{t_\alpha, t_\beta\}$), $r^{(1-i,1)}$, and $r^{(1-i,2)}$, which is consistent with the OPINI security.

iii. When a probe at the input of $\mathcal{N}^{(i,1)}$ directly reveals multiple x_i^j, $n^{(i,t,1)}$ and $r^{(i,1)}$. Since $n^{(i,t,1)}$ is the product of x_i^0, $\prod_{v\in V} x_i^v$ and $r^{(i,t,1)}$, it can be simulated using multiple $i-th$ shares of secret variables x and random bit $r^{(i,t,1)}$. These shares correspond to the output share z_i. Referring to case i, there are four types of registers contributing to z_i. The value of $\mathcal{S}^{(i,j)}$ can be simulated with x_i^j. Since the value stored in $\mathcal{P}^{i,t}$ is masked with two random bits, where $r^{(i,t,2)}$ remains unknown, it can be simulated using $r^{(i,t,2)}$. Since the value stored in $\mathcal{N}^{i,2}$ is masked with $r^{(i,2)}$, it can be simulated using $r^{(i,2)}$. Therefore, this case can be simulated with multiple input shares with index i, multiple $r^{(i,t,1)}$, $r^{(i,1)}$, x_i^j, $r^{(i,t,2)}$, and $r^{(i,2)}$.

iv. A probe at the input of $\mathcal{N}^{(i,2)}$ directly reveals multiple x_i^j, $n^{(i,t,2)}$ and $r^{(i,2)}$. It is similar to case iii. This case can be simulated using multiple input shares with index i, multiple $r^{(i,t,2)}$, $r^{(i,2)}$, x_i^j, $r^{(i,t,1)}$, and $r^{(i,1)}$.

v. When a probe is placed at the output share z_i, it observes several registers, including $\mathcal{S}^{(i,0)}$, $\mathcal{S}^{(i,v)}$, $\mathcal{P}^{(i,t)}$, $\mathcal{N}^{(i,1)}$, and $\mathcal{N}^{(i,2)}$. Since $\mathcal{P}^{(i,t)}$, $\mathcal{N}^{(i,1)}$, and $\mathcal{N}^{(i,2)}$ are masked with the random bits $r^{(i,t,1)}$ and $r^{(i,t,2)}$, $r^{(i,1)}$, and $r^{(i,2)}$, respectively, these variables can be simulated using these random bits. The values of $\mathcal{S}^{(i,0)}$ and $\mathcal{S}^{(i,v)}$ can be simulated using x_i^0 and multiple x_i^j. Therefore, this case can be jointly simulated using x_i^0, multiple x_i^j, and the above random bits, which is in line with OPINI security.

□

Integration into an Arbitrary Boolean Function. To generate first-order, low-latency OPINI masking for an arbitrary Boolean function, we describe a three-step integration involving generating all GOM-masked monomials appearing in the ANF of the target function, generating the combined redundant operations, and generating the combined output shares. For clarity, we exemplify the integration using the Boolean function $f(a,b,c,d) = ab + ad + abc$, corresponding to Equations (6) and (7). We also provide the 1-OPINI security proof for the integrated Boolean function in the following.

$$
\begin{array}{lcll}
f_0(a_0) & = & a_0 & \rightarrow \ \mathrm{w}_0 \\
f_1(b_0) & = & b_0 & \rightarrow \ \mathrm{w}_1 \\
f_2(c_0) & = & c_0 & \rightarrow \ \mathrm{w}_2 \\
f_3(b_1, r_0, r_1) & = & b_1 + r_0 + r_1 & \rightarrow \ \mathrm{w}_3 \\
f_4(c_1, r_2, r_3) & = & c_1 + r_2 + r_3 & \rightarrow \ \mathrm{w}_4 \\
f_5(d_1, r_4, r_5) & = & d_1 + r_4 + r_5 & \rightarrow \ \mathrm{w}_5 \\
f_6(b_1, c_1, r_6, r_7) & = & b_1c_1 + r_6 + r_7 & \rightarrow \ \mathrm{w}_6 \\
f_7(a_0, b_0, c_0, d_0, r_0, r_2, r_4, r_6, r_8) & = & a_0b_0 + a_0d_0 + a_0b_0c_0 + a_0r_0 + a_0r_4 + a_0b_0r_2 + a_0c_0r_0 + a_0r_6 + r_8 & \rightarrow \ \mathrm{w}_7 \\
f_8(a_0, b_0, c_0, d_0, r_1, r_3, r_5, r_7, r_9) & = & a_0r_1 + a_0r_5 + a_0b_0r_3 + a_0c_0r_1 + a_0r_7 + r_9 & \rightarrow \ \mathrm{w}_8 \\
\hline
f_9(a_1) & = & a_1 & \rightarrow \ \mathrm{w}_9 \\
f_{10}(b_1) & = & b_1 & \rightarrow \ \mathrm{w}_{10} \\
f_{11}(c_1) & = & c_1 & \rightarrow \ \mathrm{w}_{11} \\
f_{12}(b_0, r_1, r_2) & = & b_0 + r_1 + r_2 & \rightarrow \ \mathrm{w}_{12} \\
f_{13}(c_0, r_3, r_4) & = & c_0 + r_3 + r_4 & \rightarrow \ \mathrm{w}_{13} \\
f_{14}(d_0, r_5, r_6) & = & d_0 + r_5 + r_6 & \rightarrow \ \mathrm{w}_{14} \\
f_{15}(b_0, c_0, r_0, r_7) & = & b_0c_0 + r_7 + r_0 & \rightarrow \ \mathrm{w}_{15} \\
f_{16}(a_1, b_1, c_1, d_1, r_1, r_3, r_5, r_7, r_8) & = & a_1b_1 + a_1d_1 + a_1b_1c_1 + a_1r_1 + a_1r_5 + a_1b_1r_3 + a_1c_1r_1 + a_1r_7 + r_8 & \rightarrow \ \mathrm{w}_{16} \\
f_{17}(a_1, b_1, c_1, d_1, r_0, r_2, r_4, r_6, r_8, r_9) & = & a_1r_2 + a_1r_6 + a_1b_1r_4 + a_1c_1r_2 + a_1r_0 + r_9 & \rightarrow \ \mathrm{w}_{17}
\end{array}
\tag{6}
$$

$$
\begin{array}{lcl}
\mathrm{w}_0\mathrm{w}_3 + \mathrm{w}_0\mathrm{w}_5 + \mathrm{w}_0\mathrm{w}_1\mathrm{w}_4 + \mathrm{w}_0\mathrm{w}_2\mathrm{w}_3 + \mathrm{w}_0\mathrm{w}_6 + \mathrm{w}_7 + \mathrm{w}_8 & = & z_0 \\
\hline
\mathrm{w}_9\mathrm{w}_{12} + \mathrm{w}_9\mathrm{w}_{14} + \mathrm{w}_9\mathrm{w}_{10}\mathrm{w}_{13} + \mathrm{w}_9\mathrm{w}_{11}\mathrm{w}_{12} + \mathrm{w}_9\mathrm{w}_{15} + \mathrm{w}_{16} + \mathrm{w}_{17} & = & z_1
\end{array}
\tag{7}
$$

The integration proceeds in three steps. (i) For an arbitrary Boolean function, we first construct GOM-masked monomials using Algorithm 1 for $m > 2$ (e.g., abc) and Equation (5) for $m = 2$ (e.g., ab, ad). This yields multiple registers $\mathcal{S}^{(i,j)}$ that store the input shares, $\mathcal{P}^{(i,t)}$ that store the monomials derived from the input shares. For the example function, these registers correspond to w_0–w_6 and w_9–w_{15} in Equation (6). Notably, the random bits used to construct multiple GOM-masked monomials are reused, particularly in the computation of the registers $\mathcal{P}^{(i,t)}$. This reuse strategy is similar to the randomness-reassignment technique described in Section 3.1.3. Specifically, the random bits used by multiple $\mathcal{P}^{(0,t)}$ contributing to the first output share of the target function form a randomness vector R, while those used by multiple $\mathcal{P}^{(1,t)}$ contributing to the second output share are obtained by a one-position left rotation of R, denoted by $\textsc{RotateLeftByOne}(R)$. For the example function in Equation (6), the random bits used in w_3–w_6 are concatenated to form the vector $R = r_0 \| r_1 \| \ldots \| r_6 \| r_7$ while those used in w_{12}–w_{15} form its one-position left rotation, denoted by $\textsc{RotateLeftByOne}(R) = r_1 \| r_2 \| \ldots \| r_7 \| r_0$. (ii) Since the redundant operations generated in the lines 16–17 of Algorithm 1 and Equation (5) are XORed into the output shares (line 24), and the ANF itself is also an XOR-sum of monomials, they can be merged into two combined redundant functions and masked with distinct random bits. In the example, this corresponds to w_7–w_8 and w_{16}–w_{17} in Equation (6). (iii) Since these cross-domain products contribute to the final output shares through XOR operations (the terms $w^{(i,t)}$ in line 24 of Algorithm 1), they are merged to

generate the final output shares z_0 and z_1. This corresponds to z_0 and z_1 in Equation (7). For the purpose of the security proof, we temporarily classify the registers introduced in the integration as input-share registers $\mathcal{S}^{(i,j)}$, refreshed monomial registers $\mathcal{P}^{(i,t)}$, and combined redundant registers $\mathcal{C}^{(i,1)}$ and $\mathcal{C}^{(i,2)}$. In the example Boolean function of Equations (6) and (7), these four types of registers correspond to the register w_0–w_2 and w_9–w_{11}, w_3–w_6 and w_{12}–w_{15}, w_7 and w_{16}, and w_8 and w_{17}, respectively.

Theorem 2. *Assuming that all randomness bits are statistically independent of the input shares, then the masked Boolean function obtained by the above three-step integration is iterated glitch+transition-robust 1-OPINI.*

Proof.

i. A probe placed at the input of an input-share register $\mathcal{S}^{(i,j)}$ can be simulated by the input share x_i^j. For the output share z_i, the contributing registers are $\mathcal{S}^{(i,j)}$, $\mathcal{P}^{(i,t)}$, $\mathcal{C}^{(i,1)}$, and $\mathcal{C}^{(i,2)}$. The registers $\mathcal{P}^{(i,t)}$, $\mathcal{C}^{(i,1)}$, and $\mathcal{C}^{(i,2)}$ are masked by independent random bits and can therefore be simulated by the corresponding randomness. Hence, this case is simulated using input shares with index i and the above random bits, which is consistent with 1-OPINI security.

ii. A probe placed at the input of a refreshed monomial register $\mathcal{P}^{(i,t)}$ directly reveals the variables $\{x_{1-i}^u \mid u \in U_t\}$ together with the random bits $r^{(i,t,1)}$ and $r^{(i,t,2)}$. We consider the registers contributing to the output share z_{1-i}, namely $\mathcal{S}^{(1-i,j)}$, $\mathcal{P}^{(1-i,t)}$, $\mathcal{C}^{(1-i,1)}$, and $\mathcal{C}^{(1-i,2)}$. The input-share registers $\mathcal{S}^{(1-i,j)}$ are simulated using the input shares x_{1-i}^j. By the randomness-reuse strategy, the known bits $r^{(i,t,1)}$ and $r^{(i,t,2)}$ are reused in two distinct instances of $\mathcal{P}^{(1-i,t)}$, denoted by $\mathcal{P}^{(1-i,t_\alpha)}$ and $\mathcal{P}^{(1-i,t_\beta)}$, where $r^{(1-i,t_\alpha,2)} = r^{(i,t,1)}$ and $r^{(1-i,t_\beta,1)} = r^{(i,t,2)}$. Since these two instances still contain the unknown bits $r^{(1-i,t_\alpha,1)}$ and $r^{(1-i,t_\beta,2)}$, respectively, they can be simulated using these unknown bits. For any other $t \notin \{t_\alpha, t_\beta\}$, the instance $\mathcal{P}^{(1-i,t)}$ is independent of the known bits and is simulated using the random bits $r^{(1-i,t,1)}/r^{(1-i,t,2)}$. Finally, $\mathcal{C}^{(1-i,1)}$ and $\mathcal{C}^{(1-i,2)}$ are masked by distinct random bits $r^{(1-i,1)}$ and $r^{(1-i,2)}$, respectively, and are simulated by these bits. Hence, this case can be simulated using input shares indexed by $1-i$ and the above random bits, which satisfy the 1-OPINI requirement.

iii. A probe placed at the input of $\mathcal{C}^{(i,1)}$ directly reveals input shares with index i, multiple random bits $r^{(i,t,1)}$, and $r^{(i,1)}$. The registers contributing to z_i are $\mathcal{S}^{(i,j)}$, $\mathcal{P}^{(i,t)}$, $\mathcal{C}^{(i,1)}$, and $\mathcal{C}^{(i,2)}$. The registers $\mathcal{P}^{(i,t)}$ still contain the unknown bits $r^{(i,t,2)}$, and $\mathcal{C}^{(i,2)}$ is masked by $r^{(i,2)}$. Hence, this case can be simulated using input shares with index i, multiple $r^{(i,t,1)}$, $r^{(i,1)}$, $r^{(i,t,2)}$ and $r^{(i,2)}$

iv. A probe placed at the input of $\mathcal{C}^{(i,2)}$ is similar to case iii. It directly reveals input shares with index i, multiple random bits $r^{(i,t,2)}$, and $r^{(i,2)}$. The remaining contributing registers are simulated using multiple random bits $r^{(i,t,1)}$ and $r^{(i,1)}$. Hence, this case can be simulated using input shares with index i and the above random bits.

v. When a probe is placed at the output share z_i, it observes registers $\mathcal{S}^{(i,j)}$, $\mathcal{P}^{(i,t)}$, $\mathcal{C}^{(i,1)}$ and $\mathcal{C}^{(i,2)}$. By the analyses of the preceding cases, these registers can be simulated using input shares with index i, multiple $r^{(i,t,1)}/r^{(i,t,2)}$, $r^{(i,1)}$ and $r^{(i,2)}$.

□

For an n-bit Boolean function of algebraic degree $n-1$, the number of required random bits and registers is analyzed as follows. At step (i) of the integration process, at most $2 \times \{\binom{n-1}{1} + \binom{n-1}{2} + \ldots + \binom{n-1}{n-2}\} = 2^n - 4$ monomials need to be remasked, which requires $2 \times (2^{n-1} - 2) = 2^n - 4$ bits of random. Together with two additional random bits used for masking redundant functions, the total randomness requirement is $2^n - 2$ bits. In addition

to storing variables masked by the above randomness, we also use registers to store at most $2n$ shared terms generated in step (i) of the integration. Therefore, the total number of registers is $2 \times (2^{n-1} - 2 + 2 + n) = 2^n + 2n$.

This construction can be naturally extended from Boolean functions to vectorial Boolean functions by reusing the registers $\mathcal{S}^{(i,j)}$ and $\mathcal{P}^{(i,t)}$. For an n-dimensional vectorial Boolean function, only an additional $2(n-1)$-bit randomness is required to mask the redundant functions of the remaining coordinate functions. Consequently, for a $\mathbb{F}_2^n \to \mathbb{F}_2^n$ vectorial Boolean function of algebraic degree $n-1$, our scheme requires at most $2^n + 2n - 4$ bits of random and $2^n + 6n - 4$ registers.

3.2 Formal Verification for Selected Gadgets

To the best of our knowledge, there is no public formal verification tool for OPINI gadgets. For example, SILVER [KSM20] and VERICA [RFSG22] can verify several notions, such as SNI [BBD+16] and PINI, but do not cover OPINI. To verify the OPINI security of our gadgets, we introduce a symbolic verification technique. Rather than serving as a general formal verification tool for arbitrary circuits, our technique is designed as a lightweight module that can validate OPINI security for a specific type of circuits, which contain a single layer of registers.

The workflow of our formal verification module is shown in Figure 1. The input is a 1-register-layer circuit, consisting of a single register layer, a pre-register combinational circuit generating intermediate signals $\mathcal{F} = \{f_0, f_1, \ldots\}$, and a post-register combinational circuit producing final outputs $\mathcal{G} = \{g_0, g_1, \ldots\}$. This circuit can be parsed from a hardware netlist. The final output of the formal verification module is the security verdict result or a counterexample. If and only if, for every probe configuration, the corresponding simulation information set SIM satisfies the OPINI definition, the output is an OPINI-secure result. Otherwise, the module outputs a probe configuration for which the OPINI check fails.

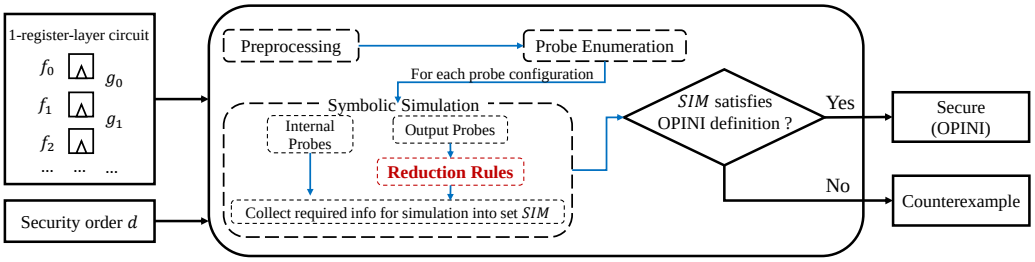


Figure 1: The operation process of our formal verification module.

Preprocessing. For each first-stage function $f_i \in \mathcal{F}$, we extract the set of its input variables, denoted by $vars_f[i]$, and the set $r_var[i]$ of linear random variables, i.e., random variables that appear in degree-1 monomials and do not appear in any high-algebraic-degree (≥ 2) monomials. Similarly, for each output function $g_j \in \mathcal{G}$, we derive a set $vars_g[j]$ that stores several f_i that contribute to the corresponding output function g_j. Taking the AND3 gadget in Equations (4) and (3) as an example, the first-stage functions are $\mathcal{F} = \{f_0, f_1, \ldots, f_{15}\}$ and the second-stage functions are $\mathcal{G} = \{z_0, z_1\}$. For the function f_3, we obtain $vars_f[3] = \{b_1, r_0, r_1\}$ and $r_var[3] = \{r_0, r_1\}$. For the output function z_0, we have $vars_g[0] = \{f_0, f_1, \ldots, f_7\}$.

Probe Enumeration. In this phase, we enumerate only the most powerful internal probing locations (i.e., the outputs of the first-stage functions $\mathcal{F}$) and collect them in the set I. We also enumerate the output-share indices and collect them in the set A. The total probing locations satisfy $t_i + t_o \leq d$, where t_i is the number of internal probes, t_o is the number of output-share indices and d is the given security order. For example, in a first-order AND3 gadget, one probe may be placed on any output of the combinational

functions f_k for $0 \leq k \leq 15$, or on output-share index 0 or 1, corresponding to the position z_0 or z_1, respectively.

Symbolic Simulation. For each probe configuration, we initialize two sets SIM and B to store variables for simulation and their corresponding share indices, respectively. Probes placed on the internal locations of the target circuit can be simulated using the input variables of corresponding first-stage functions in $\mathcal{F}$. These variables are then added to the set SIM, and their corresponding share indices are added to the set B. As an illustration, when a probe is placed on the output of the internal function f_3, all of its input variables $vars_f[3] = \{b_1, r_0, r_1\}$ are added to SIM and the share index $\{1\}$ is added to B. In addition to the simulation of internal probes, output shares with indices in the set $A \cup B$ should also be simulated. These variables required for this simulation are derived and added to SIM using several symbolic reduction rules. These rules are described in the following phase, called reduction rules.

Security Validation. For each probe configuration, we require that the share indices in B satisfy $|B| \leq t_i$. We then derive, from the simulated information set SIM, a set C of share indices and check that $C \subseteq A \cup B$ and the auxiliary set TBD should be empty. If all these conditions hold for every probe configuration, the circuit satisfies OPINI at order d. Otherwise, we output a counterexample configuration for which the OPINI check fails.

Reduction Rules. The reduction rules are proposed to address the following problem: deriving the necessary information to simulate output shares with indices in $A \cup B$. The specific method of simulating output probes via reduction rules is described in Algorithm 2. We first identify a set Q, consisting of output functions g_j that contribute to output shares with indices in $A \cup B$. This problem is converted to simulating some probes placed on the corresponding output functions $g_i \in Q$. Such output probes can be simulated using the input variables of the functions g_j. Since these input variables are several values stored in registers (i.e, the output values of several functions f_i), this problem is further converted to deriving the information required to simulate the output values of these f_i[3]. To deal with these output values, we collect the corresponding functions f_i into an auxiliary set TBD. The task of reduction rules is to derive necessary information to simulate the outputs of all functions in TBD. To clearly illustrate these rules, we provide detailed descriptions and examples for each rule, together with Algorithm 2, which demonstrates their application.

- Reduction Rule 1 (Perfect Mask). This rule, originally introduced in [ZCF25], plays a particularly elegant role in our symbolic reduction. In our setting, if a certain function f_i from the set TBD contains a linear random variable $r \in r_var[i]$ that does not appear in the input-variable set of any other unresolved functions in TBD, and r is not in the set SIM, then f_i can be simulated directly using this unique random variable r. Consequently, r is added to SIM and the index i is removed from TBD. This rule corresponds to lines 9 to 22 of Algorithm 2. The formal description is given below:

$$\exists i \in TBD, \exists r \in r_var[i] : \left(r \notin \bigcup_{\substack{j \in TBD \\ j \neq i}} vars_f[j] \ \wedge \ r \notin SIM\right) \\ \implies f_i \text{ is simulatable using the unique random bit } r. \tag{8}$$

  Example. Consider the AND3 gadget in Equations (4) and (3). When a probe is placed at the output share z_0, the unresolved set is $TBD = \{0, 1, \ldots, 7\}$. Since the linear random variable r_7 appears only in f_7 and in none of the other functions indexed by TBD, Rule 1 applies and yields $SIM \leftarrow SIM \cup \{r_7\}, TBD \leftarrow TBD \backslash \{7\}$. Repeating this reduction, the linear random variables $r_6, r_4/r_5, r_2/r_3$, and r_0/r_1 are sequentially added to SIM, and the corresponding indices $6, 5, 4, 3$ are removed from

[3]Since the output values of f_i are stored in registers, we can only observe their output values, not their input variables.

TBD. After exhausting Rule 1, the information for simulation and unresolved sets are $SIM = \{r_7, r_6, r_4/r_5, r_2/r_3, r_0/r_1\}, TBD = \{0, 1, 2\}$.

- Reduction Rule 2 (Multi-function Simulation). This rule enables jointly simulating multiple unresolved functions in TBD via XOR combination. For any subset $U \subseteq TBD$ whose size is permitted by the set $size$, we consider the XOR-combined function $f' \leftarrow \bigoplus_{i \in U} f_i$. If all linear random variables stored in the set r_var_f' are already in SIM, then every $f_i \in U$ can be simulated using its input variables. Consequently, we add these input variables to SIM and the index $i \in U$ is removed from the set TBD. This rule corresponds to lines 23 to 37 of Algorithm 2. The formal characterization is given below:

$$\begin{aligned} &\exists U \subseteq TBD, |U| \in size : \ \big(r_var_f' \subseteq SIM\big) \\ &\qquad \Longrightarrow \ \forall\, i \in U, \ f_i \text{ is simulatable using all input variables of } f_i. \end{aligned} \tag{9}$$

Example. For the AND3 gadget after Reduction Rule 1, when the subset size is $|U| = 1$, we sequentially examine the indices $i \in \{0, 1, 2\}$, that is, all elements of TBD. For each such index, the corresponding XOR-combined function f' contains no linear random variables, i.e., $r_var_f' \leftarrow \emptyset$, so the condition $r_var_f' \subseteq SIM$ hold trivially. Therefore, The functions f_0, f_1, and f_2 are all simulatable. We add their input variables to SIM and remove the corresponding indices from TBD. At this point, the simulation information set becomes $SIM = \{r_7, r_6, r_4/r_5, r_2/r_3, r_0/r_1, a_0, b_0, c_0\}$ and the unsolved set is $TBD = \emptyset$. The allowed subset size $|U| = 2$ is mainly considered for high-order gadgets, where some random bits are typically reused in an antisymmetric form, for example the states $r''_{ij} = -r''_{ji}$ and $r'''_{ij} = -r'''_{ji}$ in the HPC4 gadget (Algorithm 3 in [CSV24]).

- Reduction Rule 3 (Simulation Echo). This rule acts as a feedback mechanism for expanding the allowed subset sizes used in Reduction Rules 1 and 2. After Rules 1 and 2 saturate under the current size set $size = \{1, 2\}$, the unresolved set TBD may still contain multiple functions. Some of these functions may be jointly simulatable via XOR combination for subset sizes that are not yet permitted. To capture such cases, for each non-empty subset $U \subseteq TBD$ with $|U| \notin size$, we construct the XOR-combined function f'. If all linear random variables stored in the set r_var_f' are already in SIM, then the subset size $|U|$ is added to $size$. Rules 1 and 2 are then rerun using the updated $size$. This simulation echo continues until no new subset size can be activated. Rule 3 corresponds to lines 38 to 46 of Algorithm 2. Its formal characterization is given below:

$$\begin{aligned} &\exists U \subseteq TBD, \ |U| \notin size : r_var_f' \subseteq SIM \\ &\qquad \Longrightarrow \ size \leftarrow size \cup \{|U|\}. \end{aligned} \tag{10}$$

Example. Since all indices in TBD have already been resolved by the previous rules in the AND3 case, the rule terminates immediately. Note that Rule 3 serves as an extension mechanism. When the subset size $|U|$ required in Rule 2 is neither 1 nor 2, Rule 3 can activate additional subset sizes and may be relevant for more complex, higher-order masked circuits.

Leveraging three symbolic reduction rules, the proposed verification module can handle complex circuits, like the AES S-box. This addresses a key challenge of general-purpose formal verification tools such as SILVER [KSM20] and VERICA [RFSG22], which often require huge computational and memory costs on large circuits. Our verification module is narrower in scope than general-purpose tools and applies to existing single-cycle OPINI gadgets (e.g., HPC4, GOM, and OTSM).

Algorithm 2 Simulating Output Probes via Reduction Rules

Input: the target circuit consisting sequentially of combinatinal functions $\mathcal{F} = \{f_i\}$, a layer of registers, combinational functions $\mathcal{G} = \{g_j\}$,
some information from the preprocessing phase: $\{r_var[i]\}$, $\{vars_f[i]\}$, $\{vars_g[j]\}$,
the simulated information set SIM,
two share indices set A, B,

Output: the simulated information set SIM and auxiliary set TBD.

$TBD \leftarrow \emptyset$
$Q \leftarrow \{\, g_j \in \mathcal{G} \mid g_j$ contributes to an output share with index in $A \cup B \,\}$
for each q in Q **do**
  $TBD \leftarrow TBD \cup vars_g[q]$
end for
$size \leftarrow \{1, 2\}$
while true **do**
  $updated1 \leftarrow false$, $updated2 \leftarrow false$, $updated3 \leftarrow false$
  while true **do**
    $flag \leftarrow false$
    for each index i in TBD **do**
      $U \leftarrow \bigcup_{j \in TBD,\ j \neq i} vars_f[j]$
      find $r \in r_var[i]$ such that $r \notin U$ **and** $r \notin SIM$
      if such an r exists **then** ▷ Reduction Rule 1
        $SIM \leftarrow SIM \cup \{r\}$, $TBD \leftarrow TBD \setminus \{i\}$
        $updated1 \leftarrow true$, $flag \leftarrow true$
      end if
    end for
    if $flag = false$ **then**
      break
    end if
  end while
  while true **do**
    $flag \leftarrow false$
    for each subset $U \subseteq TBD$ with $|U| \in size$ **do**
      $f' \leftarrow \bigoplus_{i \in U} f_i$
      $r_var_f' \leftarrow$ (linear "r"-vars of f') $\setminus$ (higher-degree vars of f')
      if $r_var_f' \subseteq SIM$ **then** ▷ Reduction Rule 2
        $vars_a \leftarrow$ all input variables of $f_i, i \in U$
        $SIM \leftarrow SIM \cup vars_a$, $TBD \leftarrow TBD \setminus U$
        $updated2 \leftarrow true$, $flag \leftarrow true$
      end if
    end for
    if $flag = false$ **then**
      break
    end if
  end while
  for each subset $U \subseteq TBD$ with $|U| \notin size$ and $|U| \neq 0$ **do**
    $f' \leftarrow \bigoplus_{i \in U} f_i$
    $r_var_f' \leftarrow$ (linear "r"-vars of f') $\setminus$ (higher-degree vars of f')
    if $r_var_f' \subseteq SIM$ **then** ▷ Reduction Rule 3
      $size \leftarrow size \cup \{|U|\}$ ▷ update allowed sizes
      $updated3 \leftarrow true$
      break ▷ restart reduction rules 1 and 2
    end if
  end for
  if $updated1 = false$ and $updated2 = false$ and $updated3 = false$ **then**
    break
  end if
end while
return (SIM, TBD)

4 Case Studies

In this section, we construct the first-order OPINI masking for the S-boxes from the Ascon [DEMS21], PRESENT [BKL+07], PRINCE [BCG+12], and AES [Dae99] ciphers, and report their hardware performance in Tables 1 and 2. We then verify the OPINI security of these masked S-boxes using our symbolic verification technique. The corresponding verification results are summarized in Table 3.

4.1 S-box Gadgets and Full Ciphers

As a standard for lightweight cryptography, Ascon [DEMS21] is designed for efficient and low-cost implementations and is friendly to masking. Its S-box is a vectorial Boolean function over $\mathbb{F}_2^5 \to \mathbb{F}_2^5$ with algebraic degree 2. Owing to this low degree, the S-box can be implemented using a single layer of AND gates. Based on the five coordinate functions in Equation (11), we apply the technique described in Section 3.1 to construct a single-cycle OPINI gadget for the Ascon S-box. The requirements for random bits and registers are summarized below. Since each monomial derived from the input shares with index 1 is masked using two random bits, we require 8 random bits to mask the shared variables b_1, c_1, d_1, and e_1, and an additional 10 random bits to mask the redundant functions in the circuit contributing to the output shares with index 0. Owing to the randomness-reassignment technique, the circuit that produces the other output shares can be securely implemented using the same set of 18 random bits. In addition to the registers used to store the values masked by the above random bits, we require 8 additional registers to store the shared inputs $a_{0/1}$, $b_{0/1}$, $c_{0/1}$, and $d_{0/1}$. Consequently, the masked Ascon S-box requires only 18 random bits and 36 registers to achieve OPINI security. Since our technique applies to arbitrary vectorial Boolean functions, it can be trivially extended to S-boxes of different sizes. As representative examples, we also construct masking schemes for the 4-bit S-box (e.g., from PRESENT and PRINCE) and the 8-bit S-box from AES, following processes similar to those used for the Ascon S-box.

$$\begin{aligned} f(a,b,c,d,e) &= ad + de + a + b + d, \\ f(a,b,c,d,e) &= ae + be + a + b + c + d + e, \\ f(a,b,c,d,e) &= ab + a + c + d + 1, \\ f(a,b,c,d,e) &= bc + bd + cd + a + b + c + d + e, \\ f(a,b,c,d,e) &= ad + cd + de + b + c + d + e. \end{aligned} \tag{11}$$

To compare our schemes with the state of the art, we synthesize our implementations using Synopsys Design Compiler and the NanGate 45 nm library with the synthesis options: `compile_ultra -no_autoungroup -no_boundary_optimization`. Regarding experimental performance of related work, we use the reported experimental results of LMDPL and CCHPC_{RS} from [SBHM20, LMDM25]. The experimental results for HPC3+, HPC4, TSM, and OTSM are obtained by synthesizing the corresponding source codes. HPC4 mainly focuses on the scalability of the 2-input multiplication gadget across arbitrary security orders. The implementation of the first-order masked Ascon S-box provided in the artifact of [CSV24] employs 5 HPC4 gadgets to realize the non-linear functionality and 12 additional registers to synchronize internal signals. Our work concentrates on the construction of the first-order OPINI gadgets for arbitrary Boolean functions. In comparison, our gadget achieves a 21% reduction in hardware area and a 28% reduction in randomness compared to the first-order masked Ascon S-box using the HPC4 gadget.

We also report the hardware costs of masked PRESENT S-box [BKL+07], PRINCE S-box [BCG+12] and AES S-box [Dae99] in Table 1. For a broader perspective, we also provide several generic low-latency PINI gadgets and threshold implementations in Table 1. Note that OPINI offers higher security strength than PINI, as PINI does not account for transition default. Satisfying PINI may not be sufficient to guarantee security in a

single-cycle iterative implementation, whereas OPINI is explicitly designed to address this issue. In terms of delay, as shown in Table 1, our GOM gadget outperforms TSM for the PRESENT and PRINCE S-boxes. For the AES S-box, our GOM gadget is superior to LMDPL and CCHPC_{RS} in terms of delay, but slightly inferior to TSM. Threshold implementation is also an efficient solution for the first-order security. For example, the 4-share LLTI scheme [AZN21] requires 25780 GE of hardware area and does not require randomness. Beyond the ANF-based constructions considered in this work, other circuit representations may also affect the efficiency of the protected implementation. In particular, the two-input AND gadget HPC4 enables two masked AES S-box implementations based on the circuit representations of [Can05] and [BP12]. Both representations have an AND-depth of 4 and require 40 and 34 AND gates, respectively. For the same circuit representation, the GOM-based implementation requires less hardware area than the HPC4-based counterpart, since our proposed GOM_{AND} gadget has a smaller area cost than HPC4. For the same masked gadget, the implementation based on [BP12] requires less hardware area than that based on [Can05], because the former uses fewer AND gates. This shows that the efficiency of a protected implementation depends not only on the masked gadget itself, but also on the underlying circuit representation.

Table 1: Hardware performance of different first-order masked implementations.

Scheme	Security notion	Latency [cycle]	Area [GE]	Delay [bit]	Rand. [ns]	Ref.
AND2 Gadget						
HPC3^{+}	OPINI	2	95	0.15	3	[KM22]
HPC4	OPINI	1	94	0.18	5	[CSV24]
GOM	OPINI	1	77	0.19	5	This Work
Ascon S-box						
HPC3^{+}	OPINI	2	661	0.24	15	[KM22]
HPC4	OPINI	1	583	0.36	25	[CSV24]
GOM	OPINI	1	460	0.36	18	This Work
PRESENT S-box						
TSM	PINI	1	427	0.73	17	[VDB+24]
GOM	OPINI	1	569	0.59	20	This Work
PRINCE S-box						
TSM	PINI	1	450	0.79	18	[VDB+24]
OTSM[a]	OPINI	1	573	0.65	28	[RM26]
GOM	OPINI	1	631	0.52	20	This Work
AES S-box						
LMDPL	PINI[b]	1	3480	2.50	36	[SBHM20]
4-share LLTI	TI	1	25780	3.61	0	[AZN21]
TSM	PINI	1	14293	1.86	262	[VDB+24]
CCHPC_{RS}	PINI[b]	1	4062	2.22	34	[LMDM25]
HPC4 (Canright)	OPINI	4	4832	0.72	200	[Can05]
HPC4 (B-P)	OPINI	4	4297	0.53	170	[BP12]
OTSM[a]	OPINI	1	20055	1.89	508	[RM26]
GOM_{AND}(Canright)	OPINI	4	4192	0.66	200	This Work
GOM_{AND} (B-P)	OPINI	4	3753	0.49	170	This Work
GOM	OPINI	1	33762	1.89	268	This Work

[a] Independent and concurrent work.
[b] Under the robust but relaxed probing model [MM24].

Compared with the concurrent work OTSM, our gadget reduces the randomness cost by 29% (resp., 47%) and the delay by 20% (resp., 0%) for the PRINCE S-box (resp., AES S-box). These improvements mainly stem from the randomness-reassignment technique. Since our design primarily targets the randomness rather than the hardware area, the area of GOM is inferior to that of OTSM. To assess the total hardware area including randomness generation, we adopt the evaluation methodology of OTSM and instantiate an unrolled Trivium PRNG [CMM+24]. For the PRINCE S-box, the total area (gadget + PRNG) is 3560 GE for OTSM and 3428 GE for GOM. For the AES S-box, the total area is 56658 GE for OTSM and 54915 GE for GOM. Overall, our design slightly yields a 3% reduction in total area when accounting for randomness generation.

As for the full-cipher results, we provide the hardware performance of first-order masked Ascon-p[12], PRESENT, PRINCE and AES implementations in Table 2. Following the artifact of [CSV24], we develop a first-order masked implementation of Ascon-p[12], which iterates 12 rounds over a (320×2)-bit shared state. Due to the hardware advantage of GOM over HPC4 for the first-order masked Ascon S-box, our masked Ascon-p[12] implementation achieves 18% and 28% reductions in the hardware area and randomness, respectively. Referring to the implementation in [KSM22], we adopt a first-order masked, round-based PRESENT encryption function by replacing the original masked S-boxes with our GOM-based S-boxes. The resulting design exhibits a latency of 31 clock cycles. Likewise, based on the implementations in [RM26], we develop first-order masked, round-based PRINCE encryption/decryption and AES encryption by replacing the original masked S-boxes with our GOM-based S-boxes. Since GOM requires less randomness than OTSM, our masked PRINCE and AES implementations achieve a similar reduction in randomness. Compared with the two TI-based implementations [AZN21, MMM21], our masked PRINCE requires less hardware area, albeit at the cost of higher randomness.

Table 2: Hardware performance of different first-order masked full ciphers.

Design	Scheme	Latency [cycle]	Area [kGE]	Delay [ns]	Rand. [bit]	Ref.
Ascon-p[12]	HPC4	12	41.98	1.22	1600	[CSV24]
	GOM	12	34.24	1.25	1152	This Work
PRESENT	GHPC	62	31.56	5.22	72	[KSM22]
	GOM	31	13.90	1.55	360	This Work
PRINCE	LLTI	12	25.86	2.05	48	[AZN21]
	4-share TI	12	26.16	–	0	[MMM21]
	OTSM[a]	12	12.33	1.41	448	[RM26]
	GOM	12	13.08	1.39	320	This Work
AES	TSM	21	72.07	1.78	736	[VDB+25]
	TSM$^+$−OTSM[a]	11	269.88	3.30	5080	[RM26]
	GOM	11	549.50	3.30	4288	This Work

[a] Independent and concurrent work.

4.2 Formal Verification of Gadgets

To verify the claimed security of the masked S-boxes constructed in Section 4.1, we employ the off-the-shelf verification tool PROLEAD [MM22], which is widely used to analyze the statistical independence of simulated intermediate probes under the glitch- and transition-robust probing model, e.g., [CSV24]. We first develop a general single-cycle iterative architecture, shown in Figure 2, to evaluate our gadgets. The architecture is similar to that in Figure 13 of [CSV24]. PROLEAD-based assessments are then performed by instantiating each masked S-box within this architecture, respectively. The evaluations are conducted on a machine equipped with an AMD EPYC 7302 CPU at 2.4 GHz with 48 cores and 256 GB of RAM. In all cases, no leakage is reported after 100 million simulations.

Since PROLEAD is based on simulations and statistical hypothesis testing, 100 million simulations provide a high level of confidence. However, PROLEAD cannot formally establish the OPINI security of a given design.

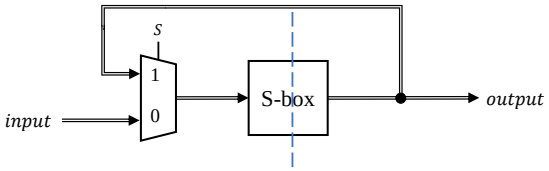


Figure 2: Schematic of the iterative S-box.

To verify the OPINI security of our masked S-boxes, we apply the symbolic verification technique proposed in Section 3.2. We begin by describing implementation details of our verification technique, and then present verification results for several low-latency OPINI gadgets, including GOM, HPC4, and OTSM.

To implement our verification technique, we develop a symbolic analysis procedure in SageMath. For each netlist signal (i.e., corresponding to the Boolean functions f_i and g_j in Section 3.2), we extract its input-variable set (i.e., $vars_f[i]$ and $vars_g[j]$) by recursively traversing the signal's transitive fan-in cone backward through gate input pins. The traversal terminates at primary inputs, randomness sources, or flip-flop outputs. We collect all such sources to form the input-variable set. We extract the set $r_var[i]$ by converting f_i into Algebraic Normal Form (ANF) and collecting the random variables whose algebraic degree is one, i.e., they appear in degree-1 monomials but in no monomial of degree at least 2. The ANF representation is computed using SageMath's `BooleanPolynomialRing`, which supports algebraic degree computation and polynomial operations. We model probes symbolically by parsing the corresponding ANF expressions, and then apply the reduction rules to derive the information needed to simulate these probes. To improve runtime, we parallelize both the preprocessing and symbolic simulation phases using SageMath's `multiprocessing` module.

Table 3: Verification time of single-cycle OPINI gadgets using our technique

Scheme	Time (This Work)	Scheme	Time (This Work)
$\text{GOM}_{\text{Ascon S-box}}$	[1]✓[4.05 s]	$\text{HPC4}_{\text{Ascon S-box}}$	[1]✓[50.38 s]
$\text{GOM}_{\text{PRESENT S-box}}$	[1]✓[4.69 s]	HPC4 o1	[1]✓[0.54 s]
$\text{GOM}_{\text{PRINCE S-box}}$	[1]✓[5.27 s]	HPC4 o2	[2]✓[1.78 m]
$\text{GOM}_{\text{AES S-box}}$	[1]✓[19.39 m]	$\text{OTSM}_{\text{PRINCE S-box}}$	[1]✓[9.08 s]

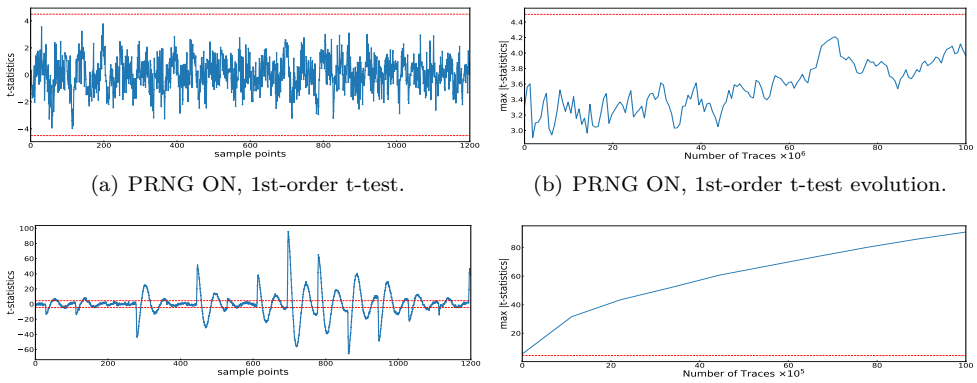
For the masked Ascon S-box, the inputs provided to our verification procedure are the target circuit and the security order $d = 1$. The circuit is characterized by the function set $\mathcal{F} = \{f_0, f_1, \ldots, f_{35}\}$, corresponding to the 36 registers discussed in Section 4.1, and the output function set $\mathcal{G} = \{g_0, g_1, \ldots, g_9\}$, corresponding to the 10 output wires. Based on the above two inputs, the total number of probe configurations to be verified is $\binom{36}{1} + \binom{10}{1} = 46$. All probe cases are analyzed in parallel using the reduction rules outlined in Algorithm 2. The verification experiments are conducted on a virtual machine equipped with 6 logical cores and 8 GB of RAM, running on an Intel Core i7-8700 CPU at 3.20 GHz. Using the same analysis procedure, we also verify other low-latency OPINI schemes, including our masked PRESENT, PRINCE, and AES S-boxes, as well as the existing HPC4 gadget. In Table 3, we denote the first- and second-order HPC4 gadgets by HPC4 o1 and HPC4 o2, respectively, and the masked Ascon implementation based on the first-order HPC4 gadget by $\text{HPC4}_{\text{Ascon S-box}}$. As shown in Table 3, the verification time increases with the circuit size and the security order. For the Ascon S-box with 46 probe configurations, the verification completes in approximately 4.05 seconds and confirms the

expected OPINI security. However, the masked AES S-box yields $\binom{300}{1} + \binom{16}{1} = 316$ probe configurations, and the verification requires approximately 19.39 minutes. Moreover, our analysis procedure also verifies the OPINI security of the OTSM-instantiated PRINCE S-box. Due to the large scale of the OTSM-instantiated AES S-box, our procedure does not terminate within 48 hours, and thus neither confirms OPINI security nor returns a counterexample. As a weaker evaluation, we verify the security of eight AES S-box subcircuits (one per output bit), requiring 6.24, 6.70, 6.83, 6.11, 5.54, 4.97, 5.41, and 6.38 minutes, respectively.

5 Experimental Verification

Although the OPINI security of our gadgets is guaranteed by the security proofs and formal verification techniques, we further conduct FPGA-based experiments to demonstrate the practical security. Specifically, we first evaluate a single masked S-box using the architecture in Figure 2, and then the full cipher implementations.

The practical experiment is conducted on a SAKURA-G board equipped with a Spartan-6 FPGA as the host core. To prevent overlap in power traces between clock cycles and facilitate trace alignment, the clock frequency of the FPGA board is set to 3 MHz, which is common practice in TVLA-based leakage evaluations [VDB+24, VDB+25, WFH+26]. Power traces are captured using a PicoScope 5244D oscilloscope configured with a sampling rate of 250 MS/s. The random bits required by our gadget are generated using an unrolled Trivium [CMM+24] implementation. To evaluate the security of our designs, we apply the fixed-versus-random t-test [GGJR+11], a widely recognized methodology for assessing the security of masked implementations.



(a) PRNG ON, 1st-order t-test.

(b) PRNG ON, 1st-order t-test evolution.

(c) PRNG OFF, 1st-order t-test.

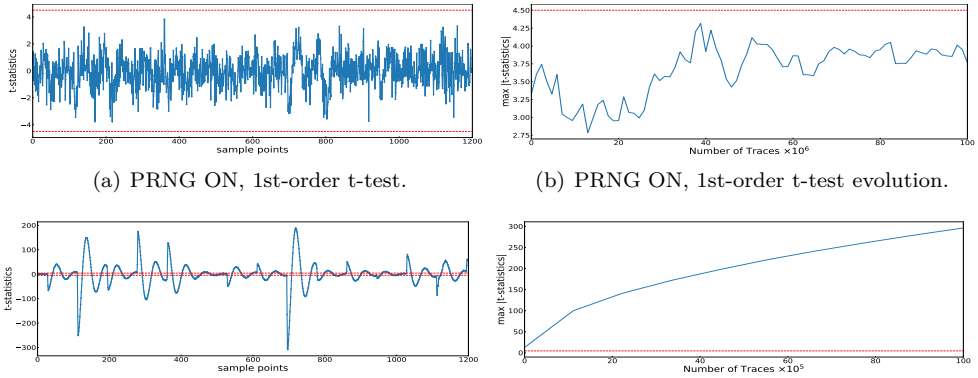
(d) PRNG OFF, 1st-order t-test evolution.

Figure 3: The t-test results for our single-cycle iterative Ascon S-box.

To evaluate the masked Ascon S-box, we instantiate it within the architecture shown in Figure 2. Since this gadget is first-order secure, the t-test result using 100 million traces, shown in Figure 3(a), confirms the claimed security level. Figure 3(b) plots the evolution of the maximum absolute t-statistic as the number of traces increases. The first-order statistic consistently stays below the 4.5 threshold across all trace counts, confirming the first-order security. When the PRNGs are disabled, the results using 10 million traces, shown in Figures 3(c) and 3(d), validate our setup by clearly exhibiting first-order leakage.

Similar experiments for the masked PRESENT S-box and PRINCE S-box are also conducted. Due to page limitations, we present the t-test results for the masked PRESENT S-box in Figures 4 and those for the masked PRINCE S-box in the Supplementary Material, both of which confirm the claimed first-order security. In contrast, the masked AES S-box could not be successfully implemented on the target FPGA board because of its unaffordable resource requirements. Nevertheless, given the previously established security

proofs, the formal verification results, and the practical experimental results for the masked S-boxes from the Ascon, PRESENT, and PRINCE ciphers, we believe that experiments for the masked AES S-box would yield similar t-test results.



(a) PRNG ON, 1st-order t-test.

(b) PRNG ON, 1st-order t-test evolution.

(c) PRNG OFF, 1st-order t-test.

(d) PRNG OFF, 1st-order t-test evolution.

Figure 4: The t-test results for our single-cycle iterative PRESENT S-box.

To further demonstrate practical security at the full-cipher level, we additionally evaluate the round-based PRESENT encryption implementation developed in Section 4.1 as a representative case. Due to page limitations, the corresponding t-test results are provided in the Supplementary Material. Using 100 million traces with the PRNG enabled, the results confirm the first-order security. When the PRNGs are disabled, clear first-order leakage is observed, which validates the experimental setup.

6 Conclusions

In this work, we present a low-latency first-order OPINI gadget applicable to arbitrary Boolean functions. Its construction is based on two new techniques: extending the HPC4 idea to mask each monomial with two random bits and introducing a randomness-reassignment technique. Following the definition of OPINI, this gadget is suitable for single-cycle iterative architectures. Compared with the existing masked Ascon S-box, our gadget achieves reductions of 21% in hardware area and 28% in random bits. To demonstrate the generality of our gadget, we instantiate several high-algebraic-degree (> 2) S-boxes, such as the 3-degree PRESENT S-box and the 7-degree AES S-box, and report their hardware performance. We also propose a symbolic verification technique to confirm the OPINI security of our masked S-boxes within 20 minutes. Finally, we validate the practical security through FPGA-based experiments.

In the future, two aspects can be explored. As shown in Table 1, the security enhancement from robust PINI to robust OPINI leads to a 20%∼50% area increase. Further optimizations are needed to reduce the hardware area. Moreover, constructing generic low-latency, high-order OPINI gadgets remains an interesting and challenging direction.

Acknowledgments

The authors would like to thank the anonymous reviewers for their valuable comments and suggestions. This work is supported by the National Key R&D Program of China (Grant No. 2023YFA1009500, 2024YFA1013000), the National Natural Science Foundation of China (Grant No. U2336207), the National Cryptologic Science Fund of China (Grant No. 2025NCSF01013), Key R&D Program of Shandong Province, China (Grant No. 2026CXPT037), and Fundamental and Interdisciplinary Disciplines Breakthrough Plan of the Ministry of Education of China (Grant No. JYB2025XDXM114). This work is also

supported by Project SDCX-ZG-202603011 Funded by Postdoctoral Innovation Program of Shandong Province.

References

[AZN21] Victor Arribas, Zhenda Zhang, and Svetla Nikova. LLTI: Low-Latency Threshold Implementations. *IEEE Trans. Inf. Forensics Secur.*, 16:5108–5123, 2021.

[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, *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, Vienna, Austria, October 24-28, 2016*, pages 116–129. ACM, 2016.

[BCG+12] Julia Borghoff, Anne Canteaut, Tim Güneysu, Elif Bilge Kavun, Miroslav Knezevic, Lars R. Knudsen, Gregor Leander, Ventzislav Nikov, Christof Paar, Christian Rechberger, Peter Rombouts, Søren S. Thomsen, and Tolga Yalçin. PRINCE - A Low-Latency Block Cipher for Pervasive Computing Applications - Extended Abstract. In Xiaoyun Wang and Kazue Sako, editors, *Advances in Cryptology - ASIACRYPT 2012 - 18th International Conference on the Theory and Application of Cryptology and Information Security, Beijing, China, December 2-6, 2012. Proceedings*, volume 7658 of *Lecture Notes in Computer Science*, pages 208–225. Springer, 2012.

[BKL+07] Andrey Bogdanov, Lars R. Knudsen, Gregor Leander, Christof Paar, Axel Poschmann, Matthew J. B. Robshaw, Yannick Seurin, and C. Vikkelsoe. PRESENT: An Ultra-Lightweight Block Cipher. In Pascal Paillier and Ingrid Verbauwhede, editors, *Cryptographic Hardware and Embedded Systems - CHES 2007, 9th International Workshop, Vienna, Austria, September 10-13, 2007, Proceedings*, volume 4727 of *Lecture Notes in Computer Science*, pages 450–466. Springer, 2007.

[BP12] Joan Boyar and René Peralta. A small depth-16 circuit for the AES s-box. In Dimitris Gritzalis, Steven Furnell, and Marianthi Theoharidou, editors, *Information Security and Privacy Research - 27th IFIP TC 11 Information Security and Privacy Conference, SEC 2012, Heraklion, Crete, Greece, June 4-6, 2012. Proceedings*, IFIP Advances in Information and Communication Technology, pages 287–298. Springer, 2012.

[Can05] David Canright. A very compact s-box for AES. In Josyula R. Rao and Berk Sunar, editors, *Cryptographic Hardware and Embedded Systems - CHES 2005, 7th International Workshop, Edinburgh, UK, August 29 - September 1, 2005, Proceedings*, volume 3659 of *Lecture Notes in Computer Science*, pages 441–455. Springer, 2005.

[CGLS21] Gaëtan Cassiers, Benjamin Grégoire, Itamar Levi, and François-Xavier Standaert. Hardware Private Circuits: From Trivial Composition to Full Verification. *IEEE Trans. Computers*, 70(10):1677–1690, 2021.

[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, *Advances in Cryptology - CRYPTO '99, 19th Annual International Cryptology Conference, Santa Barbara, California, USA, August 15-19,*

1999, Proceedings, volume 1666 of *Lecture Notes in Computer Science*, pages 398–412. Springer, 1999.

[CMM+24] 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. *IACR Commun. Cryptol.*, 1(2):4, 2024.

[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 Trans. Cryptogr. Hardw. Embed. Syst.*, 2021(2):136–158, 2021.

[CSV24] Gaëtan Cassiers, François-Xavier Standaert, and Corentin Verhamme. Low-Latency Masked Gadgets Robust against Physical Defaults with Application to Ascon. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2024(3):603–633, 2024.

[Dae99] Joan Daemen. AES Proposal: Rijndael, 1999.

[DEMS21] Christoph Dobraunig, Maria Eichlseder, Florian Mendel, and Martin Schläffer. Ascon v1.2: Lightweight Authenticated Encryption and Hashing. *J. Cryptol.*, 34(3):33, 2021.

[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 Trans. Cryptogr. Hardw. Embed. Syst.*, 2018(3):89–120, 2018.

[GGJR+11] Benjamin Jun Gilbert Goodwill, Josh Jaffe, Pankaj Rohatgi, et al. A testing methodology for side-channel resistance validation. In *NIST non-invasive attack testing workshop*, volume 7, pages 115–136, 2011.

[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.

[GP99] Louis Goubin and Jacques Patarin. DES and Differential Power Analysis (The "Duplication" Method). In Çetin Kaya Koç and Christof Paar, editors, *Cryptographic Hardware and Embedded Systems, First International Workshop, CHES'99, Worcester, MA, USA, August 12-13, 1999, Proceedings*, volume 1717 of *Lecture Notes in Computer Science*, pages 158–172. Springer, 1999.

[KJJ99] Paul C. Kocher, Joshua Jaffe, and Benjamin Jun. Differential Power Analysis. In Michael J. Wiener, editor, *Advances in Cryptology - CRYPTO '99, 19th Annual International Cryptology Conference, Santa Barbara, California, USA, August 15-19, 1999, Proceedings*, volume 1666 of *Lecture Notes in Computer Science*, pages 388–397. Springer, 1999.

[KM22] David Knichel and Amir Moradi. Low-Latency Hardware Private Circuits. In Heng Yin, Angelos Stavrou, Cas Cremers, and Elaine Shi, editors, *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security, CCS 2022, Los Angeles, CA, USA, November 7-11, 2022*, pages 1799–1812. ACM, 2022.

[Koc96] Paul C. Kocher. Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS, and Other Systems. In Neal Koblitz, editor, *Advances in Cryptology - CRYPTO '96, 16th Annual International Cryptology Conference, Santa Barbara, California, USA, August 18-22, 1996, Proceedings*, volume 1109 of *Lecture Notes in Computer Science*, pages 104–113. Springer, 1996.

[KSM20] David Knichel, Pascal Sasdrich, and Amir Moradi. SILVER - statistical independence and leakage verification. In Shiho Moriai and Huaxiong Wang, editors, *Advances in Cryptology - ASIACRYPT 2020 - 26th International Conference on the Theory and Application of Cryptology and Information Security, Daejeon, South Korea, December 7-11, 2020, Proceedings, Part I*, volume 12491 of *Lecture Notes in Computer Science*, pages 787–816. Springer, 2020.

[KSM22] David Knichel, Pascal Sasdrich, and Amir Moradi. Generic Hardware Private Circuits Towards Automated Generation of Composable Secure Gadgets. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2022(1):323–344, 2022.

[LMDM25] Daniel Lammers, Nicolai Müller, Siemen Dhooghe, and Amir Moradi. Constant-Cycle Hardware Private Circuits. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2025(4):172–214, 2025.

[MCS22] Charles Momin, Gaëtan Cassiers, and François-Xavier Standaert. Handcrafting: Improving Automated Masking in Hardware with Manual Optimizations. In Josep Balasch and Colin O'Flynn, editors, *Constructive Side-Channel Analysis and Secure Design - 13th International Workshop, COSADE 2022, Leuven, Belgium, April 11-12, 2022, Proceedings*, volume 13211 of *Lecture Notes in Computer Science*, pages 257–275. Springer, 2022.

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

[MM24] Nicolai Müller and Amir Moradi. Robust but Relaxed Probing Model. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2024(4):451–482, 2024.

[MMM21] Nicolai Müller, Thorben Moos, and Amir Moradi. Low-Latency Hardware Masking of PRINCE. In Shivam Bhasin and Fabrizio De Santis, editors, *Constructive Side-Channel Analysis and Secure Design - 12th International Workshop, COSADE 2021, Lugano, Switzerland, October 25-27, 2021, Proceedings*, volume 12910 of *Lecture Notes in Computer Science*, pages 148–167. Springer, 2021.

[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, *Information and Communications Security, 8th International Conference, ICICS 2006, Raleigh, NC, USA, December 4-7, 2006, Proceedings*, volume 4307 of *Lecture Notes in Computer Science*, pages 529–545. Springer, 2006.

[RBN+15] Oscar Reparaz, Begül Bilgin, Svetla Nikova, Benedikt Gierlichs, and Ingrid Verbauwhede. Consolidating Masking Schemes. In Rosario Gennaro and Matthew Robshaw, editors, *Advances in Cryptology - CRYPTO 2015 - 35th Annual Cryptology Conference, Santa Barbara, CA, USA, August 16-20, 2015, Proceedings, Part I*, volume 9215 of *Lecture Notes in Computer Science*, pages 764–783. Springer, 2015.

[RFSG22] Jan Richter-Brockmann, Jakob Feldtkeller, Pascal Sasdrich, and Tim Güneysu. VERICA - Verification of Combined Attacks Automated formal verification of security against simultaneous information leakage and tampering. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2022(4):255–284, 2022.

[RM26] Hemin Rahimi and Amir Moradi. TSM+ and OTSM: Correct Application of Time Sharing Masking in Round-Based Designs. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2026(2):240–271, 2026.

[SBHM20] Pascal Sasdrich, Begül Bilgin, Michael Hutter, and Mark E. Marson. Low-Latency Hardware Masking with Application to AES. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2020(2):300–326, 2020.

[Sha79] Adi Shamir. How to Share a Secret. *Commun. ACM*, 22(11):612–613, 1979.

[VDB+24] Dilip Kumar S. V., Siemen Dhooghe, Josep Balasch, Benedikt Gierlichs, and Ingrid Verbauwhede. Time Sharing - A Novel Approach to Low-Latency Masking. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2024(3):249–272, 2024.

[VDB+25] Dilip Kumar S. V., Siemen Dhooghe, Josep Balasch, Benedikt Gierlichs, and Ingrid Verbauwhede. Higher-Order Time Sharing Masking. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2025(2):235–267, 2025.

[WFH+26] Lixuan Wu, Yanhong Fan, Jiahui He, Guowei Liu, Chaoran Wang, and Meiqin Wang. A mirrored-circuits approach for low-latency and low-randomness composable PINI gadgets. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2026:26–52, 2026.

[ZCF25] Feng Zhou, Hua Chen, and Limin Fan. Prover - Toward More Efficient Formal Verification of Masking in Probing Model. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2025(1):552–585, 2025.