IACR Transactions on Cryptographic Hardware and Embedded Systems
ISSN 2569-2925, Vol. 2025, No. 2, pp. 235–267. DOI:10.46586/tches.v2025.i2.235-267

# Higher-Order Time Sharing Masking

Dilip Kumar S. V.[1], Siemen Dhooghe[1], Josep Balasch[2], Benedikt Gierlichs[1] and Ingrid Verbauwhede[1]

[1] COSIC, KU Leuven, Belgium
[2] e-Media Research Lab, STADIUS, KU Leuven, Belgium
{dshanmug,siemen.dhooghe,josep.balasch,benedikt.gierlichs,ingrid.verbauwhede}@esat.kuleuven.be

**Abstract.** At CHES 2024, Time Sharing Masking (TSM) was introduced as a novel low-latency masking technique for hardware circuits. TSM offers area and randomness efficiency, as well as glitch-extended PINI security, but it is limited to first-order security. We address this limitation and generalize TSM to higher-order security while maintaining all of TSM's advantages. Additionally, we propose an area-latency tradeoff. We prove HO-TSM glitch-extended PINI security and successfully evaluate our circuits using formal verification tools. Furthermore, we demonstrate area- and latency-efficient implementations of the AES S-box, which do not exhibit leakage in TVLA on FPGA. Our proposed tradeoff enables a first-order secure implementation of a complete AES-128 encryption core with 92 kGE, 920 random bits per round, and 20 cycles of latency, which does not exhibit leakage in TVLA on FPGA.

**Keywords:** Hardware · Masking · Probing Security · Side-Channel Analysis · Low-Latency

## 1 Introduction

Cryptographic algorithms implemented on computer systems are susceptible to physical attacks that can extract sensitive information being processed by the system. An adversary could monitor side-channel information, such as power consumption [KJJ99], execution time [Koc96], or electromagnetic emanations [GMO01, QS01], to uncover sensitive data like cryptographic secret keys. Masking [CJRR99, GP99] is a popular countermeasure that splits data into multiple shares, thereby removing the correlation between the data and side-channel information to protect implementations against such attacks.

Over the past years, masking techniques for hardware have, most importantly, aimed to guarantee security in the presence of glitches [MPG05]. To securely mask hardware implementations, several techniques have been developed, including popular methods such as Threshold Implementations (TI) [NRR06], Consolidating Masking Schemes (CMS) [RBN+15], and Domain Oriented Masking (DOM) [GMK16]. These methods primarily focus on reducing the area overhead and minimizing the fresh randomness required to ensure security. Another line of research in hardware masking that has gained traction in recent years is focused solely on reducing latency. An initial effort, LMDPL [LMW14], introduced by Leiserson *et al.*, achieved the construction of a first-order secure AES S-Box in just 2 cycles. Subsequent improvements by Sasdrich et al. [SBHM20] further refined this approach to create a first-order secure round-based AES-128 implementation operating in ten cycles. Later, Gross et al. introduced an alternative low-latency approach, GLM [GIB18], derived from DOM, capable of securely masking vectorial Boolean functions of any algebraic degree with a latency of just one cycle, regardless of the security order.

Licensed under Creative Commons License CC-BY 4.0.
Received: 2024-10-15 Accepted: 2024-12-15 Published: 2025-03-12

Recent works include GHPC [KSM22] and TSM [VDB+24], both of which are low-latency approaches capable of producing first-order single-cycle masked gadgets. They offer the additional benefit of compositional security under the Probe Isolating Non-Interference (PINI) security notion [CS20]. A prevailing theme across all low-latency approaches is the trade-off between area and randomness to minimize latency. The single-cycle first-order GLM AES S-Box is 60.7 kGE in size and requires 2048 bits of randomness, while the $\text{GHPC}_{\text{LL}}$ AES S-Box has an area of 64.1 kGE and also requires 2048 bits of randomness. In comparison, the single-cycle TSM AES S-Box shows improvements in both area and randomness, with a size of 14.3 kGE and a randomness requirement of 262 bits. However, the area and randomness requirements remain substantial compared to other S-Boxes with higher latency that employ techniques like DOM, TI. Notably, many low-latency approaches are limited to provide only first-order security.

**Contributions.** We extend TSM to achieve higher-order security (HO-TSM) and introduce an area-latency trade-off construction that leverages TSM and HO-TSM, leading to substantial reductions in area and randomness costs. Our contributions are as follows:

- We introduce a higher-order extension, HO-TSM, that builds upon the foundation of first-order TSM [VDB+24]. The fundamental principle of HO-TSM is to process one share of each input in every clock cycle while maintaining isolation of these computations through the use of registers. Notably, the latency of HO-TSM is dictated by the security order, rather than the algebraic degree of the function. For a $d^{th}$-order secure implementation, the latency is precisely $d$ cycles.
- We formally outline our construction and demonstrate that any function secured using our method achieves $d^{\text{th}}$-order glitch-extended PINI and SNI (Strong Non-Interference) [BBD+16] security.
- We develop a two-cycle, second-order HO-TSM AES S-Box that occupies less area and requires less randomness than the only other low-latency, second-order AES S-Box by Gross *et al.* (GLM) [GMK16]. Our FPGA implementation of this S-Box exhibits no signs of first- or second-order leakage in TVLA with 100 million traces.
- We present an area-latency trade-off for HO-TSM that significantly reduces both area and randomness costs, with only a single cycle increase in latency. We construct a new two-cycle, first-order PINI and SNI secure AES S-Box that shows substantial improvements in utilization costs and develop a complete round-based AES-128 implementation that executes in twenty cycles. The FPGA implementation of our full AES-128 demonstrates no first-order leakage in TVLA with 250 million traces.
- Using our trade-off construction, we develop another three-cycle, second-order PINI and SNI secure AES S-Box. Our S-Box exhibits no first- or second-order leakage in TVLA with 100 million traces.
- We use SILVER [KSM20] to validate small examples of both HO-TSM and our trade-off construction, providing evidence for PINI and SNI security. In addition, we also verify our S-Boxes using the formal verification tools `maskVerif` [BBC+19] and PROLEAD [MM22].
- We provide the HDL source code[1] for the designs presented in this paper and their synthesis scripts.

**Outline.** Section 2 introduces notations and essential background information. In Section 3, we first illustrate our concept using a toy example by developing a straightforward second-order masked AND gate. We then provide a formal description of our approach

[1] https://github.com/KULeuven-COSIC/TSM

applicable to any Boolean function, and a proof demonstrating its $d^{th}$-order glitch-extended PINI and SNI security. We apply our approach to produce a two-cycle second-order secure AES S-Box. In Section 4, we introduce an area-latency trade-off for HO-TSM and establish its security under the PINI and SNI security notions. We construct both a first-order and a second-order AES S-Box. Using the first-order secure S-Box, we proceed to design a complete round-based AES-128 implementation. In Section 5, we conduct formal verification of our AES S-Boxes, and practically validate our designs using TVLA by implementing them on FPGA. In Section 6, we summarize our work and discuss possible future work.

# 2 Preliminaries

We introduce essential concepts and notations which form the basis for the methodologies and security proofs we present in later sections.

## 2.1 Notation

In Boolean masking, each value $x \in \mathbb{F}_2^k$ is split into $d+1$ uniform random shares $x_i$ such that $x = x_0 + \ldots + x_d$. For $x \in \mathbb{F}_2^k$ a $k$-bit word, we denote the bits of the word by square brackets $(x[0], ..., x[k-1])$. We represent the sharing of $x$ as $\bar{x} = (x_0, x_1, \cdots, x_d) \in \left(\mathbb{F}_2^k\right)^{d+1}$ such that $x_0 + x_1 \cdots + x_d = x$ with $x_i = (x_i[0], ..., x_i[k-1])$ the notation of the share-words in separate bits. For a set of share indices A $\subset \{0, \ldots, d\}$, we denote $x_A = \{x_i : x \in A\}$. We denote $x \xleftarrow{\$} X$ to represent selecting a value uniformly and randomly from the set $X$.

## 2.2 Circuit Model

For the purpose of security proofs and to explain masking methods, we represent algorithms in the shape of a directed-acyclic graph called a circuit. In this circuit, an edge represents a bit ($\mathbb{F}_2$) or a word ($\mathbb{F}_2^k$) and a node represents an operation of its fan-in such as an XOR or an AND gate. We also consider nodes without input which can output a uniformly distributed random bit or word.

## 2.3 Probing Model

We use the probing model, as originally proposed by Ishai *et al.* [ISW03], to model the side-channel adversary. More specifically, the $d^{\text{th}}$-order probing adversary is one who has access to the layout of the circuit he is attacking and who is able to request (before the execution of the circuit) the digital value of up to $d$ wires (probes) in the circuit.

The above adversary is expanded following the glitch-extended robust probing model by Faust *et al.* [FGP+18] where each probe is expanded such that it returns not only the wire value but all registered values leading up to that wire. This adversarial expansion is a way to capture the effect of glitches and propagation delays in the circuit's physical implementation.

We call a circuit $d^{\text{th}}$-order glitch-extended probing secure if there exists a simulator which can simulate probed values in such a way that an adversary cannot distinguish the actual circuit from the simulator. The adversary has the power to choose the circuit's input secrets and the simulator is not given this information. This simulation-based security is a clever way of saying that the probed values' distribution (made by the circuit's internal randomness) is independent of its input secrets.

### 2.3.1 Composable Probing Security

We consider a simulation security game for the probing adversary, because apart from easing certain proofs, it allows for composable security. Namely, using the Strong Non-

Interference (SNI) framework by Barthe *et al.* [BBD+16] and its extension to Probe-Isolated Non-Interference (PINI) by Cassiers and Standaert [CS20], one can show that if a small part of the circuit (gadget) is composable secure then its composition with other composable secure gadgets remains secure.

We first provide the definition of simulatability as given by in [CS20].

**Definition 1** (Simulatability [CS20])**.** Let $P = \{p_1, ..., p_\ell\}$ be a set of $\ell$ probes of a gadget $C$ and $C_P$ the tuple of values of the probes for an execution of $C$. Let $I = \{(i_1, j_1), ..., (i_m, j_m)\} \subset \{0, ..., d-1\} \times \{0, ..., k-1\}$ be a set of input wires of $C$. A simulator is a randomized function $\mathcal{S} : \mathbb{F}_q^m \to \mathbb{F}_q^\ell$. The set of probes $P$ can be simulated with the set of input wires $I$ if there exists a simulator $\mathcal{S}$ such that for any inputs $\bar{x}$, the distributions $C_P(\bar{x})$ and $\mathcal{S}(x_{i_1}[j_1], ..., x_{i_m}[j_m])$ are equal, where the probability is over the random bits in $C$ and $\mathcal{S}$.

In this work, we consider composability in the form of both the SNI [BBD+16] and the PINI [CS20] frameworks. The reason for opting for a more restrictive simulation-based model will become clear in Section 4, where we provide a low-cost area-latency trade-off. In the more restrictive NI model, we consider simulation where the simulator is not given any input shares when probing an output. We introduce both PINI and SNI.

**Definition 2** (SNI)**.** A gadget is called $t$–SNI if for any set of $t_0$ (glitch-extended) probes on intermediate variables and every set of $t_1$ (glitch-extended) probes on output shares such that $t_0 + t_1 \leq t$, the totality of the probes can be simulated by only $t_0$ shares of each input.

**Definition 3** (PINI)**.** Let $G$ be a gadget over $d$ shares and $P$ a set of $t_0$ (glitch-extended) probes on wires of $G$ (called internal probes). Let $A$ be a set of $t_1$ share indices. $G$ is $t$-PINI if for all $P$ and $A$ such that $t_0 + t_1 \leq t$, there exists a set $B$ of at most $t_0$ share indices such that probes on the set of wires $P \cup y_A$ can be simulated with the wires $x_{A \cup B}$, with $x_i$ denoting all inputs of share $i$ and $y_i$ denoting all outputs of share $i$.

In the regular PINI notion, the simulator is given the input share (the share with the same index) for every probed output share and in the regular SNI notion, the simulator is given arbitrary shares of each input instead of the same index for each input. Instead, we demand that the simulation of the gadget can be made using the same index for each input share for each placed internal probe and that the simulator is not given any shares when probing the gadget's output.

## 2.4 Time Sharing Masking

Time Sharing Masking (TSM) is a low-latency masking scheme introduced by Kumar S. V. *et al.* [VDB+24] designed to perform first-order masking of any (vectorial) Boolean function in hardware using a single register layer. Moreover, the technique has been proven to be secure under the PINI composable security notion. In this subsection, we provide a comprehensive description of TSM as applied to an arbitrary (vectorial) Boolean function.

Consider an arbitrary Boolean function

$$f : \mathbb{F}_2^k \to \mathbb{F}_2^\ell \quad : \quad x = \big(x[0], \ldots, x[k-1]\big) \mapsto f\big(x[0], \ldots, x[k-1]\big).$$

We denote its two-share masking by $\bar{F} : \mathbb{F}_2^{2k} \to \mathbb{F}_2^{2\ell}$, such that $\bar{x} \mapsto (F_0(\bar{x}), F_1(\bar{x}))$ and $F_0(\bar{x}) + F_1(\bar{x}) = f(x_0 + x_1)$. The sharing of the $\ell$ coordinate functions is given by $F_0 = (F_0[0], \ldots, F_0[\ell-1])$ and $F_1 = (F_1[0], \ldots, F_1[\ell-1])$.

We explain the TSM method constructively, beginning with a reformulation of the algebraic normal form (ANF) for one coordinate function $f[m]$ of $f$, where $m \in \{0, \ldots, \ell-1\}$.

The ANF decomposes $f[m]$ in a way that facilitates its masking. Any single-output Boolean function $f[m]$ can be expressed in its ANF as a sum of products of input bits:

$$f[m]\big(x[0],\ldots,x[k-1]\big) \;=\; \sum_{S\in\mathcal{P}_k} \alpha_S \prod_{j\in S} x[j] \quad \text{with} \quad \alpha_S \in \mathbb{F}_2. \tag{1}$$

Here, $\mathcal{P}_k$ denotes the power set of the indices $\{0,\ldots,k-1\}$, so that $|\mathcal{P}_k| = 2^k$. The sum is over all elements $S \in \mathcal{P}_k$. For any $S$, $\alpha_S$ determines whether the product of input bits $x[j]$ for $j \in S$ is present in the ANF of $f[m]$.

We define:

$$\mathcal{M}_m \;=\; \big\{S \in \mathcal{P}_k \;\mid\; \alpha_S = 1\big\} \tag{2}$$

as the set of all indices of products present in $f[m]$.

### 2.4.1 Decomposing Product Terms in the ANF

Each product indexed by $S \in \mathcal{M}_m$ is masked using Boolean masking and decomposed into a sum of products of shares from domains 0 and 1:

$$\prod_{j\in S} x[j] \;=\; \prod_{j\in S}\big(x_0[j] + x_1[j]\big) \;=\; \sum_{I\subseteq S}\Big(\prod_{j\in I} x_0[j]\Big)\,\Big(\prod_{j\in S\setminus I} x_1[j]\Big). \tag{3}$$

Here, the sum is over all $I \in \mathcal{P}(S)$, the power set of $S$. Recall that $S$ represents one specific combination of input bits $x[j]$. The sum is thus over all possible combinations of bits in $S$. The first product involves shares from domain 0 for bits in $I$, while the second involves shares from domain 1 for bits which are not in $I$, but in $S$.

We define two sets of functions $g^0$ and $g^1$, each operating on a single share domain, 0 and 1, respectively:

$$g^0_{\pi(I)}(x_0) \;=\; \prod_{j\in I} x_0[j], \qquad g^1_{\pi(I)}(x_1) \;=\; \prod_{j\in I} x_1[j] \tag{4}$$

where $I \subseteq \{0,\ldots,k-1\}$. For brevity, we write $g^0_{\pi(I)}(\cdot)$ to denote $g^0_{\pi(I)}$ evaluated on the share $x_0$. Similarly, $g^1_{\pi(I)}(\cdot)$ for the function $g^1_{\pi(I)}$ evaluated on $x_1$. The indexing function $\pi$ maps each $I \in \mathcal{P}_k$ to a unique integer in $\{0,\ldots,2^k-1\}$. For the empty set, $g^0_{\pi(\emptyset)}(\cdot) = g^1_{\pi(\emptyset)}(\cdot) = 1$.

Using these definitions in Eq.(4), we rewrite Eq.(3) as:

$$\prod_{j\in S} x[j] \;=\; \prod_{j\in S}\big(x_0[j] + x_1[j]\big) \;=\; \sum_{I\subseteq S}\Big(g^0_{\pi(I)}(x_0)\Big)\,\Big(g^1_{\pi(S\setminus I)}(x_1)\Big). \tag{5}$$

A concrete example of these notations is a 2-input masked AND gate with inputs $x[0]$ and $x[1]$. For $k = 2$, the power set of $\{0,1\}$ consists of four subsets: $\emptyset, \{0\}, \{1\}, \{0,1\}$, indexed as: $\pi(\emptyset) = 0$, $\pi(\{0\}) = 1$, $\pi(\{1\}) = 2$, $\pi(\{0,1\}) = 3$. The corresponding functions $g^0$ and $g^1$ are summarized below:

| **Subset** $I$ | $\emptyset$ | $\{0\}$ | $\{1\}$ | $\{0,1\}$ |
|---|---|---|---|---|
| $g^0_{\pi(I)}(x_0)$ | 1 | $x_0[0]$ | $x_0[1]$ | $x_0[0]\cdot x_0[1]$ |
| $g^1_{\pi(I)}(x_1)$ | 1 | $x_1[0]$ | $x_1[1]$ | $x_1[0]\cdot x_1[1]$ |

Applying Eq. (5) to the monomial $\prod_{j\in\{0,1\}} x[j]$, we obtain:

$$\begin{aligned}\prod_{j\in\{0,1\}} x[j] &= \sum_{I\subseteq\{0,1\}} g^0_{\pi(I)}(x_0)\cdot g^1_{\pi(\{0,1\}\setminus I)}(x_1)\\ &= \underbrace{g^0_0(x_0)\,g^1_3(x_1)}_{I=\emptyset} + \underbrace{g^0_1(x_0)\,g^1_2(x_1)}_{I=\{0\}} + \underbrace{g^0_2(x_0)\,g^1_1(x_1)}_{I=\{1\}} + \underbrace{g^0_3(x_0)\,g^1_0(x_1)}_{I=\{0,1\}}\\ &= x_1[0]\cdot x_1[1] \;+\; x_0[0]\cdot x_1[1] \;+\; x_0[1]\cdot x_1[0] \;+\; x_0[0]\cdot x_0[1]\\ &= \big(x_0[0]+x_1[0]\big)\big(x_0[1]+x_1[1]\big).\end{aligned}$$

### 2.4.2 Applying TSM to a Single Coordinate

This decomposition is applied to each term in the ANF of $f[m]$. Summing over all terms $S\in\mathcal{M}_m$, we have:

$$f[m](x_0+x_1) \;=\; \sum_{S\in\mathcal{P}_k}\alpha_S\sum_{I\subseteq S} g^0_{\pi(I)}(x_0)\,g^1_{\pi(S\setminus I)}(x_1) \;=\; \sum_{S\in\mathcal{M}_m}\sum_{I\subseteq S} g^0_{\pi(I)}(x_0)\,g^1_{\pi(S\setminus I)}(x_1) \tag{6}$$

The expression in Eq.(6) can be reorganized by subsets $I\in\mathcal{P}_k$ associated with $g^0_{\pi(I)}$. This reorganization will be essential later when we extend the approach to vectorial Boolean functions. In this reorganization, for each $I\in\mathcal{P}_k$, we capture the set of $g^1$ functions that need to be summed and multiplied with the corresponding $g^0_{\pi(I)}$ function.

$$\sum_{S\in\mathcal{M}_m}\sum_{I\subseteq S} g^0_{\pi(I)}(\cdot)\,g^1_{\pi(S\setminus I)}(\cdot) \;=\; \sum_{I\in\mathcal{P}_k}\Big(g^0_{\pi(I)}(\cdot)\;\cdot\sum_{\substack{S\in\mathcal{M}_m\\ I\subseteq S}} g^1_{\pi(S\setminus I)}(\cdot)\Big) \tag{7}$$

Specifically, for each $I\in\mathcal{P}_k$, the set of $g^1$ functions is constructed by selecting all $S\in\mathcal{M}_m$ where $I\subseteq S$, and including the functions $g^1_{\pi(S\setminus I)}$ in this set.

Formally, this set of $g^1$ functions can be defined using the indices:

$$J_{m,\pi(I)} \;=\; \big\{\pi\big(S\setminus I\big)\;\big|\;S\in\mathcal{M}_m,\; I\subseteq S\big\}. \tag{8}$$

The set $J_{m,\pi(I)}$ represents those terms in the ANF of $f[m]$ (denoted by $\mathcal{M}_m$) that contribute to the product decomposition involving $g^0_{\pi(I)}$. The final expression of $f[m]$ is given by:

$$f[m](x_0+x_1) \;=\; \sum_{I\in\mathcal{P}_k}\Big(g^0_{\pi(I)}(\cdot)\cdot\sum_{\substack{S\in\mathcal{M}_m\\ I\subseteq S}} g^1_{\pi(S\setminus I)}(\cdot)\Big) \;=\; \sum_{I\in\mathcal{P}_k}\Big(g^0_{\pi(I)}(\cdot)\cdot\sum_{j\in J_{m,\pi(I)}} g^1_j(\cdot)\Big). \tag{9}$$

If $J_{m,\pi(I)}$ is empty, it implies that the subset $I$ does not contribute to the ANF of $f[m]$ under the specified conditions. Consequently, the sum $\sum_{j\in J_{m,\pi(I)}} g^1_j(\cdot)$ evaluates to zero.

### 2.4.3 First-Order Security

Next, to ensure first-order security, we refresh the outputs of $g^0$ with fresh random bits $r$, where $r=\{r_0,\ldots,r_{2^k-1}\}$. Specifically, the output of each $g^0_{\pi(I)}$ is refreshed by adding the corresponding random bit $r_{\pi(I)}$, leading to

$$f[m](x_0+x_1) \;=\; \sum_{I\in\mathcal{P}_k}\big(\big(g^0_{\pi(I)}(\cdot)+r_{\pi(I)}\big)\cdot\sum_{j\in J_{m,\pi(I)}} g^1_j(\cdot)\big) \;+\; \sum_{I\in\mathcal{P}_k}\big(r_{\pi(I)}\cdot\sum_{j\in J_{m,\pi(I)}} g^1_j(\cdot)\big).$$

We can split this expression into two phases, as depicted in Figure 1:

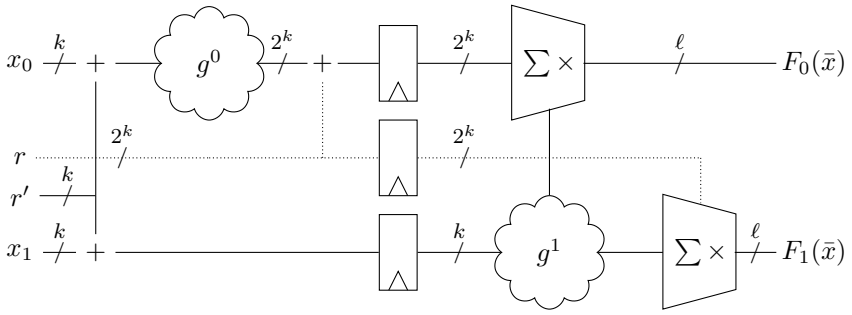



**Figure 1:** Application of TSM to an arbitrary (vectorial) Boolean function described by the set of functions $g^0$ and $g^1$.

(i) computing and refreshing $g^0$ on the first share (with addition of $r$),

(ii) computing $g^1$ on the second share and recombining.

These steps allow us to safely construct the two output shares:

$$\begin{aligned} F_0[m] &= \sum_{I\in\mathcal{P}_k} \Big( \{ g^0_{\pi(I)}(\cdot) + r_{\pi(I)} \} \cdot \sum_{j\in J_{m,\pi(I)}} g^1_j(\cdot) \Big), \\ F_1[m] &= \sum_{I\in\mathcal{P}_k} \Big( \{ r_{\pi(I)} \} \cdot \sum_{j\in J_{m,\pi(I)}} g^1_j(\cdot) \Big). \end{aligned} \tag{10}$$

where the curly brackets indicate storing a value in a register, which prohibits glitch propagation.

Lastly, before processing the gadget, the inputs $x_0, x_1$ themselves are refreshed with randomness $r' \in \mathbb{F}_2^k$ (i.e., $x_0 \leftarrow x_0 + r'$ and $x_1 \leftarrow x_1 + r'$) to ensure composable security.

### 2.4.4 Extending TSM to Vectorial Functions

We now generalize this approach to vectorial functions by ensuring that the masking process is applied efficiently across all coordinate functions $f[m]$, $m \in \{0, \ldots, \ell-1\}$. From Eq.(3), it is evident that all monomials up to degree $k$ can be constructed using the functions $g^0$ and $g^1$. For each coordinate function $f[m]$, $m \in \{0, \ldots, \ell-1\}$, there is a corresponding set $J_{m,\pi(I)}$ determined by the terms that actually appear in $f[m]$, denoted by $\mathcal{M}_m$, see Eq.(8). Consequently, the same $g^0$ and $g^1$ building blocks can be efficiently re-used to extend TSM to a vectorial Boolean function ($f : \mathbb{F}_2^k \to \mathbb{F}_2^\ell$) by applying the above construction to each output coordinate $f[m]$.

To establish correctness, we demonstrate that the sum of the output shares reconstructs the original Boolean function $f$. The randomness $r_{\pi(I)}$, introduced during the refreshing of $g^0_{\pi(I)}$, cancels out between the two shares, leaving only the correctly reconstructed terms from the ANF of $f[m]$:

$$\begin{aligned} F_0[m] + F_1[m] &= \sum_{I\in\mathcal{P}_k} \Big( \{ g^0_{\pi(I)}(\cdot) + r_{\pi(I)} \} + \{ r_{\pi(I)} \} \Big) \cdot \sum_{j\in J_{m,\pi(I)}} g^1_j(\cdot) \\ &= \sum_{I\in\mathcal{P}_k} \Big( g^0_{\pi(I)}(\cdot) \cdot \sum_{j\in J_{m,\pi(I)}} g^1_j(\cdot) \Big) = f[m](x_0 + x_1). \end{aligned} \tag{11}$$

For vectorial functions $f : \mathbb{F}_2^k \to \mathbb{F}_2^\ell$, this reasoning extends independently to each coordinate $f[m]$, $m \in \{0, \ldots, \ell-1\}$. Consequently, the sum of the shares satisfies $F_0(\bar{x}) + F_1(\bar{x}) = f(x_0 + x_1)$.

# 3 Higher-Order Time Sharing Masking (HO-TSM)

We now extend TSM to achieve higher-order security while retaining the benefits of the first-order construction. We begin with a second-order secure AND gate as an illustrative example in Section 3.1. Next, we provide the construction of our higher-order secure method in Section 3.2. In Section 3.3, we prove that our proposed solution is glitch-extended PINI and SNI secure. Finally, we apply our method to develop a two-cycle second-order masked AES S-Box in Section 3.4.

## 3.1 Preliminary Example

Before formally introducing HO-TSM, we illustrate its core concept with a simple example of a second-order secure AND gate. Consider two inputs, $x$ and $y$, for the AND gate, producing an output $z = x \cdot y$. Assume $x$ and $y$ are shared as $\bar{x} = (x_0, x_1, x_2)$ and $\bar{y} = (y_0, y_1, y_2)$, respectively. The HO-TSM approach processes one share of each input per clock cycle while ensuring the isolation of computations using registers, as shown in Figure 2.

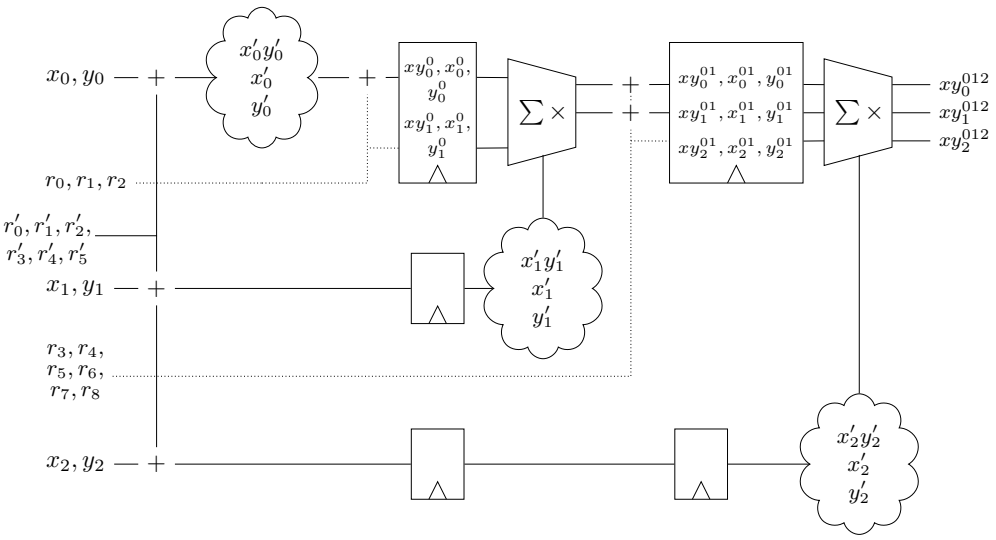



**Figure 2:** Application of HO-TSM$_2$: a second-order secure AND gate.

To ensure composable security, the input shares $(x_0, x_1, x_2)$ and $(y_0, y_1, y_2)$ are initially refreshed with six random bits $\{r'_0, r'_1, r'_2, r'_3, r'_4, r'_5\}$, as follows:

| $\boldsymbol{x' = x'_0 + x'_1 + x'_2}$ | $\boldsymbol{y' = y'_0 + y'_1 + y'_2}$ |
|---|---|
| $x'_0 = x_0 + r'_0 + r'_1$ | $y'_0 = y_0 + r'_3 + r'_4$ |
| $x'_1 = x_1 + r'_0 + r'_2$ | $y'_1 = y_1 + r'_3 + r'_5$ |
| $x'_2 = x_2 + r'_1 + r'_2$ | $y'_2 = y_2 + r'_4 + r'_5$ |

In the first phase, all cross-product combinations of $(x'_0, y'_0, x'_0y'_0)$ are calculated, refreshed using random bits $\{r_0, r_1, r_2\}$, and stored in registers:

| $\boldsymbol{xy^0 = x'_0 \cdot y'_0}$ | $\boldsymbol{x^0 = x'_0}$ | $\boldsymbol{y^0 = y'_0}$ |
|---|---|---|
| $xy_0^0 = x'_0y'_0 + r_0$ | $x_0^0 = x'_0 + r_1$ | $y_0^0 = y'_0 + r_2$ |
| $xy_1^0 = r_0$ | $x_1^0 = r_1$ | $y_1^0 = r_2$ |

In the second phase, all cross-product combinations of $(x'_1, y'_1, x'_1y'_1)$ are computed and combined with the register contents from Phase 1. This phase also increases the number of intermediate shares from two to three:

| $\boldsymbol{xy^{01} = (x'_0 + x'_1) \cdot (y'_0 + y'_1)}$ | $\boldsymbol{x^{01} = (x'_0 + x'_1)}$ | $\boldsymbol{y^{01} = (y'_0 + y'_1)}$ |
|---|---|---|
| $xy_0^{01} = xy_0^0 + x_0^0 \cdot y'_1 + y_0^0 \cdot x'_1 + r_3$ | $x_0^{01} = x_0^0 + r_5$ | $y_0^{01} = y_0^0 + r_7$ |
| $xy_1^{01} = xy_1^0 + x_1^0 \cdot y'_1 + y_1^0 \cdot x'_1 + x'_1y'_1 + r_4$ | $x_1^{01} = x_1^0 + x'_1 + r_6$ | $y_1^{01} = y_1^0 + y'_1 + r_8$ |
| $xy_2^{01} = r_3 + r_4$ | $x_2^{01} = r_5 + r_6$ | $y_2^{01} = r_7 + r_8$ |

In the third phase, cross-product combinations of $(x'_2, y'_2, x'_2y'_2)$ are computed and combined with the intermediate shares from Phase 2 to produce the final outputs:

| $\boldsymbol{xy^{012} = (x'_0 + x'_1 + x'_2) \cdot (y'_0 + y'_1 + y'_2)}$ |
|---|
| $xy_0^{012} = xy_0^{01} + x_0^{01} \cdot y'_2 + y_0^{01} \cdot x'_2$ |
| $xy_1^{012} = xy_1^{01} + x_1^{01} \cdot y'_2 + y_1^{01} \cdot x'_2$ |
| $xy_2^{012} = xy_2^{01} + x_2^{01} \cdot y'_2 + y_2^{01} \cdot x'_2 + x'_2y'_2$ |

This computation requires 15 fresh random bits and 21 registers to store the intermediate shares, with a latency of two clock cycles. While the area and randomness costs of HO-TSM may be significant for a single AND gate, which serves merely as an illustrative example, the method is intended for use with complex non-linear functions, where its efficiency and advantages are most pronounced.

## 3.2 Construction

In this section, we provide a formal description of HO-TSM. The $d$-th order HO-TSM, denoted by HO-TSM$_d$, extends the principles of TSM to higher-order security. To provide a comprehensive understanding, we first present a general overview of HO-TSM. We then explain how to construct HO-TSM$_d$ step by step using a recursive approach.

HO-TSM$_d$ operates on $(d + 1)$ shares and requires $d$ register stages (or equivalently, $(d+1)$ phases). In each phase, the algorithm processes one share, computes all its monomial combinations denoted by $g^i$(functions operating on individual shares), and combines the result with the output from the previous phase stored in the register. This combination is then refreshed to generate an additional output share, except in the final phase, where the combination output serves as the final output of the gadget . The $d$-th register stage maintains a $(d + 1)$-sharing of cross-products using shares $\{0, \ldots, d\}$.

For example, Figure 3 illustrates HO-TSM$_2$. In the first phase, share $x_0$ is processed to compute its monomials, which are refreshed to produce two output shares. In the subsequent phase, $x_1$ is processed and combined with the refreshed outputs from the first phase, resulting in three shares. Finally, in the third phase, share $x_2$ is processed and combined with the outputs of the previous phase to produce the final output shares of HO-TSM$_2$.

An intuitive way to understand the HO-TSM method is to recognize that it is constructed recursively from lower-order HO-TSM instances. As illustrated in Figure 4, HO-TSM$_d$ is built from HO-TSM$_{d-1}$, which securely operates on the first $d$ shares $(x_0, \ldots, x_{d-1})$. A new set of functions, $g^d$, is then defined to operate exclusively on the additional share $x_d$. Finally, the outputs of HO-TSM$_{d-1}$ are combined with the outputs of $g^d$, merging the partial results to produce a secure $(d + 1)$-share masking of the target function.

Consider an arbitrary Boolean function

$$f : \mathbb{F}_2^k \rightarrow \mathbb{F}_2^\ell, \quad x = \big(x[0], \ldots, x[k-1]\big) \mapsto f\big(x[0], \ldots, x[k-1]\big). \tag{12}$$

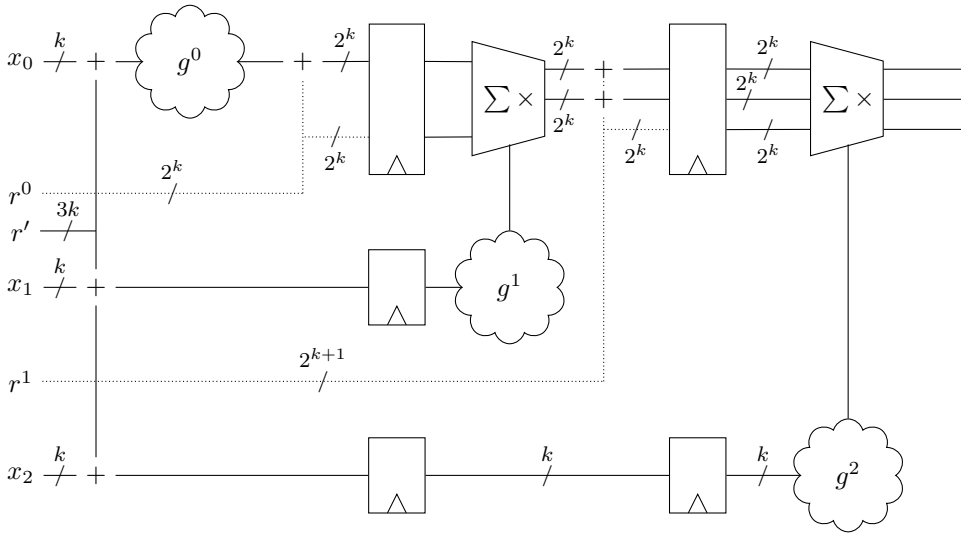



**Figure 3:** Application of HO-TSM$_2$ to an arbitrary (vectorial) Boolean function described by the functions $g^0$, $g^1$, and $g^2$.

We denote its $(d+1)$-share masking by $\bar{F} : \mathbb{F}_2^{k(d+1)} \to \mathbb{F}_2^{\ell(d+1)}, \bar{x} \mapsto \big(F_0(\bar{x}), \ldots, F_d(\bar{x})\big)$, such that $F_0(\bar{x}) + F_1(\bar{x}) + \cdots + F_d(\bar{x}) = f\big(x_0 + x_1 + \cdots + x_d\big)$.

The description of HO-TSM builds on the concepts introduced in TSM, including the decomposition of terms, refreshing of shares, and the reformulation of the algebraic normal form (ANF) for masked computation. It is recommended to review Section 2.4 beforehand, as many of these concepts are directly reused and extended in this section.

Following the same strategy as in Section 2.4, we first focus on a single coordinate function $f[m]$, $m \in \{0, \ldots, \ell - 1\}$, and subsequently generalize to the full vectorial case. Extending to all $\ell$ outputs involves applying the same procedure efficiently to each coordinate function.

### 3.2.1 Decomposing Product Terms

In analogy to how TSM uses $g^0$ and $g^1$, we define two sets of functions:

(i) A set $h$ operating on the first $d$ share domains (0 to $d-1$),

(ii) A set $g^d$ operating on the last (i.e., $d$-th) share domain.

The functions $h_{\pi(I)}$ compute cross-products across the first $d$ shares, while $g^d_{\pi(I)}$ focuses on the monomials involving the $d$-th share exclusively. Concretely:

$$h_{\pi(I)}(x_0, \ldots, x_{d-1}) \;=\; \prod_{i \in I} \Big( x_0[i] + x_1[i] + \cdots + x_{d-1}[i] \Big), \quad g^d_{\pi(I)}(x_d) \;=\; \prod_{i \in I} x_d[i]. \quad (13)$$

Here, $I \subseteq \{0, \ldots, k-1\}$, and $\pi$ is an indexing function over the power set of $\{0, \ldots, k-1\}$. To simplify notation, we use $h_{\pi(I)}(\cdot)$ to represent the function $h_{\pi(I)}$ applied to the first $d$ shares. Similarly, $g^d_{\pi(I)}(\cdot)$ refers to $g^d_{\pi(I)}$ evaluated using the final share $x_d$. More details on the notations used can be found in Section 2.4.

As in the two-share case for first-order TSM, each shared term in $f[m]$ (of the form $\prod_{j \in J} x[j]$) can be decomposed into a sum of products that splits between the $d$-th share

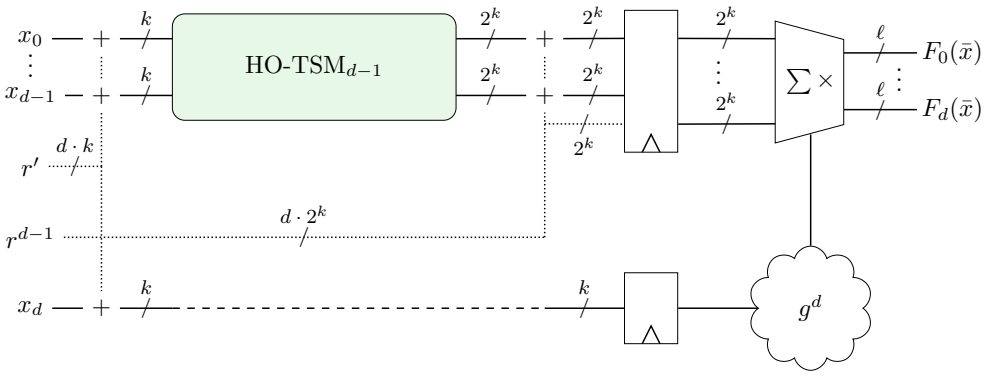



**Figure 4:** Recursive method of HO-TSM$_d$.

and the sum of the previous $d$ shares. Formally, for $J \subseteq \{0, \ldots, k-1\}$,

$$\prod_{j \in J} x[j] = \prod_{j \in J} ((x_0[j] + \cdots + x_{d-1}[j]) + x_d[j]) = \sum_{I \subseteq J} h_{\pi(I)}(x_0, \ldots, x_{d-1}) \cdot g^d_{\pi(J \setminus I)}(x_d). \tag{14}$$

To illustrate, consider the second-order masked AND gate introduced in Section 3.1, representing $x[0] \cdot x[1]$ with $k = 2$. The corresponding $h$-functions and $g^2$-functions for all subsets $I \in \mathcal{P}_2$ are summarized below:

| **Subset** $I$ | $\emptyset$ $(\pi(I) = 0)$ | $\{0\}$ $(\pi(I) = 1)$ | $\{1\}$ $(\pi(I) = 2)$ | $\{0,1\}$ $(\pi(I) = 3)$ |
|---|---|---|---|---|
| $h_{\pi(I)}(x_0, x_1)$ | 1 | $(x_0[0] + x_1[0])$ | $(x_0[1] + x_1[1])$ | $(x_0[0] + x_1[0])(x_0[1] + x_1[1])$ |
| $g^2_{\pi(I)}(x_2)$ | 1 | $x_2[0]$ | $x_2[1]$ | $x_2[0] \cdot x_2[1]$ |

By applying Eq. (14), we decompose the term $\prod_{j \in \{0,1\}} x[j]$, where each $x[j]$ is $\big(x_0[j] + x_1[j] + x_2[j]\big)$:

$$\begin{aligned}
\prod_{j \in \{0,1\}} \big(x[j]\big) &= \sum_{I \subseteq \{0,1\}} h_{\pi(I)}(x_0, x_1) \cdot g^2_{\pi(\{0,1\} \setminus I)}(x_2) \\
&= \underbrace{h_0(\cdot)\, g_3^2(\cdot)}_{I=\emptyset} + \underbrace{h_1(\cdot)\, g_2^2(\cdot)}_{I=\{0\}} + \underbrace{h_2(\cdot)\, g_1^2(\cdot)}_{I=\{1\}} + \underbrace{h_3(\cdot)\, g_0^2(\cdot)}_{I=\{0,1\}} \\
&= \big(x_0[0] + x_1[0] + x_2[0]\big)\big(x_0[1] + x_1[1] + x_2[1]\big).
\end{aligned}$$

### 3.2.2 Decomposing a Single Coordinate

Analogous to Eq. (9) in the first-order TSM case, the decomposition in Eq. (14) is applied to each term in the algebraic normal form of $f[m]$. By utilizing the same sets $J_{m,\pi(I)}$ as defined in Eq. (8), we express:

$$f[m]\Big((x_0 + \cdots + x_{d-1}) + x_d\Big) = \sum_{I \in \mathcal{P}_k} \Big(h_{\pi(I)}(x_0, \ldots, x_{d-1}) \cdot \sum_{j \in J_{m,\pi(I)}} g_j^d(x_d)\Big). \tag{15}$$

The sets $J_{m,\pi(I)}$ from TSM can be directly reused in HO-TSM because they are constructed solely based on the structure of the ANF of $f[m]$, which remains unchanged regardless of the number of shares.

### 3.2.3 Recursive Construction of HO-TSM$_d$

Having decomposed $f[m]$ into the functions $h$ and $g^d$, we now outline the recursive steps required to construct HO-TSM$_d$. These steps detail how the outputs of $h$ and $g^d$ are securely computed and combined to achieve the $(d+1)$-share masking of $f[m]$. As illustrated in Figure 4, a HO-TSM$_{d-1}$ block securely implements each of the $h_{\pi(I)}$ functions, shared across $d$ shares, while the $g^d$ cloud computes all the functions $g^d_{\pi(I)}$. The recursive process proceeds as follows:

1. The input shares $\big(x_0, \ldots, x_d\big)$ are first refreshed with randomness $r' = \big(r'_0, \ldots, r'_d\big)$ to ensure composable security. Each $r'_i \in \mathbb{F}_2^k$ for $i \in \{0, \ldots, d-1\}$ is an independent random value, while $r'_d = \sum_{i=0}^{d-1} r'_i$. The refreshed shares are $\big(x'_0, \ldots, x'_d\big)$, with $x'_i = x_i + r'_i$ for $i \in \{0, \ldots, d\}$.[2]

2. The first $d$ refreshed input shares, $(x'_0, \ldots, x'_{d-1})$, are securely processed by the HO-TSM$_{d-1}$ block. This block computes the outputs of all $h_{\pi(I)}$ functions (as defined in Eq. (13)), producing intermediate $d$-shared outputs:

$$\overline{h_{\pi(I)}}(x'_0, \ldots, x'_{d-1}) = \big(h_{\pi(I)}(\cdot)|_0, h_{\pi(I)}(\cdot)|_1, \ldots, h_{\pi(I)}(\cdot)|_{d-1}\big),$$

   where $h_{\pi(I)}(\cdot)\big|_i$ represents the $i$-th share of the function $h_{\pi(I)}$, evaluated over the first $d$ shares $(x'_0, \ldots, x'_{d-1})$.

3. Each of the $d$ shares $h_{\pi(I)}(\cdot)\big|_i$ is refreshed using a dedicated random word $r_i^{d-1}$, where each $r_i^{d-1}$ consists of $2^k$ bits (one random bit per subset $\pi(I)$). Altogether, $d \cdot 2^k$ fresh random bits are required in this step. Specifically, $h_{\pi(I)}(\cdot)\big|_i$ is refreshed as $\big(h_{\pi(I)}(\cdot)\big|_i + r_i^{d-1}[\pi(I)]\big)$. A corresponding new share $r_d^{d-1} = \sum_{i=0}^{d-1} r_i^{d-1}$ is also generated in this step.

4. The refreshed outputs

$$\Big\{ h_{\pi(I)}(\cdot)\big|_i + r_i^{d-1}[\pi(I)] \Big\} \quad \text{for } i = 0, \ldots, d-1$$

   along with the newly generated share $r_d^{d-1}$, are stored in a register. Consequently, the register now contains $(d+1)$ shares for each $h_{\pi(I)}$ function. This process ensures that $h_{\pi(I)}(\cdot)$, which was previously shared across $d$ shares, is now securely represented in a $(d+1)$-share domain.

5. In the last phase, the remaining refreshed input share $x'_d$ is processed to compute the outputs of the $g^d_{\pi(I)}$ functions. These results are then combined with the refreshed outputs of $h_{\pi(I)}$ functions, which were previously computed by the HO-TSM$_{d-1}$ block, to produce the final $(d+1)$-shared outputs $\big(F_0[m], \ldots, F_d[m]\big)$. The output equations are as follows:

$$\begin{aligned} F_i[m] &= \sum_{I \in \mathcal{P}_k} \Big( \Big\{ h_{\pi(I)}(\cdot)\big|_i + r_i^{d-1}[\pi(I)] \Big\} \cdot \sum_{j \in J_{m,\pi(I)}} g_j^d(x'_d) \Big) && \text{for } i < d, \\ F_d[m] &= \sum_{I \in \mathcal{P}_k} \Big( \Big\{ r_d^{d-1}[\pi(I)] \Big\} \cdot \sum_{j \in J_{m,\pi(I)}} g_j^d(x'_d) \Big) && \text{for } i = d. \end{aligned} \tag{16}$$

The recursion proceeds iteratively, reducing the order at each step until it reaches the base case of first-order masking, HO-TSM$_1$ (i.e., TSM), which is securely implemented using the original first-order method of [VDB+24].

[2] Due to the recursive structure, the input shares may be re-shared multiple times. Although one could attempt to reduce this overhead, it is not the main source of randomness usage and the current approach allows for a clean recursion and straightforward security arguments.

### 3.2.4 Extending HO-TSM to Vectorial Functions

Extending HO-TSM to vectorial Boolean functions ($f : \mathbb{F}_2^k \to \mathbb{F}_2^\ell$) involves applying the decomposition described for a single coordinate function $f[m]$, $m \in \{0, \ldots, \ell - 1\}$, to each coordinate. The key observation is that the sets $J_{m,\pi(I)}$ (as defined in Eq. (8)), which determine which $g^d$-functions are summed together for each coordinate, need to be updated to reflect the ANF of each $f[m]$. Importantly, the sets $J_{m,\pi(I)}$ constructed for TSM can be directly reused in HO-TSM, as they depend solely on the structure of the ANF of $f[m]$. Furthermore, the refreshed and registered shares of $h_{\pi(I)}$, along with the functions $g^d_{\pi(I)}$, which decompose the terms across the $(d+1)$-share domain, are universal and can be reused across all coordinates of $f$. This ensures an efficient extension of HO-TSM to vectorial functions by leveraging the same decomposition principles, Eq. (15), and building blocks, Eq. (13), for all coordinates.

**Table 1:** Comparison of algorithmic costs of HO-TSM and GLM for order $d$ when masking a function $\mathbb{F}_2^k \to \mathbb{F}_2^k$ function of algebraic degree $k-1$.

| Method | # Registers | # Register Layers | # Random Bits |
|---|---|---|---|
| HO-TSM$_d$ | $2^k \frac{d(d+3)}{2} + k\frac{d(d+1)}{2}$ | $d$ | $2^k \frac{d(d+1)}{2} + k\frac{d(d+1)}{2}$ |
| GLM [GIB18] | $(d+1)^k k$ | 1 | $(d+1)^k k$ |

We provide the algorithmic costs of number of registers, register stages, and random bits in Table 1. Comparing HO-TSM with other methods is difficult as it easily becomes dependent on the actual function that is implemented. Nevertheless, we compare to the single cycle case of the GLM method where we clearly see that we trade-off the number of registers and randomness for more register layers. A better view of the cost of HO-TSM is given in Section 3.4 where it is applied to create a second-order masked AES S-box.

## 3.3 Correctness and Security

We begin with the correctness and security proof for HO-TSM$_d$ and then provide SILVER results for some small examples.

**Theorem 1.** *Any circuit secured by HO-TSM$_d$ is correct.*

*Proof.* We establish the correctness of HO-TSM$_d$ by induction on $d$, the order of the masking.

- **Base Case** ($d = 1$): When $d = 1$, HO-TSM$_1$ is simply the first-order two-share masking (TSM). As shown in Section 2.4 and established in [VDB+24], we have, for each output coordinate $m$: $F_0[m] + F_1[m] = f[m](x_0 + x_1)$, and therefore $F_0(\bar{x}) + F_1(\bar{x}) = f(x_0 + x_1)$. Hence, correctness holds for $d = 1$.
- **Inductive Hypothesis:** Assume that HO-TSM$_{d-1}$, which produces $d$-shared outputs, satisfies correctness. That is, for some $(d-1) \geq 1$, the outputs $F_0[m], \ldots, F_{d-1}[m]$ satisfy $\sum_{i=0}^{d-1} F_i[m] = f[m](x_0 + x_1 + \cdots + x_{d-1})$, meaning each coordinate function $f[m]$ is correctly reconstructed from the sum of all $d$-output shares in the HO-TSM$_{d-1}$ design.
- **Inductive Step** ($d$): To complete the proof, we verify that HO-TSM$_d$ correctly produces $(d+1)$-shared outputs satisfying the required property. By construction:

  1. The input shares $(x_0, \ldots, x_d)$, refreshed with randomness $r' = (r'_0, \ldots, r'_d)$ for composable security, do not affect correctness as the refreshing process preserves $\sum_{i=0}^{d} x_i$. This is because $\sum_{i=0}^{d} r'_i = \sum_{i=0}^{d-1} r'_i + r'_d = 0$.

2. HO-TSM$_{d-1}$ is applied to the first $d$ refreshed shares $(x'_0, \ldots, x'_{d-1})$, yielding intermediate outputs:

$$\overline{h_{\pi(I)}(\cdot)} = \left(h_{\pi(I)}(\cdot)\big|_0, h_{\pi(I)}(\cdot)\big|_1, \ldots, h_{\pi(I)}(\cdot)\big|_{d-1}\right),$$

whose sum of shares equals $\sum_{i=0}^{d-1} h_{\pi(I)}(\cdot)\big|_i = h_{\pi(I)}(x'_0, \ldots, x'_{d-1})$. By the inductive hypothesis, HO-TSM$_{d-1}$ is correct.

3. Each share $h_{\pi(I)}(\cdot)\big|_i$ is refreshed with randomness $r_i^{d-1}[\pi(I)]$, producing $\left\{h_{\pi(I)}(\cdot)\big|_i + r_i^{d-1}[\pi(I)]\right\}$. Adding the refreshed $d$-shares and the extra random share $r_d^{d-1} = \sum_{i=0}^{d-1} r_i^{d-1}$, the randomness cancels out, recovering: $\sum_{i=0}^{d-1}\left(h_{\pi(I)}(\cdot)\big|_i + r_i^{d-1}[\pi(I)]\right) + r_d^{d-1}[\pi(I)] = h_{\pi(I)}(x'_0, \ldots, x'_{d-1})$.

4. The functions $g^d_{\pi(I)}$, which operate exclusively on the last refreshed share $x'_d$, are then computed. These results are combined with the refreshed shares of $h_{\pi(I)}(\cdot)$, producing the final $(d+1)$-shared outputs $(F_0[m], \ldots, F_d[m])$. Summing the outputs gives:

$$\begin{aligned}\sum_{i=0}^{d} F_i[m] &= \sum_{I \in \mathcal{P}_k} \left(h_{\pi(I)}(x'_0, \ldots, x'_{d-1}) \cdot \sum_{j \in J_{m,\pi(I)}} g_j^d(x'_d)\right) \\ &= f[m](x'_0 + \cdots + x'_d) = f[m](x_0 + \cdots + x_d).\end{aligned}$$

This correctness argument applies independently to all coordinates $f[m]$, $m \in \{0, \ldots, \ell - 1\}$, as the sets $J_{m,\pi(I)}$ ensure proper reconstruction for each $f[m]$ based on its ANF. The final $(d+1)$-shared outputs for the vectorial function $f$ are obtained by concatenating the outputs for all coordinates, ensuring that $F_0(\bar{x}) + F_1(\bar{x}) + \cdots + F_d(\bar{x}) = f(x_0 + x_1 + \cdots + x_d)$, where $F_i(\bar{x}) = \left(F_i[0], \ldots, F_i[\ell - 1]\right)$ for $i \in \{0, \ldots, d\}$.

□

**Theorem 2.** *Any circuit secured by HO-TSM$_d$ is $d^{th}$-order glitch-extended PINI and SNI.*

*Proof.* We prove the theorem by induction on $d$.

- **Base Case** ($d = 1$): From [VDB+24], it is established that the method is secure for $d = 1$. We observe in the proof that the simulation can be performed from scratch without giving any input shares when probing an output.
- **Induction Hypothesis:** Assume that any circuit secured by HO-TSM$_{d-1}$ is $(d-1)^{\text{th}}$-order glitch-extended PINI and SNI.
- **Induction Step** ($d$): We aim to show that if the theorem holds for HO-TSM$_{d-1}$, then it also holds for HO-TSM$_d$. Please refer to Figure 4 for reference. To demonstrate $d^{\text{th}}$-order PINI and SNI security, we use a simulation-based argument. First, we categorize all possible probe locations as follows:

  1. Probing the HO-TSM$_{d-1}$ block.
  2. Probing the $i^{\text{th}}$ share in the upper register.
  3. Probing the lower register.
  4. Probing the $i^{\text{th}}$ output share.

  Next, we construct the simulator $\mathcal{S}$. By the induction hypothesis, the HO-TSM$_{d-1}$ block is $(d-1)^{\text{th}}$-order PINI and SNI secure, and we can utilize its simulator $\mathcal{S}'$. Depending on the probe positions, we provide the following shares to $\mathcal{S}$:

– Probes in Group 1: We provide $\mathcal{S}$ the same input which would be given to $\mathcal{S}'$.
– Probes in Group 2: We provide the simulator with nothing.
– Probes in Group 3: We provide the simulator with $x_d$.
– Probes in Group 4: We provide the simulator with nothing.

With the above input shares, the simulator $\mathcal{S}$ can perfectly simulate the probes as follows:

– For any probe in Group 1: By the induction hypothesis, $\mathcal{S}'$ perfectly simulates this probe. When $d$ probes are placed in Group 1 (or 2), since the share $x_d$ is excluded, the simulator can be given $(x_0, \ldots, x_{d-1})$ to perfectly simulate the entire HO-TSM$_{d-1}$ block.
– For any probe in Group 2: A probe in this group views a single output of the HO-TSM$_{d-1}$ block. The simulator $\mathcal{S}'$ can perfectly simulate this output due to its $(d-1)^{\text{th}}$-order PINI and SNI security. The simulator $\mathcal{S}$ also creates the randomness $r_i^{d-1}$. If $i = d$, $\mathcal{S}$ creates all random shares $r_i^{d-1}$ without needing to simulate any output of the HO-TSM$_{d-1}$ block.
– For any probe in Group 3: The simulator has access to $x_d$ and creates the randomness $r'_d$.
– For any probe in Group 4: A probe in this group views one share of the upper register. Due to the randomness $r_i^{d-1}$, each value in the upper register can be simulated as uniform randomness unless a probe has already been placed in that position in Group 2 or when a probe was placed on $r_d^{d-1}$. In such cases, the simulation follows the logic of Group 2. Additionally, a probe in this group also views the lower register value. If a probe was placed in Group 3, the simulator has $x_d$ and can simulate $g^d$. Furthermore, if $d$ outputs of the HO-TSM$_{d-1}$ block need to be simulated, $r'_d$ is also observed. However, since the adversary has only $d$ probes, no probe could have been placed in Group 4 simultaneously. Otherwise, $r'_d = \sum_{i=0}^{d-1} r'_i$ remains unobserved, and the lower register can be simulated as uniform random, with $g^d$ simulated appropriately.

The above demonstrates that the construction achieves $d^{\text{th}}$-order PINI and SNI security. Since both the base case and the induction step have been established, the theorem holds for all $d \geq 1$.

□

**Formal Verification.** SILVER [KSM20] is a formal verification tool that evaluates the security and composability of masked gadgets against several security notions such as probing security, Non-Interference (NI), Strong Non-Interference (SNI), and Probe-Isolating Non-Interference (PINI). We applied our HO-TSM method to create two small examples for different orders of security: a HO-TSM$_2$ AND gate, introduced in Section 3.1, and a HO-TSM$_3$ AND gate. Our two examples successfully pass all second-order and third-order tests, respectively, under all security notions (probing, NI, SNI, PINI) in both the standard and robust probing models. This confirms that our two gadgets are indeed (2-PINI,2-SNI) and (3-PINI,3-SNI) secure.

## 3.4 Application: Second-Order Masked AES S-Box

An unprotected AES S-Box takes eight input bits and produces eight output bits, described as $S : \mathbb{F}_2^8 \to \mathbb{F}_2^8$. The coordinate functions, when presented in their algebraic normal form, include all terms up to the seventh degree, in total 254 terms. To mask the AES S-Box

using HO-TSM$_2$, we begin by computing the sharings for all 254 terms by processing each share individually in three phases, see Figure 3. These sharings are then commonly reused across all eight coordinate functions.

In total, for the first stage, we require 254 random bits ($r^0$) and 508 (254×2) registers. In the second stage, we require 508 (254×2) random bits ($r^1$) and 762 (254×3) registers. Additionally, 24 random bits ($r'$) are required for the initial input refresh. The total utilization cost for the second-order secure AES S-Box, including the cost for combinatorial logic, is summarized in Table 2.

**Table 2:** Utilization results of low-latency second-order masked AES S-Boxes.

| Design | Method | Area (kGE) | Random bits | Cycles | Library |
|---|---|---|---|---|---|
| ***This work*** | **HO-TSM** | **33.7**[a] / **48.6**[b] | **786** | **2** | **NanGate45** |
| [GIB18] | GLM | 57.1 | 4446 | 2 | UMC90 |

[a] `compile_ultra -no_autoungroup -no_boundary_optimization`
[b] `compile -exact_map`

The synthesis results are gathered using the NanGate 45nm Open Cell Library [NAN]. We use Synopsys DC Compiler v2021.06 for Synthesis and provide results for two different sets of options. The first set of options (`compile_ultra -no_autoungroup -no_boundary_optimization`) aims for maximum logic optimization while making sure that no logic is moved across the register layer by separating the registers from logic by placing them in distinct modules. The second set of options (`compile -exact_map`) aims for a direct mapping of our design to logic. The area numbers were gathered by synthesizing our designs with a target frequency of 100 MHz and do not include the area cost of generating the required randomness. Our two-cycle second-order AES S-Box has both lower area as well as randomness cost when compared to the GLM design proposed by Gross *et al.* [GIB18].

# 4 An Area-Latency Trade-Off

In this section, we present a trade-off for HO-TSM which notably decreases both the area and randomness costs while increasing the latency by only a single cycle. In Section 4.1, we introduce our trade-off construction with a toy example for ease of understanding. Section 4.2 details the construction process for arbitrary functions. In Section 4.3, we provide proof that our trade-off construction achieves $d^{th}$-order PINI and SNI security. Sections 4.4 and 4.5 demonstrate the practical application of our method by constructing first- and second-order secure AES S-Boxes, showing improvements in reduced randomness and area. Finally, in Section 4.6, we construct a first-order round-based AES-128 and discuss the associated implementation costs.

## 4.1 Preliminary Example

We provide a toy example to illustrate the area-latency trade-off construction. While this approach is better suited for functions with a larger number of inputs—such as the AES S-Box presented later in this section, which better highlights the efficiency gains—a smaller function simplifies the explanation and aids understanding. Consider a Boolean function $fn$ that takes four input bits and produces one output:

$$fn(a,b,c,d) = a + d + ab + ac + bc + cd + abc + acd + abcd.$$

First, we partition the inputs into two subsets: $P = \{a,b\}$ and $Q = \{c,d\}$. A function $gn$ is applied to each subset to generate the three outputs: $gn(P) = \{a,b,ab\}$ and

$gn(Q) = \{c, d, cd\}$. To securely compute these outputs, we apply HO-TSM$_d$ to the two instances of $gn$, thereby producing the shared outputs $P'$ and $Q'$, as shown in Eq. (17):

$$\begin{aligned} P'_i &= \text{HO-TSM}_d(gn(P)) = \{(a)_i, (b)_i, (ab)_i\}, \quad i \in \{0, 1, \ldots, d\}, \\ Q'_j &= \text{HO-TSM}_d(gn(Q)) = \{(c)_j, (d)_j, (cd)_j\}, \quad j \in \{0, 1, \ldots, d\}. \end{aligned} \tag{17}$$

Next, we generalize the Domain-Oriented Masking (DOM) multiplier introduced by Gross *et al.*[GMK16] to combine the two HO-TSM$_d$ blocks and compute the shared output of $fn$. Specifically, we define $(d+1)^2$ domains and perform share-wise multiplication for the terms in $fn$, distributing the results to their respective domains. For $d = 1$, the masked terms of $fn$ are distributed across the domains as follows:

$$\begin{aligned} fn_{\{0,0\}} &= (a)_0 \cdot (c)_0 + (b)_0 \cdot (c)_0 + (ab)_0 \cdot (c)_0 + (a)_0 \cdot (cd)_0 + (ab)_0 \cdot (cd)_0 \\ &\quad + (a)_0 + (d)_0 + (ab)_0 + (cd)_0, \\ fn_{\{0,1\}} &= (a)_0 \cdot (c)_1 + (b)_0 \cdot (c)_1 + (ab)_0 \cdot (c)_1 + (a)_0 \cdot (cd)_1 + (ab)_0 \cdot (cd)_1, \\ \hline fn_{\{1,0\}} &= (a)_1 \cdot (c)_0 + (b)_1 \cdot (c)_0 + (ab)_1 \cdot (c)_0 + (a)_1 \cdot (cd)_0 + (ab)_1 \cdot (cd)_0, \\ fn_{\{1,1\}} &= (a)_1 \cdot (c)_1 + (b)_1 \cdot (c)_1 + (ab)_1 \cdot (c)_1 + (a)_1 \cdot (cd)_1 + (ab)_1 \cdot (cd)_1 \\ &\quad + (a)_1 + (d)_1 + (ab)_1 + (cd)_1. \end{aligned} \tag{18}$$

After computing the domains, we refresh them using $d(d+1)/2$ random bits, ensuring that each off-diagonal domain $fn_{\{i,j\}}$ is symmetrically refreshed with the same random bit. The domains are then registered and compressed to produce the $(d+1)$ output shares $\{fn_0, fn_1, \ldots, fn_d\}$. For $d = 1$, the final output shares $\{fn_0, fn_1\}$ are computed as:

$$\begin{aligned} fn_0 &= \Big\{fn_{\{0,0\}}\Big\} + \Big\{fn_{\{0,1\}} + r\Big\}, \\ fn_1 &= \Big\{fn_{\{1,1\}}\Big\} + \Big\{fn_{\{1,0\}} + r\Big\}, \end{aligned} \quad \text{where } r \xleftarrow{\$} \mathbb{F}_2.$$

**Formal Verification.** In Section 4.3, we show that our DOM-based multiplication is sufficient to ensure composable security, due to the PINI and SNI composability properties of HO-TSM. To further validate the construction, we implemented the function $fn$ for $d = 1$ and $d = 2$ and tested both implementations using SILVER. The two circuits passed all first- and second-order probing, NI, SNI, and PINI tests, respectively, in the standard and robust probing models, confirming the security of the construction.

## 4.2 Construction

The area and randomness requirements for HO-TSM$_d$ scale with the number of inputs, as detailed in Table 1. To improve efficiency, a trade-off is made by splitting the input space of a function into two (preferably equal-sized) parts and applying a HO-TSM$_d$ block to each. This approach provides a square root improvement in the area and randomness overhead for each block. However, an additional phase is necessary to combine the outputs of the two HO-TSM$_d$ blocks, which results in a one-cycle latency gain.

For an arbitrary Boolean function $fn : \mathbb{F}_2^k \to \mathbb{F}_2^\ell$, the construction applied to $fn$ to provide $d^{th}$-order security proceeds as follows:

1. Let $gn : \mathbb{F}_2^n \to \mathbb{F}_2^{2^n}$ be a function that enumerates all monomials of its $n$-bit input. Concretely, for each subset $I \subseteq \{0, \ldots, n-1\}$, the corresponding output coordinate $gn[\pi(I)]$ is the product $\prod_{i \in I} x[i]$. Thus, $gn$ outputs every possible product of the input variables, including each input on its own.

2. Referring to Figure 5, we take a shared input $x = (x[0], x[1], \ldots, x[k-1])$, such that $\bar{x} = (x_0, x_1, \ldots, x_d)$, and partition it into two disjoint subsets: : $P = \{x[0], \ldots,$

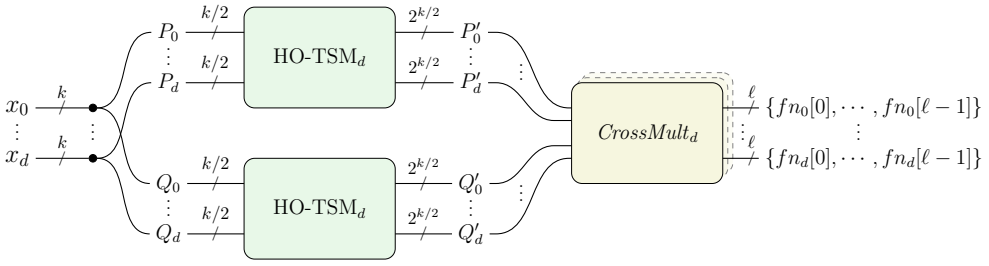



**Figure 5:** An area-latency trade-off construction for $fn$.

$x[\lfloor k/2 \rfloor - 1]\}$ and $Q = \{x[\lfloor k/2 \rfloor], \ldots, x[k-1]\}$. Each subset is processed by a secure implementation of $gn$ using a HO-TSM$_d$ block. In other words, one HO-TSM$_d$ block takes the shares of $P$ and produces shared outputs $P' = (P'_0, P'_1, \cdots, P'_d)$, while another HO-TSM$_d$ instance takes as input $Q$ and produces $Q' = (Q'_0, Q'_1, \cdots, Q'_d)$.

An indexing function $\pi_P$ maps each $I \subseteq \{0, \ldots, \lfloor k/2 \rfloor - 1\}$ to a unique integer in $\{0, \ldots, 2^{k/2} - 1\}$. Consequently, $P'[\pi_P(I)]$ represents the $(d+1)$-shared representation of $\prod_{i \in I} x[i]$. Similarly, an indexing function $\pi_Q$ is defined for subsets $J \subseteq \{\lfloor k/2 \rfloor, \ldots, k-1\}$, with $Q'[\pi_Q(J)]$ representing shares of $\prod_{j \in J} x[j]$.

3. We introduce a masked multiplier $CrossMult_d$(Algorithm 1), which then takes as inputs $P'$ and $Q'$ and produces the shared outputs of a coordinate function $fn[m] = (fn_0[m], fn_1[m], \ldots, fn_d[m])$, where $m \in \{0, 1, \ldots, \ell - 1\}$. $CrossMult_d$ involves $(d+1)^2$ domains, denoted as $fn_{\{i,j\}}[m]$, where each domain operates on a pair of share domains, $P'_i$ and $Q'_j$. The domains are then refreshed, registered, and compressed to generate $(d+1)$ output shares of $fn[m]$.

4. To compute all $\ell$ coordinate functions of $fn$, $CrossMult_d$ is instantiated in parallel $\ell$ times, producing the output shares $(fn_0[m], fn_1[m], \ldots, fn_d[m])_{m=0}^{\ell-1}$.

## 4.3 Correctness and Security

We first establish the correctness of the area-latency trade-off construction, followed by its security proof.

**Theorem 3.** *Any circuit secured by the area-latency trade-off construction is correct.*

*Proof.* The correctness of HO-TSM$_d$ has already been proven in Theorem 1. Therefore, we assume that the outputs $P'$ and $Q'$ from the HO-TSM$_d$ blocks are correct and proceed to prove the correctness of $CrossMult_d$.

We begin by initializing every domain $fn_{\{i,j\}}[m]$ to zero. Next, the set of indices $S$ for each term in $fn[m]$ must lie in one of two categories. In the first category, the indices of the term appear across both subsets $P = \{0, \ldots, \lfloor k/2 \rfloor - 1\}$ and $Q = \{\lfloor k/2 \rfloor, \ldots, k-1\}$. In that case, there exist partial terms $\bar{p} = P'[\pi_P(J_1)]$ and $\bar{q} = Q'[\pi_Q(J_2)]$ such that $\prod_{i \in S} x[i] = p \times q$. Expressing $\bar{p} = (p_0, \ldots, p_d)$ and $\bar{q} = (q_0, \ldots, q_d)$ in their shared form, the algorithm adds $p_i \cdot q_j$ into the domain $fn_{\{i,j\}}[m]$ for all $i, j$, including the diagonal when $i = j$.

In the second category, the term lies entirely within one subset (either $P'$ or $Q'$), so the partial term is simply $\bar{p}$ or $\bar{q}$, and the algorithm places $p_i$ or $q_j$ only in the diagonal domain $\{i, i\}$ or $\{j, j\}$.

After these updates, each domain $\{i, j\}$ is combined with a random bit $r''_{\{i,j\}}$, which is also added to $fn_{\{j,i\}}[m]$. Because each $r''_{\{i,j\}}$ appears exactly twice symmetrically, these bits cancel out when all domains are summed together. Finally, every output $fn_i[m]$ is

**Algorithm 1** *CrossMult*$_d$

**Input:** $P' = (P'_0, P'_1, \cdots, P'_d), Q' = (Q'_0, Q'_1, \cdots, Q'_d)$
**Output:** $fn[m] = (fn_0[m], fn_1[m], \cdots, fn_d[m])$

**for** $i = 0$ **to** $d$ **do** ▷ Initialize relevant domains for $fn$[m]
  **for** $j = 0$ **to** $d$ **do**
    $fn_{\{i,j\}}[m] \leftarrow 0$
  **end for**
**end for**

**for** each $S \in \mathcal{M}_m$ **do** ▷ $\mathcal{M}_m$ is the set of all terms in $fn[m]$, see Eq.(2).
  $J_1 \leftarrow S \cap \{0, \ldots, \lfloor k/2 \rfloor - 1\}$
  $J_2 \leftarrow S \cap \{\lfloor k/2 \rfloor, \ldots, k-1\}$
  **if** $J_1 \neq \emptyset$ **and** $J_2 \neq \emptyset$ **then**
    $\bar{p} \leftarrow P'[\pi_P(J_1)]$ ▷ $\bar{p} = (p_0, p_1, \ldots, p_d)$
    $\bar{q} \leftarrow Q'[\pi_Q(J_2)]$ ▷ $\bar{q} = (q_0, q_1, \ldots, q_d)$
    **for** $i = 0$ **to** $d$ **do**
      **for** $j = 0$ **to** $d$ **do**
        $fn_{\{i,j\}}[m] \leftarrow fn_{\{i,j\}}[m] + p_i \cdot q_j$ ▷ Update corresponding domains
      **end for**
    **end for**
  **else if** $J_1 = \emptyset$ **then**
    $\bar{q} \leftarrow Q'[\pi_Q(J_2)]$ ▷ $\bar{q} = (q_0, q_1, \ldots, q_d)$
    **for** $j = 0$ **to** $d$ **do**
      $fn_{\{j,j\}}[m] \leftarrow fn_{\{j,j\}}[m] + q_j$
    **end for**
  **else if** $J_2 = \emptyset$ **then**
    $\bar{p} \leftarrow P'[\pi_P(J_1)]$ ▷ $\bar{p} = (p_0, p_1, \ldots, p_d)$
    **for** $i = 0$ **to** $d$ **do**
      $fn_{\{i,i\}}[m] \leftarrow fn_{\{i,i\}}[m] + p_i$
    **end for**
  **end if**
**end for**

**for** $i = 0$ **to** $d$ **do** ▷ Refresh and store domains in registers
  **for** $j = i+1$ **to** $d$ **do**
    $r''_{\{i,j\}} \xleftarrow{\$} \mathbb{F}_2$
    $v_{\{i,j\}}[m] \leftarrow \left\{ fn_{\{i,j\}}[m] + r''_{\{i,j\}} \right\}$ ▷ $\{ \cdot \}$ denotes register storage
    $v_{\{j,i\}}[m] \leftarrow \left\{ fn_{\{j,i\}}[m] + r''_{\{i,j\}} \right\}$
  **end for**
**end for**

**for** $i = 0$ **to** $d$ **do** ▷ Compression and final computation of $fn_i[m]$
  $fn_i[m] \leftarrow \left\{ fn_{\{i,i\}}[m] \right\} + \sum_{j=0, j\neq i}^{d} v_{\{i,j\}}[m]$
**end for**

formed by gathering $fn_{\{i,i\}}[m]$ and the off-diagonal domains relevant to $i$, and then we sum over $i$. Concretely,

$$fn[m] \;=\; \sum_{i=0}^{d} fn_i[m] \;=\; \sum_{i=0}^{d} \sum_{j=0}^{d} fn_{\{i,j\}}[m].$$

In this double sum, each random bit $r''_{\{i,j\}}$ is added twice and thus vanishes, while every product $p_i \cdot q_j$ or pure partial term $p_i$ or $q_j$ appears exactly once. This ensures that $fn[m]$ is reconstructed without any missing or extra terms, establishing the correctness of *CrossMult*$_{\mathrm{d}}$. For vectorial Boolean functions $fn : \mathbb{F}_2^k \to \mathbb{F}_2^\ell$, *CrossMult*$_d$ processes each output coordinate $fn[m]$ independently and in parallel. By ensuring the correctness of each

coordinate separately, the correctness of the entire vectorial function is guaranteed. □

**Theorem 4.** *The area-latency trade-off for order $d$ is $d^{th}$-order PINI and SNI.*

*Proof.* The security of the area-latency trade-off construction relies on the security of HO-TSM$_d$ (Theorem 2) and that of *CrossMult*$_d$. We can categorize all possible probe locations into three groups as follows:

1. Probing any of the HO-TSM$_d$ blocks.
2. Probing the first stage of *CrossMult*$_d$. (Lines 1 to 35 in Algorithm 1)
3. Probing the second stage of *CrossMult*$_d$. (Lines 36 to 38 in Algorithm 1)

For each of these locations, we provide the following shares to the simulator:

- For any probe in Group 1, the simulator is given the share domain following the simulation for HO-TSM$_d$.
- For any probe in Groups 2 or 3, the simulator is given nothing.

We then demonstrate the simulation of the probed values:

- For any probe in Group 1: A probe in this group is simulated following the simulation procedure for HO-TSM$_d$.
- For any probe in Group 2: Since the first stage of *CrossMult*$_d$ is higher-order non-complete and operates on one share of each input, namely $P'_i$ and $Q'_j$, the probe observes a single share domain of an output share from each HO-TSM$_d$ block. Recall that a probe on the output of an HO-TSM$_d$ block can be simulated without giving any input shares, owing to its $d$-th order PINI and SNI security. Consequently, as the output of each HO-TSM$_d$ block can be simulated without providing any information to the simulator, the simulation is perfect.
- For any probe in Group 3: The security is derived from the security of *CrossMult*$_d$. Specifically, according to the proof in Faust *et al.* [FGP+18, Proposition 3], a glitch-extended probe in the second stage of *CrossMult*$_d$ can be simulated with one share per input. Since this corresponds to probing a single output share from each HO-TSM$_d$ block, this can be simulated perfectly without giving the simulator any shares.

□

## 4.4 First-Order Secure AES S-Box

As our first case study, we apply our construction to build a first-order secure AES S-Box. The input bits $\{a, b, c, d, e, f, g, h\}$ are divided into two equal sets: $P = \{a, b, c, d\}$ and $Q = \{e, f, g, h\}$. In the initial step, we apply HO-TSM$_1$, the first-order case of TSM presented in [VDB+24], to both sets $P$ and $Q$. The HO-TSM$_1$ blocks produce all degree-1, -2, -3, and -4 product terms as outputs.

The shared outputs of the first and second HO-TSM$_1$ blocks are:

$$\begin{aligned} P'_i = \{&(a)_i, (b)_i, (c)_i, (d)_i, (ab)_i, (ac)_i, (ad)_i, (bc)_i, (bd)_i, (cd)_i, \\ &(abc)_i, (abd)_i, (acd)_i, (bcd)_i, (abcd)_i\}, \quad i \in \{0, 1\} \\ Q'_j = \{&(e)_j, (f)_j, (g)_j, (h)_j, (ef)_j, (eg)_j, (eh)_j, (fg)_j, (fh)_j, (gh)_j, \\ &(efg)_j, (efh)_j, (egh)_j, (fgh)_j, (efgh)_j\}, \quad j \in \{0, 1\} \end{aligned} \tag{19}$$

The eight coordinate functions for the output of the AES S-Box in algebraic normal form contain 254 terms, including terms up to the seventh degree. A term is classified as an inner-module term if it includes inputs exclusively from either the set $P$ or the set $Q$. For example,

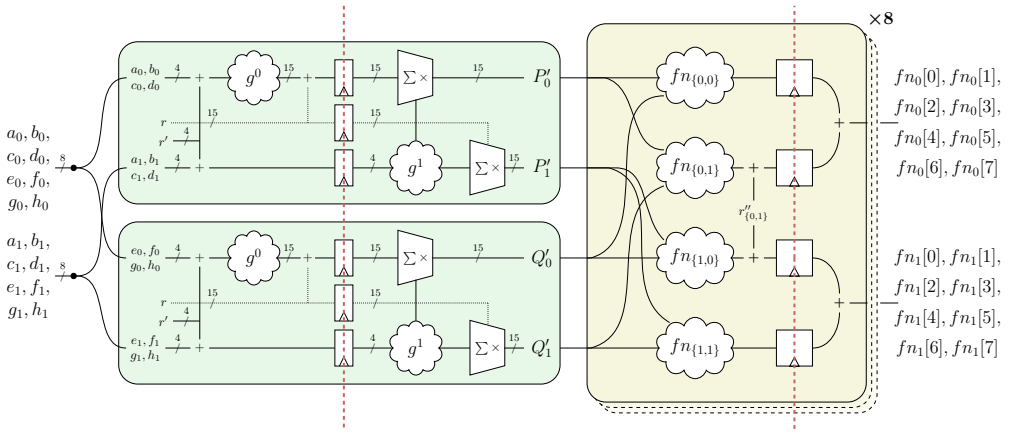



**Figure 6:** The area-latency trade-off for a first-order secure AES S-Box. Red vertical dashed lines indicate register stages.

$(abd)$ is an inner-module term. A cross-module term includes inputs from both sets $P$ and $Q$. For example, $(abcfg)$ is a cross-module term. We define the first-order $CrossMult_1$ block, shown in Figure 6, using four domains: $fn_{\{0,0\}}[l], fn_{\{0,1\}}[l], fn_{\{1,0\}}[l], fn_{\{1,1\}}[l]$ for the eight coordinate functions, where $l \in \{0, 1, \ldots, 7\}$. The domain $fn_{\{i,j\}}[l]$ takes $P'_i$ and $Q'_j$ as inputs. Each cross-module term in the coordinate function $fn[l]$, such as $abcfg$, is calculated and distributed across all four domains. Specifically, $(abc)_0 \cdot (fg)_0$, $(abc)_0 \cdot (fg)_1$, $(abc)_1 \cdot (fg)_0$, and $(abc)_1 \cdot (fg)_1$ are assigned to domains $fn_{\{0,0\}}[l], fn_{\{0,1\}}[l], fn_{\{1,0\}}[l], fn_{\{1,1\}}[l]$ respectively. Each inner-module term in the coordinate function $fn[l]$ is distributed to domains $fn_{\{0,0\}}[l]$ and $fn_{\{1,1\}}[l]$. For example, $(abd)_0$ is assigned to $fn_{\{0,0\}}[l]$ and $(abd)_1$ to $fn_{\{1,1\}}[l]$. The four domains are subsequently refreshed using one random bit per coordinate function $fn[l]$ and stored in four registers: $v_{\{0,0\}}[l], v_{\{0,1\}}[l], v_{\{1,0\}}[l], v_{\{1,1\}}[l]$. The outputs of the S-Box are calculated straightforwardly by summing the contents of the registers: $fn_0[l] = v_{\{0,0\}}[l] + v_{\{0,1\}}[l]$ and $fn_1[l] = v_{\{1,0\}}[l] + v_{\{1,1\}}[l]$.

**Table 3:** Relevant First-Order Masked AES S-Box Implementations.

| Design | Area (GE) | Rand (bits) | Latency (Cycles) | Library |
|---|---|---|---|---|
| LMDPL[SBHM20] | 3480 | 36 | 1 | GF28 |
| TSM[VDB+24] | 14352[a] / 20523[b] | 262 | 1 | NanGate45 |
| GLM[GIB18] | 60730 | 2048 | 1 | UMC90 |
| $GHPC_{LL}$[KSM22] | 64111 | 2048 | 1 | UMC180 |
| **TSM** | **4020**[a] / **4657**[b] | **46** | **2** | **NanGate45** |
| LMDPL[LMW14] | 2825 | 36 | 2 | - |
| GLM[GIB18] | 6740 | 416 | 2 | UMC90 |
| LLTI[AZN21] | 25780 | 0 | 2 | NanGate45 |
| TI[AZN21] | 58410 | 0 | 2 | NanGate45 |
| GHPC[KSM22] | 77145 | 8 | 2 | UMC180 |
| TI (B-P)[GC17] | 2913 | 20 | 3 | NanGate45 |
| TI (Canright)[GC17] | 2835 | 32 | 3 | NanGate45 |
| HPC (B-P)[CGM+23] | 2780 | 46 | 4 | NanGate45 |
| HPC (Canright)[CGM+23] | 1950 | 36 | 4 | NanGate45 |

*a* `compile_ultra -no_autoungroup -no_boundary_optimization`
*b* `compile -exact_map`

**Implementation Costs.** The area-latency trade-off for the first-order secure AES S-Box was synthesized using the same compiler options discussed in Section 3.4, targeting a clock frequency of 100 MHz. It occupies an area of 4657 GE with `compile` option and 4020 GE with `compile_ultra` option. Each HO-TSM$_1$ block occupies approximately 500 GE, aligning with the area reported for the PRINCE S-Box in [VDB+24], which also operates on four inputs. As for $CrossMult_d$, since the domains of all coordinate functions $\{fn_{\{i,j\}}[0], fn_{\{i,j\}}[1], fn_{\{i,j\}}[2], fn_{\{i,j\}}[3], fn_{\{i,j\}}[4], fn_{\{i,j\}}[5], fn_{\{i,j\}}[6], fn_{\{i,j\}}[7]\}$ operate on the same set of inputs $P'_i$ and $Q'_j$, we group their combinatorial logic to further optimize for area and avoid any logic duplication. The S-box requires 46 random bits and has a latency of two cycles. Eight random bits are used for the initial refresh ($r'$ as shown in Figure 6), fifteen random bits are used for refreshing the outputs of $g^0$ for each HO-TSM$_1$ block ($r$), and eight random bits are used for the eight $CrossMult_1$ blocks ($r''$).

Table 3 summarizes the area, latency and randomness requirements of various relevant works. Our AES S-Box demonstrates significant improvements over existing techniques. It is both smaller in area and demands considerably less randomness compared to GLM [GIB18], while maintaining the same latency of two cycles. When compared to other PINI implementations such as HPC and GHPC, our AES S-Box achieves a latency reduction by half compared to the HPC S-Box and occupies significantly less area than the GHPC S-Box. Despite these advantages, our S-Box is slightly larger than the LMDPL S-Box [SBHM20] and it requires more randomness and an additional clock cycle. However, it is important to note that LMDPL employs dual-rail logic, which inherently requires a pre-charge phase between consecutive evaluations.

## 4.5 Second-Order Secure AES S-Box

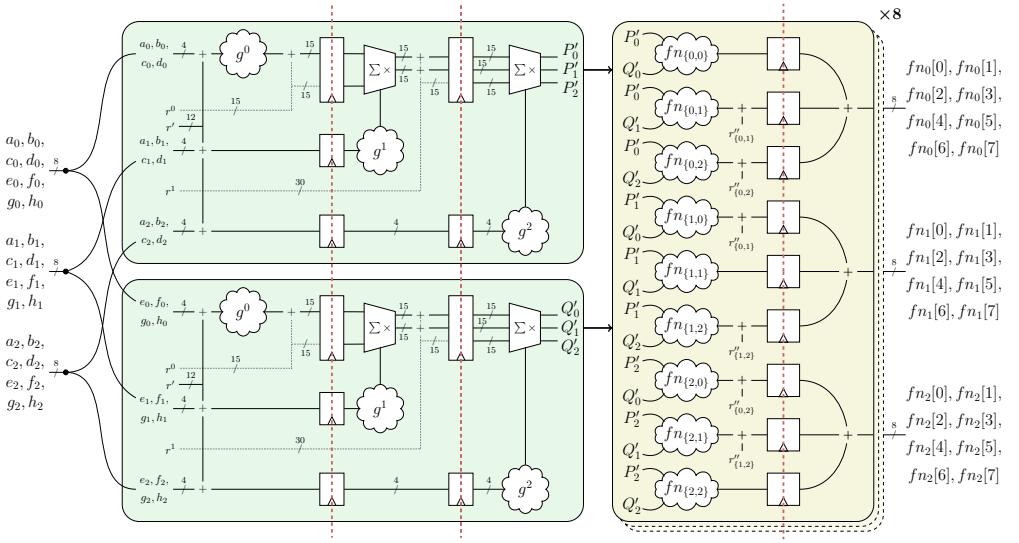



**Figure 7:** The area-latency trade-off for the second-order secure AES S-Box. Red vertical dashed lines indicate register stages.

In our second case study, we apply the area-latency trade-off for the second-order secure AES S-Box. The structure of the S-Box, as depicted in Figure 7, closely resembles that of the first-order S-Box described in Section 4.4. To avoid redundancy, we will focus more on the differences between the two. We implement the two HO-TSM$_2$ blocks using the second-order case of the HO-TSM methodology introduced in Section 3. These blocks accept inputs $P = \{a, b, c, d\}$ and $Q = \{e, f, g, h\}$ and generate all degree-1, -2, -3, and -4

product terms as outputs $P'_i$ and $Q'_j$ (where $i, j \in \{0, 1, 2\}$). These outputs are similar to Eq.(19), with each term represented using three shares instead of two. We define the second-order $CrossMult_2$ block, illustrated in Figure 7, using nine domains: $fn_{\{i,j\}}[l]$, where $i, j \in \{0, 1, 2\}$ for the eight coordinate functions. The cross-module terms are spread across all nine domains, while the inner-module terms are assigned to $fn_{\{0,0\}}[l]$, $fn_{\{1,1\}}[l]$, and $fn_{\{2,2\}}[l]$. Each of the nine domains is then refreshed using three random bits per coordinate function $fn[l]$ and stored in nine registers.

**Table 4:** Relevant Second-Order Masked AES S-Box Implementations.

| Design | Area (GE) | Rand (bits) | Latency (Cycles) | Library |
|---|---|---|---|---|
| **HO-TSM (Section 3.4)** | **33700**[a] / **48600**[b] | **786** | **2** | **NanGate45** |
| GLM[GIB18] | 57110 | 4446 | 2 | UMC90 |
| **HO-TSM (Section 4.5)** | **8401**[a] / **9469**[b] | **138** | **3** | **NanGate45** |
| HPC (B-P)[CGM+23] | 6790 | 138 | 4 | NanGate45 |
| HPC (Canright)[CGM+23] | 4560 | 96 | 4 | NanGate45 |
| TI[CRB+16] | 3796 | 162 | 5 | NanGate45 |
| DOM[GKM+24] | 4300 | 78 | 5 | UMC65 |
| TI[CBR+15] | 7849 | 126 | 6 | NanGate45 |
| DOM[GMK16] | 5300 | 54 | 8 | UMC180 |

*a* `compile_ultra -no_autoungroup -no_boundary_optimization`
*b* `compile -exact_map`

**Implementation Costs.** The area-latency trade-off of the second-order secure S-Box occupies an area of 9469 GE with the `compile` option and 8401 GE with the `compile_ultra` option. It requires 138 random bits in total. Twenty-four random bits are used for the initial refresh ($r'$), as shown in Figure 7. Fifteen random bits are used for refreshing the outputs of $g^0$ in the first stage, and thirty random bits are used for refreshing the outputs of the second stage ($r^1$) for each HO-TSM$_2$ block. Additionally, twenty-four random bits are used for the eight $CrossMult_2$ blocks ($r''$). The latency of this S-Box is three cycles, two register stages inside HO-TSM$_2$ and one inside $CrossMult_2$. In Table 4, we enumerate all relevant second-order masked AES S-Boxes. Our proposed S-Box demonstrates good performance compared to other designs, particularly when evaluated based on the product of area and latency. Additionally, our S-Box maintains a randomness requirement that is comparable to other designs. It is noteworthy that our construction improves both area efficiency and randomness requirements when compared to our two-cycle second-order secure S-Box described in Section 3.4, with the trade-off of one additional clock cycle.

## 4.6 Round-Based First-Order Secure AES-128

In order to evaluate the performance of the area-latency trade-off of the first-order secure AES S-Box in comparison to the state-of-the-art, we implemented a low-latency, round-based AES-128 design with parallel instances of our AES S-Box. Our architecture includes both a masked round function and a masked key schedule. We instantiate sixteen parallel instances of our S-Box for SubBytes in the round function and four instances for SubWord in the key expansion. Since our S-Box is glitch-extended PINI and SNI-secure, we trivially compose our S-Boxes with other linear elements such as MixColumns and ShiftRows and avoid introducing any register layers outside the S-Box to store the intermediate state, which would increase latency.

**Table 5:** Utilization results of relevant first-order secure round-based AES-128 implementations.

| Design | Method | Area (kGE) | Rand (bits/round) | Latency (Cycles) | Library |
|---|---|---|---|---|---|
| **This Work** | **TSM** | **91.8**[a] / **106.1**[b] | **920** | **20** | **NanGate45** |
| [SBHM20] | LMDPL | 157.5 | 720 | 10 | GF28 |
| [ADN+22] | TI | 115.7 | 0 | 42 | NanGate45 |
| [KM22] | HPC | 44.5 | 1360 | 55 | NanGate45 |

a `compile_ultra -no_autoungroup -no_boundary_optimization`
b `compile -exact_map`

Table 5 presents some relevant first-order, round-based AES-128 designs with low latency. Our area results were obtained with the target frequency set to 100 MHz. Notably, our design occupies a smaller area compared to both the TI [ADN+22] and LMDPL [SBHM20] designs. The TI S-Box (5.84kGE) is slightly larger than ours, which explains the difference in area between the designs. The LMDPL S-Box operates on dual-rail logic, requiring pre-charging before each evaluation. To avoid delays caused by pre-charging, the authors use a duality approach by duplicating the round function and state/key registers, creating two instances of the evaluation circuit. This method allows alternating evaluations: while one instance pre-charges, the other evaluates. Consequently, they achieve a latency of 10 cycles for the entire AES operation. Despite the smaller size of their LMDPL S-Box, duplicating the circuitry leads to a larger overall area compared to our design. Our design requires more randomness per round than the LMDPL design, whereas the TI design requires no online randomness at all. With a latency of twenty cycles, our design falls between the LMDPL and TI designs. The HPC design is the most compact among all designs, although it has higher latency compared to the others.

# 5 Security Evaluation

In this section, we conduct an experimental evaluation of the security of our designs, using both formal and practical verification methods. Specifically, in Section 5.1, we demonstrate the security of our AES S-Boxes using the formal verification tools `maskVerif` [BBC+19] and PROLEAD [MM22]. In Section 5.2, we practically verify and present side-channel analysis results of our designs by implementing them on FPGA.

## 5.1 Formal Verification

Since SILVER is limited to verifying only small circuits and cannot handle larger designs, we use two other formal verification tools, `maskVerif` and PROLEAD, to verify the two-cycle, first-order AES S-Box (Section 4.4) and the three-cycle, second-order AES S-Box (Section 4.5).

`maskVerif` [BBC+19] is a language-based formal verification tool capable of verifying masked circuits under the $t$-probing, $t$-NI, and $t$-SNI security notions for hardware designs, with scalability for arbitrary orders. To verify our S-Boxes, we represented them in Static Single Assignment (SSA) form. For the SNI tests, defining more than one output at a time caused the tool to hang, so we designated each of the eight coordinate functions as the output, one at a time, and conducted eight separate verifications. The two S-Boxes successfully passed all (1-probing, 1-NI, 1-SNI) and (2-probing, 2-NI, 2-SNI) tests, respectively.

PROLEAD [MM22] is a simulation-based leakage detection tool that evaluates the statistical independence of intermediate probes in a masked circuit under the robust probing

model. We generated netlists for the two S-Boxes using the `compile_ultra` command and performed fixed versus random G-tests in `compact` mode with default parameters. For the first-order S-Box, no leakage was detected across 200 million simulations, as the $-\log_{10}(p)$ value did not exceed the default threshold of 5.0 for any probing set, confirming its first-order probing security. For the second-order S-Box, we encounter a substantial number of probing sets ($5.98 \times 10^5$), resulting in a higher probability of false positives, as noted by the authors of both [MM22] and [ZZL+24]. We again conducted 200 million simulations in the multivariate setting(which took eight days), and the $-\log_{10}(p)$ values for almost all probing sets did not exceed the default threshold of 5.0, except for very few probing sets that have $5.0 < -\log_{10}(p) < 6.0$. However, we do not observe any continuous increase in $-\log_{10}(p)$ with an increasing number of simulations, which is ultimately the deciding factor for labeling a design as insecure [Mül]. Hence, we affirm the second-order probing security of our S-Box.

## 5.2 Practical Evaluation

For our larger designs, we revert to practical side-channel analysis by implementing three distinct protected designs on FPGA. These are:

- **Design A**: Two-cycle second-order secure AES S-Box (Section 3.4)
- **Design B**: Three-cycle second-order secure AES S-Box (Section 4.5)
- **Design C**: First-order secure round-based AES-128 (Section 4.6)

We now describe the measurement platforms, the measurement procedure, and finally present the leakage assessment results for each design.

**Experimental Setup.** We use two different FPGA boards, each with distinct advantages, in conjunction with Xilinx ISE v14.7.

1. SAKURA-X [SAK]: This board features a Xilinx Kintex-7 FPGA, which provides sufficient resources for implementing larger designs. Consequently, we deploy our larger designs (A and C) on this board. To mitigate potential IO leakage, these designs receive pre-shared inputs from a secondary SPARTAN-6 control FPGA.
2. SAKURA-G [GIS14]: Equipped with two Xilinx SPARTAN-6 FPGAs, this board generally achieves a better signal-to-noise ratio (SNR) for smaller designs, making it particularly suitable for Design B. Similar to SAKURA-X, one SPARTAN-6 FPGA handles the pre-shared inputs and outputs, while the other executes the protected design.

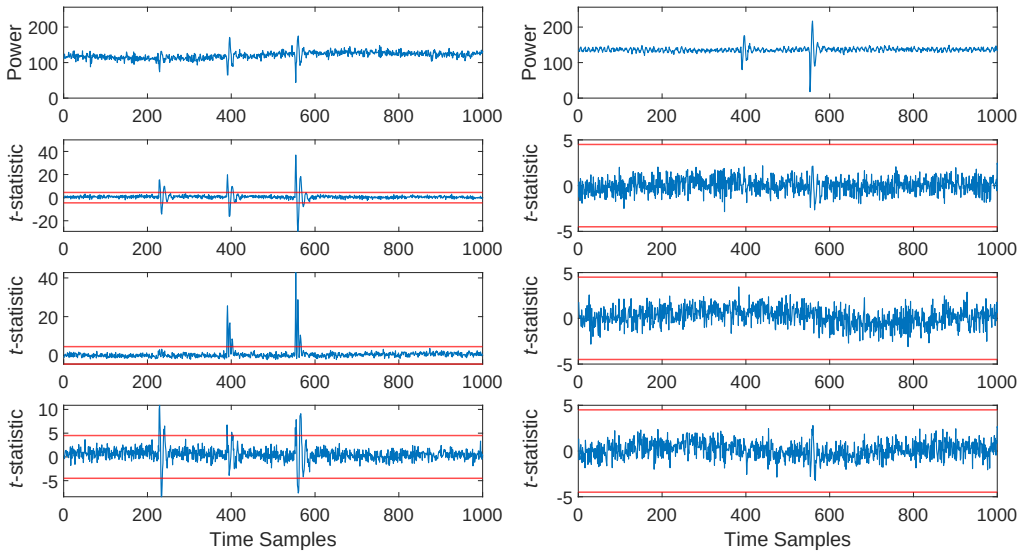
For power measurement, we capture the on-board amplified power via the J3 measurement point on SAKURA-G and via J19 with an additional 30 dB pre-amplifier on SAKURA-X. To prevent overlap in power traces between clock cycles, both boards operate at a conservative clock frequency of 3 MHz. This frequency is externally generated and synchronized with our Tektronix DPO7254C oscilloscope, enabling precise synchronous sampling at 500 MS/s.

**Randomness Generation.** For the two second-order secure AES S-Boxes (Designs A and B), we employ an unprotected AES-128 core in Output Feedback (OFB) mode on the control FPGA (SPARTAN-6) to generate the needed randomness. Each time the S-Box operates, the target FPGA fetches sufficient random bits from the control side.

For the first-order secure round-based AES-128 (Design C), we instantiate twelve unrolled Trivium [CMM+23, Can06] cores on the Kintex-7 FPGA, each generating 64 bits of fresh randomness in parallel with the main AES core. This setup ensures that fresh random bits are available every clock cycle.

**TVLA.** We apply the non-specific Test Vector Leakage Assessment (TVLA) approach as outlined by Bilgin *et al.* [BGN+14] and originally introduced in [GJJR11, CDMG+13]. For efficiency, we adopt the fast computation method proposed by Reparaz *et al.* [RGV17]. Fixed versus random tests are conducted by supplying either a fixed or random input in a randomly interleaved sequence. An absolute $t$-value threshold of $\pm 4.5$ is used to identify potential leakage.

**Validation with Disabled PRNGs.** As the first step, we validate our setup by reporting leakage results with the PRNGs disabled. In this configuration, our designs operate without online randomness, and the randomness for the initial input sharing is set to zero. Figures 8a, 9a, and 10a present (from top to bottom) the power trace, and the first-, second-, and third-order univariate $t$-test statistics for Designs A, B, and C, respectively, based on 100 k traces. Red lines indicate the $\pm 4.5$ threshold commonly used to identify leakage. The top row in each figure shows the sample power trace (raw oscilloscope ADC output) for the three designs. For Designs A and B, the power measurement in the top row corresponds to one AES S-Box computation. For Design C, the measurement covers the complete ten rounds of the protected round-based AES-128 core. As expected, we observe clear peaks in the first-order $t$-test, confirming the validity of our setup and measurement methodology.

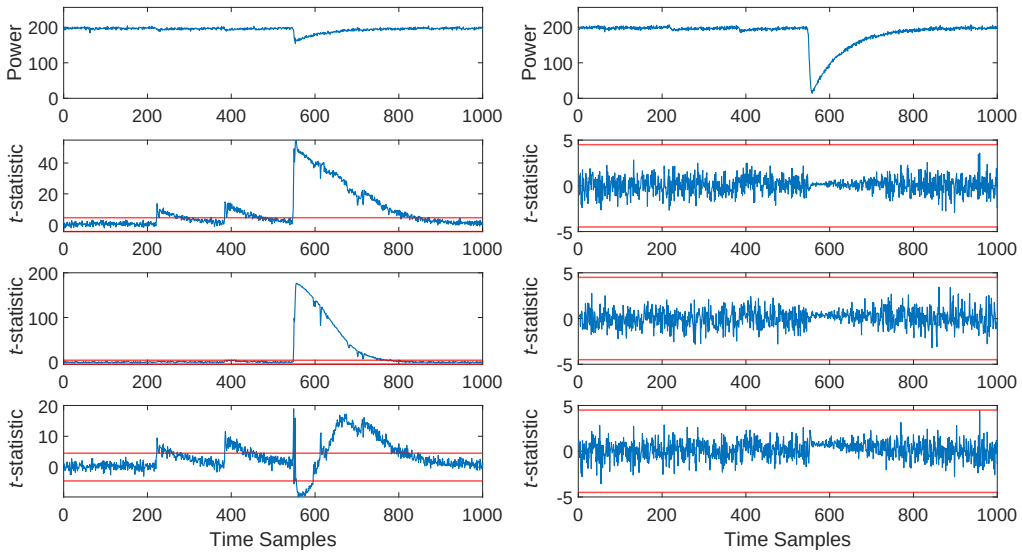



**(a)** PRNGs set to *OFF* with 100k traces. **(b)** PRNGs set to *ON* with 100 million traces.

**Figure 8:** Univariate TVLA results for Design A: 2-cycle $2^{nd}$-order AES S-Box(Section 3.4) on SAKURA-X.(From top to bottom: power trace, 1st-, 2nd-, and 3rd-order $t$-statistics)

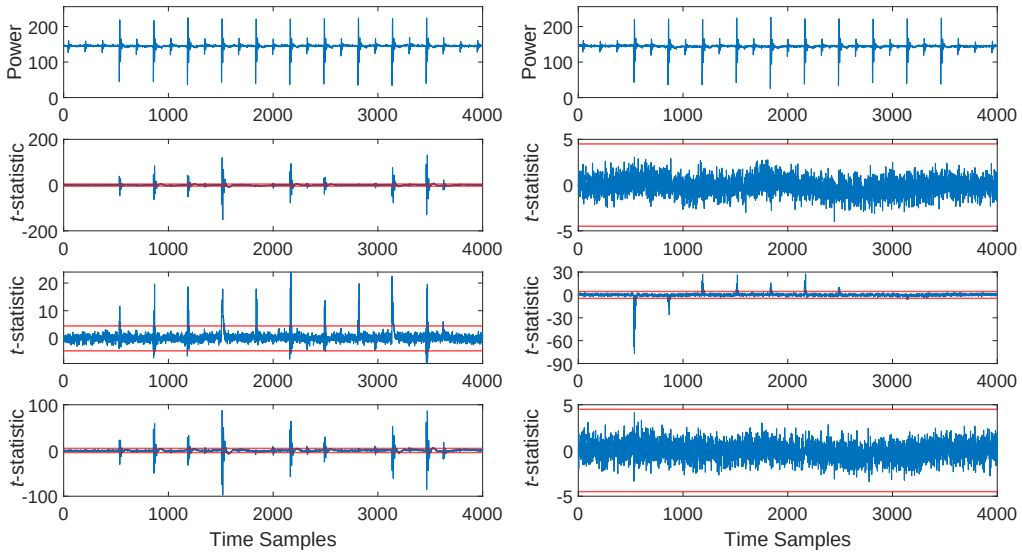
**Results with Enabled PRNGs.** Figures 8b, 9b, and 10b show the leakage assessment results with PRNGs enabled.

- Designs A and B: We analyze 100 million traces. We observe no first- or second-order leakage, as the $t$-test values do not exceed the threshold of $\pm 4.5$ at any time sample.
- Design C: We analyze 250 million traces. It is no surprise that we observe no first-order leakage and significant peaks for second-order leakage, both results are as expected. We expect all three designs to exhibit third-order leakage, but with the




**(a)** PRNGs set to *OFF* with 100k traces. **(b)** PRNGs set to *ON* with 100 million traces.

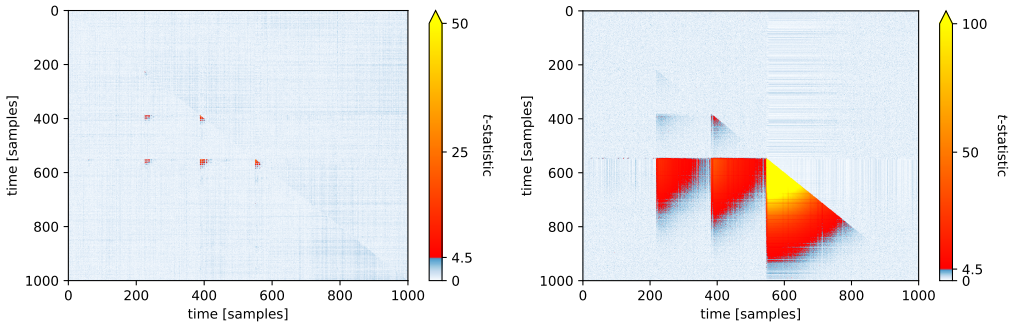
**Figure 9:** Univariate TVLA results for Design B: 3-cycle $2^{nd}$-order AES S-Box (Section 4.5) on SAKURA-G.(From top to bottom: power trace, 1st-, 2nd-, and 3rd-order $t$-statistics)




**(a)** PRNGs set to *OFF* with 100k traces. **(b)** PRNGs set to *ON* with 250 million traces.

**Figure 10:** Univariate TVLA results for Design C: 10 rounds of first-order secure round-based AES-128 (Section 4.6).(From top to bottom: power trace, 1st-, 2nd-, and 3rd-order $t$-statistics)

given number of traces they do not. One may speculate that the number of traces is not sufficient.




**(a)** Design A: Two-cycle $2^{nd}$-order AES S-Box(Section 3.4) on SAKURA-X.

**(b)** Design B: Three-cycle $2^{nd}$-order AES S-Box(Section 4.5) on SAKURA-G.

**Figure 11:** Bivariate analysis of our two $2^{nd}$-order AES S-Box designs. Bottom left triangle: PRNGs set to *OFF* with 100k traces. Top right triangle: PRNGs set to *ON* with 100 million traces.

**Bivariate Analysis.** We also conduct bivariate analysis using the tool by Cassiers *et al.* [CB23] for our two second-order secure AES S-Boxes (Designs A and B) using the same number of sample points as in the univariate tests, without downsampling. We verify the functionality of our setup by initially conducting tests with PRNGs disabled. The lower left triangles of Figures 11a and 11b display the second-order $t$-test results based on 100k traces, revealing expected leakage. Interestingly, the time samples for the leaks (around 200, 400, 600) align closely with the univariate $t$-test results shown in Figures 8a and 9a, confirming the robustness of our setup. Finally, for both Designs A and B, we observe no leakage in the top right triangles when the PRNGs are enabled.

# 6 Conclusion

We introduce HO-TSM as a higher-order extension of TSM. The fundamental concept behind HO-TSM remains akin to TSM, ensuring security by separating share processing through the use of register layers. Our method is versatile, as it applies to any arbitrary (vectorial) Boolean function. We also establish the composable security of HO-TSM under the PINI and SNI security notions, which simplifies circuit construction by alleviating concerns about composition.

Furthermore, we introduce a trade-off between area and latency for HO-TSM that significantly improves both area and randomness efficiency. These enhancements are demonstrated by our low-latency first-order AES S-Box and the full AES-128 implementation. Our three-cycle second-order AES S-Box balances low-latency with moderate area and randomness requirements, offering a middle ground between designs that prioritize either latency reduction or minimizing area and randomness.

While our focus was on the AES S-Box as a case study, an intriguing direction for future research would be to implement various other S-Boxes and assess their performance relative to existing techniques.

# Acknowledgment

This work was supported in part by the Flemish Government through the Cybersecurity Research Program with grant number VOEWICS02, by the European Commission through the Horizon 2020 research and innovation program under grant agreement Belfort ERC Ad-

vanced Grant 101020005 695305, and through the Horizon Europe research and innovation program under grant agreement HORIZON-CL3-2021-CS-01-02 101070008 ORSHIN.

## References

[ADN+22] Amund Askeland, Siemen Dhooghe, Svetla Nikova, Vincent Rijmen, and Zhenda Zhang. Guarding the first order: The rise of AES maskings. In Ileana Buhan and Tobias Schneider, editors, *Smart Card Research and Advanced Applications - 21st International Conference, CARDIS 2022, Birmingham, UK, November 7-9, 2022, Revised Selected Papers*, volume 13820 of *Lecture Notes in Computer Science*, pages 103–122. Springer, 2022.

[AZN21] Victor Arribas, Zhenda Zhang, and Svetla Nikova. LLTI: low-latency threshold implementations. *IEEE Trans. Inf. Forensics Secur.*, 16:5108–5123, 2021.

[BBC+19] Gilles Barthe, Sonia Belaïd, Gaëtan Cassiers, Pierre-Alain Fouque, Benjamin Grégoire, and François-Xavier Standaert. maskverif: Automated verification of higher-order masking in presence of physical defaults. In Kazue Sako, Steve A. Schneider, and Peter Y. A. Ryan, editors, *Computer Security - ESORICS 2019 - 24th European Symposium on Research in Computer Security, Luxembourg, September 23-27, 2019, Proceedings, Part I*, volume 11735 of *Lecture Notes in Computer Science*, pages 300–318. Springer, 2019.

[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 *CCS*, pages 116–129. ACM, 2016.

[BGN+14] Begül Bilgin, Benedikt Gierlichs, Svetla Nikova, Ventzislav Nikov, and Vincent Rijmen. Higher-order threshold implementations. In Palash Sarkar and Tetsu Iwata, editors, *Advances in Cryptology - ASIACRYPT 2014 - 20th International Conference on the Theory and Application of Cryptology and Information Security, Kaoshiung, Taiwan, R.O.C., December 7-11, 2014, Proceedings, Part II*, volume 8874 of *Lecture Notes in Computer Science*, pages 326–343. Springer, 2014.

[Can06] Christophe De Cannière. Trivium: A stream cipher construction inspired by block cipher design principles. In Sokratis K. Katsikas, Javier López, Michael Backes, Stefanos Gritzalis, and Bart Preneel, editors, *Information Security, 9th International Conference, ISC 2006, Samos Island, Greece, August 30 - September 2, 2006, Proceedings*, volume 4176 of *Lecture Notes in Computer Science*, pages 171–186. Springer, 2006.

[CB23] Gaëtan Cassiers and Olivier Bronchain. Scalib: A side-channel analysis library. *Journal of Open Source Software*, 8(86):5196, 2023.

[CBR+15] Thomas De Cnudde, Begül Bilgin, Oscar Reparaz, Ventzislav Nikov, and Svetla Nikova. Higher-order threshold implementation of the AES s-box. In Naofumi Homma and Marcel Medwed, editors, *Smart Card Research and Advanced Applications - 14th International Conference, CARDIS 2015, Bochum, Germany, November 4-6, 2015. Revised Selected Papers*, volume 9514 of *Lecture Notes in Computer Science*, pages 259–272. Springer, 2015.

[CDMG+13] J. Cooper, E. De Mulder, G. Goodwill, J. Jaffe, G. Kenworthy, and P. Rohatgi. Test vector leakage assessment (TVLA) methodology

in practice. http://icmc-2013.org/wp/wp-content/uploads/2013/09/goodwillkenworthtestvector.pdf, 2013.

[CGM+23] Gaëtan Cassiers, Barbara Gigerl, Stefan Mangard, Charles Momin, and Rishub Nagpal. Compress: Reducing area and latency of masked pipelined circuits. *IACR Cryptol. ePrint Arch.*, page 1600, 2023.

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

[CRB+16] Thomas De Cnudde, Oscar Reparaz, Begül Bilgin, Svetla Nikova, Ventzislav Nikov, and Vincent Rijmen. Masking AES with d+1 shares in hardware. In Benedikt Gierlichs and Axel Y. Poschmann, editors, *Cryptographic Hardware and Embedded Systems - CHES 2016 - 18th International Conference, Santa Barbara, CA, USA, August 17-19, 2016, Proceedings*, volume 9813 of *Lecture Notes in Computer Science*, pages 194–212. Springer, 2016.

[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.

[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.

[GC17] Ashrujit Ghoshal and Thomas De Cnudde. Several masked implementations of the boyar-peralta AES s-box. In Arpita Patra and Nigel P. Smart, editors, *Progress in Cryptology - INDOCRYPT 2017 - 18th International Conference on Cryptology in India, Chennai, India, December 10-13, 2017, Proceedings*, volume 10698 of *Lecture Notes in Computer Science*, pages 384–402. Springer, 2017.

[GIB18] Hannes Groß, Rinat Iusupov, and Roderick Bloem. Generic low-latency masking in hardware. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2018(2):1–21, 2018.

[GIS14] Hendra Guntur, Jun Ishii, and Akashi Satoh. Side-channel attack user reference architecture board sakura-g. In *2014 IEEE 3rd global conference on consumer electronics (GCCE)*, pages 271–274. IEEE, 2014.

[GJJR11] G. Goodwill, B. Jun, J. Jaffe, and P. Rohatgi. A testing methodology for side channel resistance validation. http://csrc.nist.gov/newsevents/non-invasive-attack-testing-workshop/08Goodwill.pdf, 2011.

[GKM+24] Barbara Gigerl, Franz Klug, Stefan Mangard, Florian Mendel, and Robert Primas. Smooth passage with the guards: Second-order hardware masking of the AES with low randomness and low latency. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2024(1):309–335, 2024.

[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.

[GMO01] Karine Gandolfi, Christophe Mourtel, and Francis Olivier. Electromagnetic analysis: Concrete results. In Çetin Kaya Koç, David Naccache, and Christof Paar, editors, *Cryptographic Hardware and Embedded Systems - CHES 2001, Third International Workshop, Paris, France, May 14-16, 2001, Proceedings*, volume 2162 of *Lecture Notes in Computer Science*, pages 251–261. Springer, 2001.

[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.

[ISW03] Yuval Ishai, Amit Sahai, and David A. Wagner. Private circuits: Securing hardware against probing attacks. In Dan Boneh, editor, *Advances in Cryptology - CRYPTO 2003, 23rd Annual International Cryptology Conference, Santa Barbara, California, USA, August 17-21, 2003, Proceedings*, volume 2729 of *Lecture Notes in Computer Science*, pages 463–481. Springer, 2003.

[KJJ99] Paul C. Kocher, Joshua Jaffe, and Benjamin Jun. Differential power analysis. In 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.

[LMW14] Andrew J. Leiserson, Mark E. Marson, and Megan A. Wachs. Gate-level masking under a path-based leakage metric. In Lejla Batina and Matthew Robshaw, editors, *Cryptographic Hardware and Embedded Systems - CHES 2014 - 16th International Workshop, Busan, South Korea, September 23-26, 2014. Proceedings*, volume 8731 of *Lecture Notes in Computer Science*, pages 580–597. Springer, 2014.

[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.

[MPG05] Stefan Mangard, Thomas Popp, and Berndt M. Gammel. Side-channel leakage of masked CMOS gates. In Alfred Menezes, editor, *Topics in Cryptology - CT-RSA 2005, The Cryptographers' Track at the RSA Conference 2005, San Francisco, CA, USA, February 14-18, 2005, Proceedings*, volume 3376 of *Lecture Notes in Computer Science*, pages 351–365. Springer, 2005.

[Mül] Nicolai Müller. Prolead results. https://github.com/ChairImpSec/PROLEAD/wiki/Results. Accessed: October 12, 2024.

[NAN] NANGATE. The nangate 45nm open cell library. https://www.nangate.com.

[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.

[QS01] Jean-Jacques Quisquater and David Samyde. Electromagnetic analysis (EMA): measures and counter-measures for smart cards. In Isabelle Attali and Thomas P. Jensen, editors, *Smart Card Programming and Security, International Conference on Research in Smart Cards, E-smart 2001, Cannes, France, September 19-21, 2001, Proceedings*, volume 2140 of *Lecture Notes in Computer Science*, pages 200–210. Springer, 2001.

[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*, volume 9215 of *Lecture Notes in Computer Science*, pages 764–783. Springer, 2015.

[RGV17] Oscar Reparaz, Benedikt Gierlichs, and Ingrid Verbauwhede. Fast leakage assessment. In Wieland Fischer and Naofumi Homma, editors, *Cryptographic Hardware and Embedded Systems - CHES 2017 - 19th International Conference, Taipei, Taiwan, September 25-28, 2017, Proceedings*, volume 10529 of *Lecture Notes in Computer Science*, pages 387–399. Springer, 2017.

[SAK] SAKURA. Side-channel Attack User Reference Architecture. https://satoh.cs.uec.ac.jp/SAKURA/hardware/SAKURA-X.html.

[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.

[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 Transactions on Cryptographic Hardware and Embedded Systems*, 2024(3):249–272, Jul. 2024.

[ZZL+24] Cankun Zhao, Hang Zhao, Jiangxue Liu, Bohan Yang, Wenping Zhu, Shuying Yin, Min Zhu, Shaojun Wei, and Leibo Liu. Breaking ground: A new area record for low-latency first-order masked sha-3: Advancing from the 4x area era to the 3x area era, Sep. 2024. Artifact available at https://artifacts.iacr.org/tches/2024/a25.