Compact and Low Latency First-Order AES Implementations with Low Randomness

Feng Zhou[1,2,3], Hua Chen[†2], Limin Fan[2], Junhuai Yang[2]

[1] University of Chinese Academy of Sciences, Beijing, China
[2] TCA Laboratory, Institute of Software, Chinese Academy of Sciences, Beijing, China
{zhoufeng2021,chenhua,fanlimin,yangjunhuai2023}@iscas.ac.cn
[3] Zhongguancun Laboratory, Beijing, China

Abstract. Recent years have witnessed significant progress in first-order hardware masking of AES. However, most of the work focus on the optimizations over solely one of the metrics: chip area, latency or randomness. The optimizations for one metric often leads to increasing overheads of the other metrics. Consequently, few work focus on optimizations over all three metrics of first-order AES at the same time. To bridge this gap, we introduce two compact round-based first-order AES-128 encryption implementations with the latency of 31 cycles and 40 cycles, respectively. They are provably secure in the glitch-extended probing model with relatively low consumption of randomness. To achieve this, we first introduce a method to design first-order low-latency $d+1$ TI (Threshold Implementations) for multi-output Boolean functions with a latency of only one clock cycle. Moreover, the random bits used in the low-latency TI cancels out in the expressions of output shares, which enables the applications of a COTG-based concept to significantly reduce the randomness consumption. Finally, we apply our method to design first-order implementations for AES-128 with two shares, which allows the designs to be compact. As a result, our implementations achieve a excellent trade-off over latency, area, and randomness. Compared to the 10-cycle and 20-cycle AES-128 implementations provided respectively in TCHES 2020 and TCHES 2025, the area and randomness demands of our implementations are significantly less. We also use formal verification tools, PROLEAD, and TLVA to validate the security of our designs for S-Box and round-based AES-128 implementations, respectively.

Keywords: Masking · AES · Hardware · Glitch-extended Probing Security · Low Randomness · Low Latency

1 Introduction

Cryptographic devices play a critical role in securing modern digital systems. However, they are often vulnerable to side-channel attacks, which exploit physical leakages such as power consumption [KJJ99] or electromagnetic emissions [GMO01]. To ensure robust security, it is essential to implement effective countermeasures against such attacks.

Masking [CJRR99] has emerged as one of the most prominent countermeasures against side-channel attacks, offering provable security under well-defined leakage models [FGM+18]. Nevertheless, its implementation in hardware is particularly challenging due to physical effects such as glitches [MPO05], which can undermine the assumptions of the masking model and lead to unintended information leakage. As a result, the secure design of hardware masking schemes demands special care and rigorous validation.

[†]Hua Chen is the corresponding author.

Threshold Implementations (TI) [NRR06] were the first masking schemes that offer provable security against glitches in hardware. Later advances such as Consolidated Masking Schemes (CMS) [RBN+15], Domain-Oriented Masking (DOM) [GMK16], and Generic Low-latency Masking (GLM) [GIB18] explored different trade-offs among area, randomness, and latency. For first-order, randomness-free first-order masking schemes (RFFOMS) [SM21, WFW+25] have demonstrated the feasibility of masking Boolean functions without fresh randomness.

In recent years, Probe Isolating Non-Interference (PINI) [CS20] have been proposed to provide a unified compositional framework for secure masking. Its applications in hardware masking, Hardware Private Circuits (HPCs) [CGLS20], provide generic solutions for arbitrary Boolean functions with arbitrary protection order. Earlier variants HPC1 and HPC2 have larger latency with less randomness requirements while low-latency variants such as HPC3 [KM22] and its successors [CGM+24, HB25] significantly reduce latency by increasing randomness consumption. Most recently, Time Sharing Masking (TSM) [VDB+24, VDB+25] while fulfills the PINI property is proposed to provide small and low-latency masking schemes in hardware.

Other lines of work pursue orthogonal directions. LUT-based masking schemes such as LMDPL [LMW14, SBHM20] and Self-Synchronized Masking (SESYS) [NGPM22] leverage glitch-free logic to achieve single-cycle latency.

The existing first-order low-latency AES implementations [SBHM20, VDB+25] achieve encryption latencies as low as 10 or 20 cycles. However, these designs incur significantly higher overheads in terms of randomness and area compared to alternatives with slightly higher latency.

Our Contributions. To provide a more balanced trade-off among latency, area, and randomness consumption, we propose two compact, round-based first-order implementations of AES-128 encryption that achieve both low latency and reduced randomness requirements.

We first introduce a method that leverages both uniform output sharing and randomness to construct first-order Threshold Implementations for multi-output Boolean functions using a minimal number of shares. In this method, the random bits are carefully used so that they cancel out in the output share expressions, preserving security while enabling easier application of COTG-like (COTG represents the technique *chaning of the guards* [Dae17]) techniques to reduce the overall randomness. We then apply this technique to the $\mathbb{F}_2^8$ inverter, yielding a design with a latency of only three cycles and a fresh randomness requirement of just 4 bits per S-box.

Finally, we present two first-order AES implementations with latencies of 31 and 40 cycles, respectively, based on the architecture proposed in [CGM+24]. Compared to the 40-cycle design, the 31-cycle variant defers the compression layer of the last S-box stage to the next round. This optimization reduces the latency by 22.5% (saving one cycle per round) while incurring only a modest 8.3% increase in area.

In comparison to the 10-cycle AES based on LMDPL and the 20-cycle AES based on TSM, our designs exhibit significantly lower overheads in area and randomness, while maintaining only slightly higher latency.

2 Preliminaries

This section presents the notations and essential background on statistical independence, followed by first-order masking schemes with and without fresh randomness.

2.1 Notations

Subscripts are primarily used to denote an element within a structured entity, such as the share index (or share domain) of a shared variable or the position of an element in a vector. Superscripts are reserved for distinguishing between entities, with their formatting indicating specific uses. Parentheses enclose superscripts that identify distinct shared variables, different vectors, or sets. In contrast, superscripts without parentheses represent hierarchical indices, such as bits within bytes, bytes within words, or words within larger blocks.

We use $\oplus$ or $+$ to denote field addition.

2.2 Sigma-algebra and Statistical Independence

Let $(\Omega, \mathscr{F}, \Pr)$ be a probability space. A (Boolean) random variable X is a measurable map from $(\Omega, \mathscr{F})$ to $(\{0,1\}, 2^{\{0,1\}})$. For a finite family of Boolean random variables $\mathcal{U} = \{u_0, \ldots, u_{n-1}\}$, write

$$\sigma(\mathcal{U}) := \sigma(u_0, \ldots, u_{n-1})$$

for the *sigma-algebra generated by* $\mathcal{U}$ (the smallest sigma-algebra that makes all u_i measurable). Two sigma-algebras $\mathscr{A}, \mathscr{B} \subseteq \mathscr{F}$ are *independent* if for all $A \in \mathscr{A}$ and $B \in \mathscr{B}$, $\Pr[A \cap B] = \Pr[A] \Pr[B]$. If a random variable X is independent of a sigma-algebra $\mathscr{G}$, written $X \perp\!\!\!\perp \mathscr{G}$, this is equivalent (in the Boolean case) to $\Pr[X = b \mid \mathscr{G}] = \Pr[X = b]$ for $b \in \{0,1\}$.

Definition 1 (Definition A: factorization of the joint pmf)**.** Let $\mathcal{U} = \{u_0, \ldots, u_{n-1}\}$ and $\mathcal{V} = \{v_0, \ldots, v_{m-1}\}$ be two finite sets of Boolean random variables. We say that $\mathcal{U}$ and $\mathcal{V}$ are *statistically independent*, denoted $\mathcal{U} \perp\!\!\!\perp \mathcal{V}$, if for all $\mathbf{a} \in \{0,1\}^n$ and $\mathbf{b} \in \{0,1\}^m$,

$$\Pr\left[(\mathcal{U} = \mathbf{a}) \wedge (\mathcal{V} = \mathbf{b})\right] \;=\; \Pr[\mathcal{U} = \mathbf{a}] \cdot \Pr[\mathcal{V} = \mathbf{b}].$$

Definition 2 (Definition B: independence of generated sigma-algebras)**.** With the same notation, we say that $\mathcal{U}$ and $\mathcal{V}$ are *statistically independent* if the sigma-algebras $\sigma(\mathcal{U})$ and $\sigma(\mathcal{V})$ are independent, i.e., for all $A \in \sigma(\mathcal{U})$ and $B \in \sigma(\mathcal{V})$,

$$\Pr[A \cap B] = \Pr[A] \Pr[B].$$

Remark 1 (Equivalence). For finite families of Boolean random variables, Definitions 1 and 2 are equivalent. This reconciles the standard measure-theoretic notion of independence with factorization of the joint probability mass function.

2.3 Randomness-Dependent First-Order Masking with $d + 1$ shares

To provide d-th order security in $d+1$ hardware sharing, each sensitive input variable x is split to $d + 1$ shares, $x_0, \cdots, x_d$ such that $x = \oplus_{i=0}^{d} x_i$ where d of the shares are sampled from independent uniform distribution.

There are usually two layers in masking schemes with $d + 1$ shares such as CMS and DOM. The first layer, which is so-called expansion layer, calculates all the shared product terms and refreshes them. The results of the first layer are stored in registers. The compression layer is the second layer, which compresses the refreshed product terms into $d + 1$ output shares again.

$$
\begin{array}{lll}
\hline
f_0 = a_0d_0 + b_0c_0 + b_0d_0 & \rightarrow [x'_0] & \\
f_1 = a_0d_1 + b_0c_1 + b_0d_1 + r_0 & \rightarrow [x'_1] & x'_0 + x'_1 = [x_0] \\
\hline
f_2 = a_1d_0 + b_1c_0 + b_1d_0 + r_0 & \rightarrow [x'_2] & \\
f_3 = a_1d_1 + b_1c_1 + b_1d_1 & \rightarrow [x'_3] & x'_2 + x'_3 = [x_1] \\
 & & \\
g_0 = a_0c_0 + b_0c_0 + a_0d_0 & \rightarrow [y'_0] & \\
g_1 = a_0c_1 + b_0c_1 + a_0d_1 + r_1 & \rightarrow [y'_1] & y'_0 + y'_1 = [y_0] \\
\hline
g_2 = a_1c_0 + b_1c_0 + a_1d_0 + r_1 & \rightarrow [y'_2] & \\
g_3 = a_1c_1 + b_1c_1 + a_1d_1 & \rightarrow [y'_3] & y'_2 + y'_3 = [y_1] \\
\hline
\end{array}
\tag{1}
$$

Equation 1 and Equation 2 give exemplary DOM and CMS implementation for $\mathbb{F}_2^2$ multiplication $\mathbf{M} : (x, y) = (a, b) \times (c, d)$, respectively, where $(a, b), (c, d)$ and (x, y) are elements in $\mathbb{F}_2^2$ and a, c, x are the least significant bits. In Equation 1 and Equation 2, $[\cdot]$ indicates a register stage. The registers are used to stop the propagation of glitches, which is critical to the security under glitch-extended probing model [FGM+18]. In this model, a probe placed at a wire enables the adversary to obtain all the stable values stored in registers that contribute to the computation of the value that wire carries. We call the set of stable values the *glitch-extended probing set* of the corresponding wire. If the glitch-extended probing set of any wire in a circuit is statistically independent of the secret variables, then the circuit is glitch-extended probing secure.

$$
\begin{array}{lll}
\hline
f_0 = a_0d_0 + b_0c_0 + b_0d_0 + r_0 & \rightarrow [x'_0] & \\
f_1 = a_0d_1 + b_0c_1 + b_0d_1 + r_1 & \rightarrow [x'_1] & x'_0 + x'_1 = x_0 \\
\hline
f_2 = a_1d_0 + b_1c_0 + b_1d_0 + r_2 & \rightarrow [x'_2] & \\
f_3 = a_1d_1 + b_1c_1 + b_1d_1 + r_0 + r_1 + r_2 & \rightarrow [x'_3] & x'_2 + x'_3 = x_1 \\
 & & \\
g_0 = a_0c_0 + b_0c_0 + a_0d_0 + r_3 & \rightarrow [y'_0] & \\
g_1 = a_0c_3 + b_0c_1 + a_0d_1 + r_4 & \rightarrow [y'_1] & y'_0 + y'_1 = y_0 \\
\hline
g_2 = a_1c_0 + b_1c_0 + a_1d_0 + r_5 & \rightarrow [y'_2] & \\
g_3 = a_1c_1 + b_1c_1 + a_1d_1 + r_3 + r_4 + r_5 & \rightarrow [y'_3] & y'_2 + y'_3 = y_1 \\
\hline
\end{array}
\tag{2}
$$

For instance, if the adversary place a probe at the *input wire* of the register that store x_0 in Equation 1, they have access to the joint distribution of x'_0 and x'_1 and the glitch-extended probing set of x_0 is $\{x'_0, x'_1\}$. If the adversary probe the *output wire* of the same register, the adversary only gets the distribution of x_0 since x_0 is a stable signal after storing it into registers. We denote these two situations by $\rho(x_0) = \{x'_0, x'_1\}$ and $\rho([x_0]) = \{x_0\}$ where ρ is a function that maps a wire to its glitch-extended probing set.

If the two output bits x and y of $\mathbf{M}$ are involved in the same nonlinear operation, e.g., x and y are multiplied together in a next step, it is necessary for DOM schemes to store the output shares of x, y into registers. Otherwise, when the adversary probes the cross domain term $x_0 \cdot y_1$, they can observe the joint distribution of $\{x'_0, x'_1, y'_2, y'_3\}$ whose subset $\{x'_0 = a_0d_0 + b_0c_0 + b_0d_0, y'_3 = a_1d_1 + b_1c_1 + b_1d_1\}$ leaks information about a, b, c, and d.

However, it is not necessary for CMS to store output shares into registers since the glitch-extended probing set of each term x_iy_j $(0 \leq i, j \leq 1)$ are statistically independent from a, b, c, and d.

2.4 Randomness-Free First-Order Masking Schemes

Shahmirzadi et al. introduced a randomness-free first-order masking schemes (RFFOMS) for quadratic and cubic functions in TCHES 2021 [SM21]. Wu et al. extended this technique with non-linear compression layers in TCHES 2025 [WFW+25]. The RFFOMS

fulfills four properties: (1) correctness, (2) non-completeness for each component function in expansion layer, (3) joint uniformity for the output shares of compression layer, (4) secret-independent distribution of glitch-extended probing set for each output share. Readers can refer to [WFW+25, §2.3] for details of these properties. While five properties are presented in [WFW+25, §2.3], we believe that the *balance* property can be implied by the joint uniformity of the output shares and thus we do not list it here.

Property (4) is an additional property compared to Threshold Implementations. We take Equation 3, which is a masking scheme for $\mathbf{M}$ which fulfills the four properties required by RFFOMS, as an example to illustrate this property. When the adversary places a probe at the output wire of the gate that calculates $x_0 = x'_0 + x'_1$, they have access to both x'_0 and x'_1, which necessitates the joint distribution of probing set $\{x'_0, x'_1\}$ should be statistically independent of the value of secrets.

$$
\begin{array}{lll}
\hline
f_0 = a_0d_0 + b_0c_0 + b_0d_0 & \rightarrow [x'_0] & \\
f_1 = a_0d_1 + b_0c_1 + b_0d_1 + d_1 & \rightarrow [x'_1] & x'_0 + x'_1 = [x_0] \\
\hline
f_2 = a_1d_0 + b_1c_0 + b_1d_0 & \rightarrow [x'_2] & \\
f_3 = a_1d_1 + b_1c_1 + b_1d_1 + d_1 & \rightarrow [x'_3] & x'_2 + x'_3 = [x_1] \\
 & & \\
g_0 = a_0c_0 + b_0c_0 + a_0d_0 & \rightarrow [y'_0] & \\
g_1 = a_0c_1 + b_0c_1 + a_0d_1 + c_1 & \rightarrow [y'_1] & y'_0 + y'_1 = [y_0] \\
\hline
g_2 = a_1c_0 + b_1c_0 + a_1d_0 & \rightarrow [y'_2] & \\
g_3 = a_1c_1 + b_1c_1 + a_1d_1 + c_1 & \rightarrow [y'_3] & y'_2 + y'_3 = [y_1] \\
\hline
\end{array}
\tag{3}
$$

Although the masking schemes in both [SM21] and [WFW+25] do not utilize fresh randomness, they suffer from the same latency issue as DOM: RFFOMS only requires that the glitch-extended probing set of each share is statistically independent of the secrets (property (4)). When shares of different output bits are combined, it is not ensured that the combinations will not leak information. For example, if the output shares of Equation 3 are not stored in registers and there is a combinational logic that performs the operation of $x_0 \wedge y_1$, the adversary can access $\{x'_0, x'_1, y'_2, y'_3\}$, which will leak information about a, b, c, and d.

Therefore, it is necessary for RFFOMS to store output shares in registers if the next nonlinear operations combines different output shares. This will increase the number of pipeline registers and thus have a negative impact on latency and area.

We would like to highlight a difference between DOM and RFFOMS. If linear operations are applied to the output shares of DOM schemes and then the results of the linear operations are stored in registers, then it is safe for DOM to omit storing these output shares in registers. However, this does not hold true for RFFOMS.For example, in Equation 1, it is safe for DOM to compute $x_0 + y_0$ and $x_1 + y_1$ directly without storing x_0, x_1, y_0, and y_1 in registers. On the contrary, if RFFOMS does so, the distribution of $\rho(x_0 + y_0) = \{x'_0, x'_1, y'_0, y'_1\}$ is not statistically independent of the secrets, which we have verified with the formal verification tool Prover.

Yao et al. proposed a technique to mitigate the latency issue of RFFOMS [YCW+24]. However, their technique is only applicable to quadratic Boolean functions. We will discuss more details about it in Section 3.

3 Low-latency First-Order Masking Schemes with Eliminated Randomness

3.1 Motivations: Improvements on RFFOMS by YCW+24

Yao et al. present a technique to reduce latency by using randomness that appears in the expansion layer but cancels out in the expression of output shares (compression layer) [YCW+24]. With the help of this randomness, it is possible to not store the output shares into registers when subsequent linear or nonlinear operation are performed over output shares, thus decreasing the latency.

They first explore the case of single-output Boolean functions. They observed that when two Boolean random variables u_0 and u_1 satisfying $\Pr[(u_0, u_1) = (0, 0)] + \Pr[(u_0, u_1) = (1, 1)] = \alpha$ (i.e., $\Pr[u_0 + u_1 = 0] = \alpha$) where α is a fixed constant, then the joint distribution of $u_0 + r, u_1 + r$ where r is a uniform random bit independent of u_0 and u_1 is also a fixed distribution which only depends on α [YCW+24, Proposition 1]. They use this observation to design masking schemes for single-output Boolean functions. See Equation 4 for example, where the first four lines is a masking scheme for single-output Boolean function $x = f(a, b, c, d) = ad + bc + bd$. By this observation, it is enough to search for a masking scheme that fulfills both $f_0 + f_1$ and $f_2 + f_3$ are statistically independent of a, b, c, and d while the designs given by RFFOMS requires both $\{f_0, f_1\}$ and $\{f_2, f_3\}$ are independent of a, b, c, and d.

However, Yao et al. did not provide a deeper theoretical analysis for such constructions of multi-output Boolean functions. Instead, they found low-latency solutions by a search algorithm according to the definition of glitch-extended probing security [YCW+24, Proposition 2]. In their searching algorithms, they add constraints that allow the output shares from different coordinate functions to be combined freely without storing them into registers.

$$
\begin{array}{lll}
f_0 = a_0d_0 + b_0c_0 + b_0d_0 & \to f_0{+}r_0 = [x'_0] & \\
f_1 = a_0d_1 + b_0c_1 + b_0d_1 + d_1 & \to f_1{+}r_0 = [x'_1] & x'_0 + x'_1 = x_0 \\
\hline
f_2 = a_1d_0 + b_1c_0 + b_1d_0 & \to f_2{+}r_0 = [x'_2] & \\
f_3 = a_1d_1 + b_1c_1 + b_1d_1 + d_1 & \to f_3{+}r_0 = [x'_3] & x'_2 + x'_3 = x_1 \\
 & & \\
g_0 = a_0c_0 + b_0c_0 + a_0d_0 & \to g_0{+}r_1 = [y'_0] & \\
g_1 = a_0c_1 + b_0c_1 + a_0d_1 + c_1 & \to g_1{+}r_1 = [y'_1] & y'_0 + y'_1 = y_0 \\
\hline
g_2 = a_1c_0 + b_1c_0 + a_1d_0 & \to g_2{+}r_1 = [y'_2] & \\
g_3 = a_1c_1 + b_1c_1 + a_1d_1 + c_1 & \to g_3{+}r_1 = [y'_3] & y'_2 + y'_3 = y_1
\end{array}
\tag{4}
$$

Equation 4 shows a solution for masking scheme of $\mathbf{M}$ given by the search algorithms in [YCW+24]. It can be observed that it happens to be the scheme of Equation 3, which fulfills the properties required by [SM21], with each component function of x and y adding the same random bits r_0 and r_1, respectively.

3.2 Extending the technique

Indeed, [YCW+24] provides a valuable insight on the constructions of masking schemes for single-output quadratic Boolean functions. We now generalize it from the perspective of distribution of Boolean random variables with the help of *valid sharings of zero* and the concept of *parity* of a set of Boolean random variables.

We write $\mathcal{R} = \{r_0, \ldots, r_{n-1}\}$ for a *valid sharing of zero* if $r_0, \ldots, r_{n-2}$ are independent uniform bits and $r_{n-1} = \bigoplus_{i=0}^{n-2} r_i$. For any finite set $\mathcal{S}$ of bits, its *parity* is $\mathrm{par}(\mathcal{S}) := \bigoplus_{x \in \mathcal{S}} x$.

Generalization to single-output Boolean functions. Firstly, we observe that: given two sets of Boolean random variables $\mathcal{U}$ and $\mathcal{V}$, if the parity of $\mathcal{U}$ is statistically independent of $\mathcal{V}$, then $\mathcal{U}$ masked by a valid sharing of zero is statistically independent of $\mathcal{V}$. This is formalized in Proposition 1.

Proposition 1. *Let $\mathcal{U} = \{u_0, \ldots, u_{n-1}\}$ and $\mathcal{V} = \{v_0, \ldots, v_{m-1}\}$ be sets of Boolean random variables, and let $\mathcal{R} = \{r_0, \ldots, r_{n-1}\}$ be a valid sharing of zero (sampled independently of $\mathcal{U}, \mathcal{V}$). Assume $\mathrm{par}(\mathcal{U})$ is statistically independent of $\mathcal{V}$. Define the masked set $\mathcal{W} = \{w_i = u_i \oplus r_i : 0 \le i < n\}$. Then $\mathcal{W}$ is statistically independent of $\mathcal{V}$.*

To prove Proposition 1, we introduce three lemmas. The first one states the property of a valid sharing of zero. The second lemma states the simple property of a random mask. The third one provides a sufficient and necessary condition to judge that two Boolean variable sets are statistically independent.

Lemma 1. *Let $\mathcal{R} = \{r_0, \ldots, r_{n-1}\}$ be a valid sharing of zero and let $\varnothing \neq \mathcal{R}' \subseteq \mathcal{R}$. Then $\mathrm{par}(\mathcal{R}')$ is either 0 or a uniform bit that is independent of any collection of random variables independent of $\mathcal{R}$. Moreover, $\mathrm{par}(\mathcal{R}') = 0$ if and only if $\mathcal{R}' = \mathcal{R}$.*

Proof. If $r_{n-1} \notin \mathcal{R}'$, then $\mathrm{par}(\mathcal{R}')$ is the XOR of a nonempty subcollection of the independent uniform bits $r_0, \ldots, r_{n-2}$, hence uniform. Since these bits are independent of any external collection that is independent of $\mathcal{R}$, the resulting uniform bit is independent of that external collection as well.

If $r_{n-1} \in \mathcal{R}'$, write $\mathcal{I} := \{0, \ldots, n-2\}$ and $T := \{i \mid r_i \in \mathcal{R}' \wedge i \neq n-1\}$. Then

$$\mathrm{par}(\mathcal{R}') = r_{n-1} \oplus \bigoplus_{i \in T} r_i = \bigoplus_{i \in \mathcal{I}} r_i \oplus \bigoplus_{i \in T} r_i = \bigoplus_{i \in (\mathcal{I} \setminus T)} r_i.$$

This XOR is 0 exactly when $\mathcal{I} \setminus T = \varnothing$, i.e., when $T = \mathcal{I}$ (equivalently $\mathcal{R}' = \mathcal{R}$); otherwise it is the XOR of a nonempty set of independent uniform bits, hence uniform and independent of any collection independent of $\mathcal{R}$. □

Lemma 2. *Let r be a uniform bit and let $\mathscr{F}$ be a σ-field such that r is independent of $\mathscr{F}$. If u is any $\{0,1\}$-valued random variable that is $\mathscr{F}$-measurable, then $y := r \oplus u$ is a uniform bit independent of $\mathscr{F}$. In particular, if r is independent of u, then $r \oplus u$ is uniform and independent of u.*

Proof. For any $b \in \{0,1\}$,

$$\Pr[y = b \mid \mathscr{F}] = \Pr[r \oplus u = b \mid \mathscr{F}] = \Pr[r = b \oplus u \mid \mathscr{F}] = \tfrac{1}{2},$$

because u is $\mathscr{F}$-measurable and r is uniform and independent of $\mathscr{F}$. Thus y is conditionally uniform given $\mathscr{F}$, hence uniform and independent of $\mathscr{F}$. Taking $\mathscr{F} = \sigma(u)$ yields the particular case. □

Lemma 3 was proved in [CFW04], so we omit the proof here. The original statement is phrased for Boolean *vectors* and speaks of all nonzero linear combinations of the coordinates. Our presentation recasts it for *sets* and speaks of the *parity of nonempty subsets*. The two formulations are equivalent: every nonzero linear combination corresponds to the parity of some nonempty subset, and conversely every such parity is a nonzero linear combination.

Lemma 3 ([CFW04, Lemma 5] parity form)**.** *Let $\mathcal{U}$ and $\mathcal{V}$ be two finite sets of Boolean random variables. Then $\mathcal{U}$ is statistically independent of $\mathcal{V}$ if and only if for every nonempty subset $\mathcal{U}' \subseteq \mathcal{U}$ the parity $\mathrm{par}(\mathcal{U}')$ is statistically independent of $\mathcal{V}$.*

With these three lemma, here comes the proof of Proposition 1.

Proof of Proposition 1. By Lemma 3, it suffices to show that for every nonempty $\mathcal{W}' \subseteq \mathcal{W}$, $\mathrm{par}(\mathcal{W}')$ is independent of $\mathcal{V}$. Let $\mathcal{U}' = \{u_i : w_i \in \mathcal{W}'\}$ and $\mathcal{R}' = \{r_i : w_i \in \mathcal{W}'\}$. Then $\mathrm{par}(\mathcal{W}') = \mathrm{par}(\mathcal{U}') \oplus \mathrm{par}(\mathcal{R}')$.

By Lemma 1 (with the independence clause), $\mathrm{par}(\mathcal{R}')$ is either 0 or a uniform bit independent of $\sigma(\mathcal{U}, \mathcal{V})$ because $\mathcal{R}$ is sampled independently of $(\mathcal{U}, \mathcal{V})$.

If $\mathrm{par}(\mathcal{R}') \neq 0$, then applying Lemma 2 with $\mathscr{F} = \sigma(\mathcal{U}, \mathcal{V})$ and $u = \mathrm{par}(\mathcal{U}')$ shows that $\mathrm{par}(\mathcal{W}')$ is uniform and independent of $\mathcal{V}$.

If $\mathrm{par}(\mathcal{R}') = 0$, Lemma 1 implies $\mathcal{R}' = \mathcal{R}$, hence $\mathcal{U}' = \mathcal{U}$ and $\mathrm{par}(\mathcal{W}') = \mathrm{par}(\mathcal{U})$, which is independent of $\mathcal{V}$ by assumption. Thus in all cases $\mathrm{par}(\mathcal{W}')$ is independent of $\mathcal{V}$, and the claim follows from Lemma 3. □

Proposition 1 provides a generalized explanation for masking schemes of single-output Boolean functions provided in [YCW+24], making it possible to extending this technique to Boolean functions with a higher degree (see Subsection 3.4 for more details). However, it does not cover the cases of multi-output Boolean functions.

Generalization to multi-output Boolean functions. The conclusion in Proposition 1 can be generalized to multiple sets and thus providing a suitable explanation for the masking schemes of multi-output Boolean functions. We observe that: given $t+1$ disjoint sets, $\mathcal{U}^{(0)}, \cdots, \mathcal{U}^{(t-1)}$, and $\mathcal{V}$, if set of parities $\{\mathrm{par}(\mathcal{U}^{(j)})\}_{j=0}^{t-1}$ is statistically independent of $\mathcal{V}$, then the union set of each $\mathcal{U}^{(j)}$ masked by independent valid sharings of zero, denoted by $\mathcal{W}$, is statistically independent of $\mathcal{V}$.

The observation for multiple sets is formalized in Proposition 2. To help prove Proposition 2, we first introduce Lemma 4 to state the property of the union set of different valid sharings of zero. The proof for Lemma 4 and Proposition 2 are very similar to the ones of Lemma 1 and Proposition 1, we provide them for the sake of completeness.

Lemma 4. *Let $\mathcal{R}^{(0)}, \ldots, \mathcal{R}^{(t-1)}$ be mutually independent valid sharings of zero (possibly of different lengths), and set $\mathcal{R} := \bigcup_{j=0}^{t-1} \mathcal{R}^{(j)}$. For any nonempty $\mathcal{S} \subseteq \mathcal{R}$, write $\mathcal{S}^{(j)} := \mathcal{S} \cap \mathcal{R}^{(j)}$. Then $\mathrm{par}(\mathcal{S})$ is either 0 or a uniform bit that is independent of any collection of random variables independent of $\mathcal{R}$. Moreover, $\mathrm{par}(\mathcal{S}) = 0$ if and only if for every j, $\mathcal{S}^{(j)} \in \{\varnothing, \mathcal{R}^{(j)}\}$ (and not all $\mathcal{S}^{(j)}$ are empty, since $\mathcal{S} \neq \varnothing$).*

Proof. We have $\mathrm{par}(\mathcal{S}) = \bigoplus_{j=0}^{t-1} \mathrm{par}(\mathcal{S}^{(j)})$. By Lemma 1, each summand is either 0 or a uniform bit that is independent of any collection independent of $\mathcal{R}^{(j)}$; in particular, of any collection independent of $\mathcal{R}$. If some $\mathrm{par}(\mathcal{S}^{(j)})$ is uniform, then the XOR of independent bits that includes a uniform bit is uniform; hence $\mathrm{par}(\mathcal{S})$ is uniform and independent of any collection independent of $\mathcal{R}$. Conversely, $\mathrm{par}(\mathcal{S}) = 0$ occurs exactly when every $\mathrm{par}(\mathcal{S}^{(j)}) = 0$, i.e., when each $\mathcal{S}^{(j)}$ is either $\varnothing$ or $\mathcal{R}^{(j)}$ (and at least one is $\mathcal{R}^{(j)}$ because $\mathcal{S} \neq \varnothing$). □

Proposition 2. *Let $t \geq 1$. For $0 \leq j < t$ let $\mathcal{U}^{(j)}$ be disjoint sets of Boolean random variables with $|\mathcal{U}^{(j)}| = n_j \geq 2$, and let $\mathcal{V}$ be another set. Assume the set $\{\mathrm{par}(\mathcal{U}^{(j)})\}_{j=0}^{t-1}$ is statistically independent of $\mathcal{V}$. For each j let $\mathcal{R}^{(j)}$ be an independent valid sharing of zero (independent of all $\mathcal{U}^{(i)}, \mathcal{V}$), and define $\mathcal{W}^{(j)} = \{u \oplus r : u \in \mathcal{U}^{(j)}, r \in \mathcal{R}^{(j)}\}$. Then the union $\mathcal{W} = \bigcup_{j=0}^{t-1} \mathcal{W}^{(j)}$ is statistically independent of $\mathcal{V}$.*

Proof. By Lemma 3, it suffices to show that for every nonempty $\mathcal{W}' \subseteq \mathcal{W}$, $\mathrm{par}(\mathcal{W}')$ is independent of $\mathcal{V}$. Write $\mathcal{W}' = \bigcup_{j=0}^{t-1} \mathcal{W}'^{(j)}$ with $\mathcal{W}'^{(j)} \subseteq \mathcal{W}^{(j)}$, and let $\mathcal{U}'^{(j)} := \{u \in \mathcal{U}^{(j)} : \exists r \in \mathcal{R}^{(j)}$ with $u \oplus r \in \mathcal{W}'^{(j)}\}$, $\mathcal{R}'^{(j)} := \{r \in \mathcal{R}^{(j)} : \exists u \in \mathcal{U}^{(j)}$ with $u \oplus r \in \mathcal{W}'^{(j)}\}$. Then

$$\mathrm{par}(\mathcal{W}') = \bigoplus_{j=0}^{t-1} \left(\mathrm{par}\left(\mathcal{U}'^{(j)}\right) \oplus \mathrm{par}\left(\mathcal{R}'^{(j)}\right)\right) = \left(\bigoplus_{j=0}^{t-1} \mathrm{par}\left(\mathcal{U}'^{(j)}\right)\right) \oplus \underbrace{\left(\bigoplus_{j=0}^{t-1} \mathrm{par}\left(\mathcal{R}'^{(j)}\right)\right)}_{=:R^\star}.$$

By Lemma 4 (with independence), $R^\star$ is either 0 or a uniform bit independent of $\sigma(\{\mathcal{U}^{(j)}\}_j, \mathcal{V})$.

If $R^\star \neq 0$, Lemma 2 with $\mathscr{F} = \sigma(\{\mathcal{U}^{(j)}\}_j, \mathcal{V})$ and $u = \bigoplus_j \mathrm{par}\big(\mathcal{U}'^{(j)}\big)$ gives that $\mathrm{par}(\mathcal{W}')$ is uniform and independent of $\mathcal{V}$.

If $R^\star = 0$, Lemma 4 implies that for each j, either $\mathcal{U}'^{(j)} = \varnothing$ or $\mathcal{U}'^{(j)} = \mathcal{U}^{(j)}$ (and at least one is $\mathcal{U}^{(j)}$ since $\mathcal{W}' \neq \varnothing$). Hence $\mathrm{par}(\mathcal{W}') = \bigoplus_{j\in S} \mathrm{par}\big(\mathcal{U}^{(j)}\big)$ for some nonempty index set S. By the assumption that the vector $\big(\mathrm{par}\big(\mathcal{U}^{(0)}\big), \ldots, \mathrm{par}\big(\mathcal{U}^{(t-1)}\big)\big)$ is independent of $\mathcal{V}$, any nontrivial XOR of its coordinates is independent of $\mathcal{V}$. Therefore $\mathrm{par}(\mathcal{W}')$ is independent of $\mathcal{V}$ in this case as well.

By Lemma 3, $\mathcal{W}$ is statistically independent of $\mathcal{V}$. □

Illustration. Now, we use Proposition 1 and Proposition 2 to illustrate the relationship between Equation 3 and Equation 4.

When using RFFOMS (i.e., Equation 3), apart from correctness and non-completeness, it is ensured that:

i. Uniformity: each of the four sets $\{x_0, y_0\}$, $\{x_0, y_1\}$,$\{x_1, y_0\}$, $\{x_1, y_1\}$ follows a jointly uniform distribution.

ii. Secret-independent distribution of glitch-extended probing set for each output share: the joint distribution of $\{x'_0, x'_1\}$, $\{x'_2, x'_3\}$, $\{y'_0, y'_1\}$, $\{y'_2, y'_3\}$ are statistically independent of secret variable set $\{a, b, c, d\}$, respectively.

When using masking schemes given by Yao et al. (i.e., Equation 4), apart from correctness and non-completeness, it is ensured that:

I. Uniformity: each of the four sets $\{x_0, y_0\}$, $\{x_0, y_1\}$,$\{x_1, y_0\}$, $\{x_1, y_1\}$ follows a jointly uniform distribution.

II. the joint distribution of the following sets are statistically independent of secret variable set $\{a, b, c, d\}$, respectively (this is ensured by the search algorithms to reduce the latency).

(a) $\rho(x_0, y_0) = \{x'_0, x'_1, y'_0, y'_1\} = \{f_0 + r_0, f_1 + r_0, g_0 + r_1, g_1 + r_1\}$
(b) $\rho(x_0, y_1) = \{x'_0, x'_1, y'_2, y'_3\} = \{f_0 + r_0, f_1 + r_0, g_2 + r_1, g_3 + r_1\}$
(c) $\rho(x_1, y_0) = \{x'_2, x'_3, y'_0, y'_1\} = \{f_2 + r_0, f_3 + r_0, g_0 + r_1, g_1 + r_1\}$
(d) $\rho(x_1, y_1) = \{x'_2, x'_3, y'_2, y'_3\} = \{f_2 + r_0, f_3 + r_0, g_2 + r_1, g_3 + r_1\}$

However, we observe that Property (II) can be inferred from Property (I) (uniformity).

1. Since $x_0 = x'_0 + x'_1 = f_0 + f_1$, the parity of $\mathcal{U} = \{f_0, f_1\}$ equals x_0, which is uniform and thus statistically independent of $\mathcal{V} = \{a, b, c, d\}$, it follows from Proposition 1 that $\mathcal{W} = \{f_0 + r_0, f_1 + r_0\}$ is statistically independent of $\mathcal{V} = \{a, b, c, d\}$ because $\mathcal{U}$ is masked by a sharing of zero (r_0, r_0). This indicates that a correct, non-complete, and uniform masking schemes for a single-output (quadratic) Boolean function using the randomness usage method by Yao et al. will automatically fulfills property (4) required by RFFOMS.

2. Since the output shares $\{x_0, y_0\}$, also the parities of $\mathcal{U}^{(0)} = \{f_0, f_1\}$ and $\mathcal{U}^{(1)} = \{g_0, g_1\}$, are jointly uniform and thus statistically independent of $\{a, b, c, d\}$, it follows from Proposition 2 that $\mathcal{W} = \{f_0 + r_0, f_1 + r_0, g_0 + r_1, g_1 + r_1\}$ is statistically independent of $\mathcal{V} = \{a, b, c, d\}$. Therefore the set in case (IIa) is secure. The same analysis applies to the sets in case (IIb), (IIb), and (IId).

Generalization to uniformity. In fact, since the output sharing is uniform for the schemes in [YCW+24], we can prove that the union of glitch-extended probing set of the jointly uniform output shares also follows a uniform distribution, i.e., each set in case (IIa) to (IId) follows a uniform distribution, respectively.

Intuitively, if the parity $\mathrm{par}(\mathcal{U})$ of a set of Boolean variables $\mathcal{U}$ is uniformly distributed, then masking $\mathcal{U}$ with a valid sharing of zero yields a family of Boolean variables that is jointly uniform. The same intuition extends to multiple sets. We formalize these statements in Proposition 3 and Proposition 4. Their proofs follow from Lemma 5 (see [SZZ95, Theorem 5]) together with Lemma 1, Lemma 2, and Lemma 4, and they closely parallel the arguments used in the proofs of Proposition 1 and Proposition 2. We therefore omit the details.

Lemma 5 (XOR Lemma [SZZ95])**.** *A set of Boolean random variables $\mathcal{U} = \{u_0, u_1, \cdots, u_{n-1}\}$ follows an jointly uniform distribution if and only if for every nonempty subset $\mathcal{U}' \subseteq \mathcal{U}$ the parity $\mathrm{par}(\mathcal{U}')$ follows a uniform distribution.*

Remark 2. As with Lemma 3, the original formulation of Lemma 5 uses the terminology of nonzero linear combinations of a Boolean vector, whereas we work with sets and speak of the parity of a nonempty subset; the two phrasings are equivalent.

Proposition 3. *Let $\mathcal{U} = \{u_0, \ldots, u_{n-1}\}$ be a set of Boolean random variables, and let $\mathcal{R} = \{r_0, \ldots, r_{n-1}\}$ be a valid sharing of zero (sampled independently of $\mathcal{U}$). Assume $\mathrm{par}(\mathcal{U})$ follows a uniform distribution. Define the masked set $\mathcal{W} = \{w_i = u_i \oplus r_i : 0 \le i < n\}$. Then $\mathcal{W}$ is jointly uniform.*

Proposition 4. *Let $t \ge 1$. For $0 \le j < t$, let $\mathcal{U}^{(j)}$ be disjoint sets of Boolean random variables with $|\mathcal{U}^{(j)}| = n_j \ge 2$. Assume the Boolean variables in the parity set $\{\mathrm{par}(\mathcal{U}^{(j)})\}_{j=0}^{t-1}$ are jointly uniform. For each j let $\mathcal{R}^{(j)}$ be an independent valid sharing of zero (independent of all $\mathcal{U}^{(i)}$), and define $\mathcal{W}^{(j)} = \{u \oplus r : u \in \mathcal{U}^{(j)}, r \in \mathcal{R}^{(j)}\}$. Then the Boolean variables in the union $\mathcal{W} = \bigcup_{j=0}^{t-1} \mathcal{W}^{(j)}$ are jointly uniform.*

3.3 First-Order Low-latency Threshold Implementations

Based on the conclusions presented in last section, we propose a method to design first-order low-latency Threshold Implementations for Boolean functions with only two shares. Our constructions fulfill four properties: (1) correctness; (2) non-completeness of expansion layer; (3) joint uniformity of output shares; (4) domain-separated refreshing in expansion layer – for each output bit, the uncompressed components of its two shares are refreshed with a valid sharing of zero that is identical for the two shares but unique to that output bit.

Rather than providing explicit search algorithms for designs that fulfills the first three properties (the fourth property can be satisfied trivially), we offer only a conceptual outline in the following, since the constructions for the AES S-Box proposed in this work can be obtained without the aid of such algorithms.

Constructions for single-output Boolean functions. Let $f : \mathbb{F}_2^n \to \mathbb{F}_2$ be a single-output Boolean function, which maps $x \in \mathbb{F}_2^n$ to $y \in \mathbb{F}_2$. Suppose f have a algebraic degree of t (typically t equals 2 or 3). Then our low-latency TI for f involves three phases and fulfills the above four properties.

The calculation phase. This phase calculates all the monomial and combines them into 2^t component functions (with each component function fulfill non-completeness)[1], which will be compressed into 2 output shares with each output share contains 2^{t-1} component functions. As a side note, we are only considering a linear compression layer where the

[1]Correction terms are possibly added into component functions in order to make a uniform sharing.

compression is simply the bit XOR of the 2^{t-1} component functions. The nonlinear compression technique presented in [WFW+25] is beyond the scope of this paper since the (domain-separated) refreshing technique is not applicable to nonlinear compression.

The domain-separated refreshing phase. For the two groups of 2^{t-1} component functions, refresh the results with the same valid sharing of zero $\mathcal{R} = (r_0, \cdots, r_{2^{t-1}-1}, \bigoplus_{i=0}^{2^{t-1}-1} r_i)$. This process needs $2^{t-1} - 1$ random bits. The results of this phase should be stored in registers.

The compression phase. Finally, the results of the domain-separated refreshing phase are compressed into 2 output shares. Each output share absorbs 2^{t-1} component functions. The two output shares should form a uniform sharing of y.

Constructions for multi-output Boolean functions. Let f be a Multi-Output Boolean function $f : \mathbb{F}_2^n \mapsto \mathbb{F}_2^m$, which maps $x \in \mathbb{F}_2^n$ to $y \in \mathbb{F}_2^m$. f contains m single-output Boolean functions (coordinate functions) sharing the same inputs x.

The low-latency TI for f could be constructed in two steps. Firstly, construct low-latency TI for each coordinate functions. Secondly, combining the TIs of coordinate function and ensure that they make a uniform sharing of y.

Indeed, the combinations of TIs of coordinate functions might not trivially make a uniform output sharing. Therefore, a searching algorithm might be needed to find such solutions. Note that the first three of the four properties required by RFFOMS presented in Subsection 2.4 are the same as the first three properties required by the low-latency TI proposed by us. This indicates that if we apply the domain-separated refreshing to the solutions given by the search algorithms in [SM21], we can obtain a design that fulfills the definition of our low-latency TI. However, the solutions found by the search algorithms by [SM21] fulfills a fourth constraint that is not required by us. Therefore, we believe that removing the fourth constraint in the algorithms presented in [SM21] would be adequate to find solutions that fulfill the definition of our constructions.

Nevertheless, during the designing process of masked $\mathbb{F}_2^8$ inverter, we did not utilize any algorithms such as the one described in [SM21], nor adaptations of it. Therefore we leave the development of such an algorithm as a future work.

Security analysis of our constructions. We allow arbitrary post-processing on the output shares of a multi-output Boolean function f. Concretely, we assume the adversary may combine *one* share from *each* coordinate function in any way, but may not take both shares of the *same* coordinate at once. We refer to such combinations as *arbitrary non-complete operations*. For example, in Equation 4 the adversary may combine one of $\{x_0, x_1\}$ with one of $\{y_0, y_1\}$, but cannot access x_0 and x_1 simultaneously.

Theorem 1. *The low-latency Threshold Implementations for an individual multi-output Boolean function constructed in this section*

(a) satisfy glitch-extended probing security; and

(b) permit arbitrary non-complete operations on the output shares without *storing those output shares in registers.*

Proof. (a) If the adversary probes *before* the register stage, the resulting glitch-extended probing set is non-complete by construction and is therefore secure. If the probing is *after* the register stage, through one probe they obtain the glitch-extended probing set of only a single share of a single output bit. Since this share is uniform, the corresponding glitch-extended probing set is jointly uniform by Proposition 3; hence the circuit is glitch-extended probing secure.

(b) Write the m output bits of f as $y^{(0)}, \ldots, y^{(m-1)}$, and denote their two-share encodings by $\boldsymbol{y}^{(j)} = (y_0^{(j)}, y_1^{(j)})$. Since the output sharing is uniform, for any selection vector $\boldsymbol{i} = (i_0, \ldots, i_{m-1}) \in \{0,1\}^m$ the set $\{y_{i_0}^{(0)}, \ldots, y_{i_{m-1}}^{(m-1)}\}$ is jointly uniform. By

Proposition 4, the union of their glitch-extended probing sets $\bigcup_{j=0}^{m-1} \rho\left(y_{i_j}^{(j)}\right)$ is also jointly uniform. Hence arbitrary non-complete operations over the output shares do not require storing those shares in registers.

□

As a consequence, our low-latency TIs for multi-output Boolean functions are *locally* glitch-extended probing secure. Here, locally means the guarantee holds for a single module in isolation; no composability guarantee is claimed when composing two modules protected by our low-latency TI that share some common inputs.

3.4 Applications to Cubic Functions

Now we provide examples that utilize the results of Proposition 1 to Proposition 4 to design masking schemes for cubic functions.

3.4.1 Applications to Single-Output Cubic Functions

A masking scheme for cubic function three-input AND $x = abc$ is provided in [SM21, §3.2], as shown in Equation 5. However, it needs a lot of correction terms to maintain statistically independence between secret inputs and glitch-extended probing set of output shares. This will increase the area of the circuits.

$$\begin{array}{llll}
f_0 = a_0 + a_0b_0 + a_0b_0c_0 & \to [x'_0] & \\
f_1 = a_0 + a_0b_0 + a_0c_1 + a_0b_0c_1 & \to [x'_1] & \\
f_2 = b_1c_0 + a_0b_1c_0 & \to [x'_2] & \\
f_3 = c_1 + a_0c_1 + b_1c_1 + a_0b_1c_1 & \to [x'_3] & x'_0 + x'_1 + x'_2 + x'_3 = x_0 \\
\hline
f_4 = a_1 + a_1b_0 + a_1b_0c_0 & \to [x'_4] & x'_4 + x'_5 + x'_6 + x'_7 = x_1 \\
f_5 = a_1 + a_1b_0 + a_1c_1 + a_1b_0c_1 & \to [x'_5] & \\
f_6 = b_1c_0 + a_1b_1c_0 & \to [x'_6] & \\
f_7 = c_1 + a_1c_1 + b_1c_1 + a_1b_1c_1 & \to [x'_7] &
\end{array} \tag{5}$$

Instead of adding a lot of correction terms to achieve first-order glitch-extended probing security, we apply our technique and provide a alternative solution for a masked AND3 gate. As shown in Equation 6, we add the correction term c_0 to f_2 and f_6. Since $x_0 = \oplus_{i=0}^{3} f_i = a_0bc + c_0$ where c_0 can be viewed as a uniform random mask, x_0 is uniformly distributed. The same holds true for $x_1 = a_1bc + c_0$. Therefore, (x_0, x_1) is a uniform sharing of x.

$$\begin{array}{llll}
f_0 = a_0b_0c_0 & \to & f_0{+}r_0 = [x'_0] & \\
f_1 = a_0b_0c_1 & \to & f_1{+}r_1 = [x'_1] & \\
f_2 = a_0b_1c_0{+}c_0 & \to & f_2{+}r_2 = [x'_2] & \\
f_3 = a_0b_1c_1 & \to & f_3{+}r_0 + r_1 + r_2 = [x'_3] & x'_0 + x'_1 + x'_2 + x'_3 = x_0 \\
\hline
f_4 = a_1b_0c_0 & \to & f_4{+}r_0 = [x'_4] & x'_4 + x'_5 + x'_6 + x'_7 = x_1 \\
f_5 = a_1b_0c_1 & \to & f_5{+}r_1 = [x'_5] & \\
f_6 = a_1b_1c_0{+}c_0 & \to & f_6{+}r_2 = [x'_6] & \\
f_7 = a_1b_1c_1 & \to & f_7{+}r_0 + r_1 + r_2 = [x'_7] &
\end{array} \tag{6}$$

Since the parity of $\mathcal{U} = \{f_0, f_1, f_2, f_3\}$ (resp. $\mathcal{U} = \{f_4, f_5, f_6, f_7\}$) equals x_0 (resp. x_1), which is uniform and thus statistically independent of $\mathcal{V} = \{a, b, c\}$, the set $\mathcal{W} = \{f_0{+}r_0, f_1{+}r_1, f_2{+}r_2, f_3{+}r_1{+}r_2{+}r_3\}$ (resp. $\mathcal{W} = \{f_4{+}r_0, f_5{+}r_1, f_6{+}r_2, f_7{+}r_0{+}r_1{+}r_2\}$) is uniform and statistically independent of $\{a, b, c\}$ according to Proposition 3. Compared to the solution given in [SM21], although our solution requires additional randomness, it do need less logic gates and thus potentially having a less area footprint than [SM21]. On

the other hand, the randomness requirements for our constructions could be relaxed when they are applied in an implementation of a cryptographic algorithm with the help of a similar technique to changing of the guards, which will be illustrated by the applications to AES presented in Section 4.

Remark 3. In Equation 6, the zero sharing $\{r_0, r_1, r_2, r_0 \oplus r_1 \oplus r_2\}$ generated from three random bits $\{r_0, r_1, r_2\}$ can be replaced by a zero sharing constructed from more than three random bits. For example, $\{r_0 \oplus r_1,\ r_1 \oplus r_2,\ r_2 \oplus r_3,\ r_3 \oplus r_0\}$ generated from $\{r_0, r_1, r_2, r_3\}$ also satisfies the property stated in Lemma 1, and is therefore a valid form of zero sharing for our constructions. However, such alternative forms of zero sharing require more random bits than the current construction.

3.4.2 Applications to Multi-Output Boolean Functions

We now use the $\mathbb{F}_2^4$ inverter (see in Equation 7, where m and d are MSB) to illustrate the applications in multi-output cubic functions.

$$\begin{aligned} m &= c + ac + ad + bd + bcd \\ n &= c + d + ad + bd + acd \\ z &= a + ac + bc + bd + abd \\ t &= a + b + bc + bd + abc \end{aligned} \tag{7}$$

Although our constructions in Subsection 3.3 do not provide a general method to enforce the third property (joint uniformity of the output sharing), this property is commonly achieved in existing Threshold Implementations. We therefore surveyed published $(d{+}1)$-share Threshold Implementations for the $\mathbb{F}_2^4$ inverter. To the best of our knowledge, there are only three candidates: [UHA17], [SM21], and [WFW+25]. The work of [SM21] targets an affine-equivalent version of the $\mathbb{F}_2^4$ inverter, and [WFW+25] employs nonlinear compressionboth outside the scope of our approach. Fortunately, the remaining option [UHA17], after removing fresh randomness, still satisfies joint uniformity of the output sharing. Hence, we base our $\mathbb{F}_2^4$ inverter design on that construction.

Our design for the masked $\mathbb{F}_2^4$ inverter is shown in Equation 8, where the circuit without the gray parts is taken from [UHA17]. The original scheme uses a refreshing technique like first-order CMS. The eight component functions is refreshed by $r_0, \cdots, r_6, \oplus_{i=0}^{6} r_i$ for each coordinate function. Therefore, it uses a total of 28 fresh random bits for the four coordinate functions. However, our construction only needs 12 random bits to achieve the same security level and latency as the scheme provided in [UHA17].

From the proof of Theorem 1, we know that when the adversary probes any combination from $(m_{i_0}, n_{i_1}, z_{i_2}, t_{i_3})$ with $i_0, i_1, i_2, i_3 \in \{0, 1\}$, the union of the glitch-extended probing set $\rho(m_{i_0}) \cup \rho(n_{i_1}) \cup \rho(z_{i_2}) \cup \rho(t_{i_3})$ follows a uniform distribution. For example, if a later module use m_0, n_0, z_0, t_0 as inputs, the adversary gain access to $\{m'_0, m'_1, m'_2, m'_3\}$, $\{n'_0, n'_1, n'_2, n'_3\}$, $\{z'_0, z'_1, z'_2, z'_3\}$, and $\{t'_0, t'_1, t'_2, t'_3\}$ at the same time. The 16 bits are jointly uniform.

Remark 4. In Proposition 2, it is required that the zero sharings should be statistically independent of each other, which means the 12 random bits in Equation 8 should be independent of each other. Although one might want to reuse the masks and thus save randomness, this is not allowed. For example, if we first sample 6 independent random bits $r_0, \cdots, r_5$ and then assign $r_{6+i} = r_i + r_{i+3}, r_{9+i} = r_i + r_{5-i}$ for $0 \le i \le 2$, then the security of this design will be broken since the used randomness violates the assumptions of Proposition 2.

$$
\begin{array}{llcrr}
f_0 = b_0c_0d_0 + b_0d_0 + c_0 & = c_0 \vee b_0d_0 & \rightarrow & f_0{+}r_0 = [m'_0] & \\
f_1 = b_0c_1d_0 + a_0c_1 & & \rightarrow & f_1{+}r_1 = [m'_1] & \\
f_2 = b_1c_0d_0 + a_0c_0 + a_0d_0 & = c_0(b_1d_0 + a_0) + a_0d_0 & \rightarrow & f_2{+}r_2 = [m'_2] & m_0 = m'_0 + m'_1 \\
f_3 = b_1c_1d_0 + b_1d_0 + a_1d_0 & = b_1d_0\bar{c_1} + a_1d_0 & \rightarrow & f_3{+}r_0{+}r_1{+}r_2 = [m'_3] & +m'_2 + m'_3 \\
\hline
f_4 = b_0c_0d_1 + b_0d_1 + a_0d_1 & = b_0d_1\bar{c_0} + a_0d_1 & \rightarrow & f_4{+}r_0 = [m'_4] & \\
f_5 = b_0c_1d_1 + a_1c_1 + a_1d_1 & = c_1(b_0d_1 + a_1) + a_1d_1 & \rightarrow & f_5{+}r_1 = [m'_5] & \\
f_6 = b_1c_0d_1 + a_1c_0 & & \rightarrow & f_6{+}r_2 = [m'_6] & m_1 = m'_4 + m'_5 \\
f_7 = b_1c_1d_1 + b_1d_1 + c_1 & = c_1 \vee b_1d_1 & \rightarrow & f_7{+}r_0{+}r_1{+}r_2 = [m'_7] & +m'_6 + m'_7 \\
 & & & & \\
g_0 = a_0c_0d_0 + a_0d_0 + c_0 & = c_0 \vee a_0d_0 & \rightarrow & g_0{+}r_3 = [n'_0] & \\
g_1 = a_0c_1d_0 + b_0d_0 + d_0 & = d_0(a_0c_1 + b_0 + 1) & \rightarrow & g_1{+}r_4 = [n'_1] & \\
g_2 = a_1c_0d_0 + b_1d_0 & = d_0(a_1c_0 + b_1) & \rightarrow & g_2{+}r_5 = [n'_2] & n_0 = n'_0 + n'_1 \\
g_3 = a_1c_1d_0 + a_1d_0 & = a_1d_0\bar{c_1} & \rightarrow & g_3{+}r_3{+}r_4{+}r_5 = [n'_3] & +n'_2 + n'_3 \\
\hline
g_4 = a_0c_0d_1 + a_0d_1 & = a_0d_1\bar{c_0} & \rightarrow & g_4{+}r_3 = [n'_4] & \\
g_5 = a_0c_1d_1 + b_0d_1 & = d_1(a_0c_1 + b_0) & \rightarrow & g_5{+}r_4 = [n'_5] & \\
g_6 = a_1c_0d_1 + b_1d_1 + d_1 & = d_1(a_1c_0 + b_1 + 1) & \rightarrow & g_6{+}r_5 = [n'_6] & n_1 = n'_4 + n'_5 \\
g_7 = a_1c_1d_1 + a_1d_1 + c_1 & = c_1 \vee a_1d_1 & \rightarrow & g_7{+}r_3{+}r_4{+}r_5 = [n'_7] & +n'_6 + n'_7 \\
 & & & & \\
h_0 = a_0b_0d_0 + b_0d_0 + a_0 & = a_0 \vee b_0d_0 & \rightarrow & h_0{+}r_6 = [z'_0] & \\
h_1 = a_0b_1d_0 + b_1c_0 + a_0c_0 & & \rightarrow & h_1{+}r_7 = [z'_1] & \\
h_2 = a_1b_0d_0 + a_1c_0 + b_0c_0 & & \rightarrow & h_2{+}r_8 = [z'_2] & z_0 = z'_0 + z'_1 \\
h_3 = a_1b_1d_0 + b_1d_0 & = b_1d_0\bar{a_1} & \rightarrow & h_3{+}r_6{+}r_7{+}r_8 = [z'_3] & +z'_2 + z'_3 \\
\hline
h_4 = a_0b_0d_1 + b_0d_1 & = b_0d_1\bar{a_0} & \rightarrow & h_4{+}r_6 = [z'_4] & \\
h_5 = a_0b_1d_1 + a_0c_1 + b_1c_1 & & \rightarrow & h_5{+}r_7 = [z'_5] & \\
h_6 = a_1b_0d_1 + b_0c_1 + a_1c_1 & & \rightarrow & h_6{+}r_8 = [z'_6] & z_1 = z'_4 + z'_5 \\
h_7 = a_1b_1d_1 + b_1d_1 + a_1 & = a_1 \vee b_1d_1 & \rightarrow & h_7{+}r_6{+}r_7{+}r_8 = [z'_7] & +z'_6 + z'_7 \\
 & & & & \\
l_0 = a_0b_0c_0 + b_0c_0 + a_0 & = a_0 \vee b_0c_0 & \rightarrow & l_0{+}r_9 = [t'_0] & \\
l_1 = a_0b_1c_0 + b_1d_1 + b_1 & = b_1(d_1 + a_0c_0 + 1) & \rightarrow & l_1{+}r_{10} = [t'_1] & \\
l_2 = a_1b_0c_0 + b_0d_1 & & \rightarrow & l_2{+}r_{11} = [t'_2] & t_0 = t'_0 + t'_1 \\
l_3 = a_1b_1c_0 + b_1c_0 & = b_1c_0\bar{a_1} & \rightarrow & l_3{+}r_9{+}r_{10}{+}r_{11} = [t'_3] & +t'_2 + t'_3 \\
\hline
l_4 = a_0b_0c_1 + b_0c_1 & = b_0c_1\bar{a_0} & \rightarrow & l_4{+}r_9 = [t'_4] & \\
l_5 = a_0b_1c_1 + b_1d_0 & & \rightarrow & l_5{+}r_{10} = [t'_5] & \\
l_6 = a_1b_0c_1 + b_0d_0 + b_0 & = b_0(d_0 + a_1c_1 + 1) & \rightarrow & l_6{+}r_{11} = [t'_6] & t_1 = t'_4 + t'_5 \\
l_7 = a_1b_1c_1 + b_1c_1 + a_1 & = a_1 \vee b_1c_1 & \rightarrow & l_7{+}r_9{+}r_{10}{+}r_{11} = [t'_7] & +t'_6 + t'_7
\end{array}
\tag{8}
$$

3.5 Application to $\mathbb{F}_2^8$ Inversion

The AES S-Box decomposition adopted in this work is shown in Figure 1. It consists of input and output linear mappings, with the central part of the diagram focusing on its core component: the decomposition of the $\mathbb{F}_2^8$ inversion.

As in [SM21], the $\mathbb{F}_2^4$ multiplication, $\mathbb{F}_2^4$ squarer-scaler, and their XOR is integrated into one module, which we refer to square-scale-multiplier.

More specifically, it holds that $u = x \times y \oplus \mathrm{SqSc}(x \oplus y)$, $v = u^{-1}$, $q = x \times v$, and $p = y \times v$ in Figure 1 where $\times$, SqSc and $^{-1}$ denote for the multiplication, squarer-scaler, and inversion in $\mathbb{F}_2^4$. x and p are the 4-bit least significant bits (LSB) of the $\mathbb{F}_2^8$ inversion input and output.

The straightforward method to apply our method on $\mathbb{F}_2^8$ inversion would be find a low-latency TI for each submodule in Figure 1. However, since the randomness used in our method cancels out in the output shares, they have no affects in the distribution of output shares. As a result, the straightforward solution will face the same non-uniform issue as in [SM21].

Uniformity issue in [SM21]. In [SM21], the authors provided a method to solve the uniformity issue. However, we identify that their method is not correct.

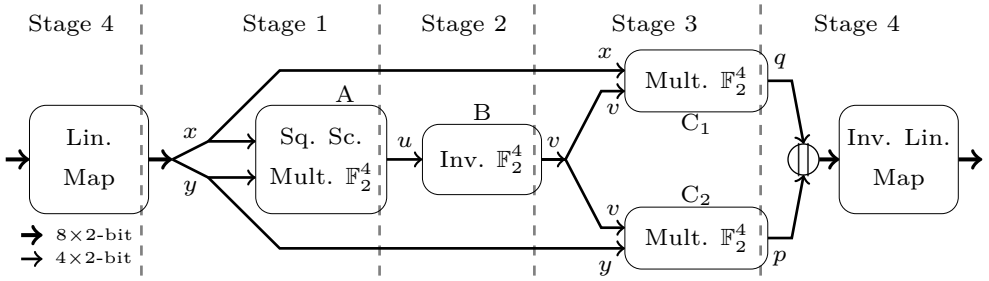


Figure 1: Masked AES S-Box with four stages

Their method is to re-schedule the output linear mapping and the MixColumns operations over four states bytes A, B, C, and D, which are the results of the $\mathbb{F}_2^8$ inversion for four state bytes that are involved in the MixColumns. The fist step is to apply a matrix transformation as shown in Equation 9 to A, B, C, and D. The authors of [SM21] claimed that each byte of the results, i.e. X, Y, Z, and T, is uniform. We now illustrate that this claim does not hold.

$$\begin{bmatrix} 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} A \\ B \\ C \\ D \end{bmatrix} = \begin{bmatrix} X \\ Y \\ Z \\ T \end{bmatrix} \tag{9}$$

We used a modified version of Prover [ZCF25] to analyze the uniformity of the $\mathbb{F}_2^8$ inversion implementation without fresh randomness in [SM21]. We identified that the linear combination of the first bit A_0^0 and the fifth bit A_0^4 in the first output share is not balanced, i.e., $\Pr[A_0^0 + A_0^4 = 0] - \frac{1}{2} = p \simeq -0.0021 \neq 0$. Since the inputs of A, B, C, and D are independent in the first round, the same holds for the output sharing of B, C, and D

Therefore the linear combination of the first bit ($X_0^0 = A_0^0 + C_0^0 + D_0^0$) and fifth bit ($X_0^4 = A_0^4 + C_0^4 + D_0^4$) of $X = A + C + D$ will also be not balanced: from the pilling-up lemma [Mat94], $\Pr[X_0^0 + X_0^4 = (A_0^0 + A_0^4) + (C_0^0 + C_0^4) + (D_0^0 + D_0^4)] - \frac{1}{2} = 4p^3 \neq 0$.

From Lemma 5, we can know that X is not uniform. The same holds for Y, Z, and T.

Our solution for uniformity issue. In fact, the TIs for $p = x \cdot v$ and $q = y \cdot v$ require that the sharing of x and v be jointly uniform and the sharing of y and v be jointly uniform, respectively. However, this is out of the consideration of our method since our method only considers a TI that is locally secure without composable guarantees.

Furthermore, the joint uniform sharing of p and q requires that the sharing of the twelve bits x, y and v (or x, y and u) should be jointly uniform (*12-bit uniformity*). Since the original sharing of x and y uses 8 random bit (*and only provides only 8-bit uniformity*), we believe that it is necessary to refresh the intermediate sharing of u or v with 4 additional fresh random bits (*providing the extra 4-bit uniformity*) for the sharing of x, y and u (or v) to be jointly uniform. As a result, we use four random bits (r_1 in Equation 10) that do not cancel out in the masking of square-scale-multiplier to refresh its outputs such that the sharing of its inputs and outputs becomes jointly uniform. **Masking square-scale-multiplier.** We use the scheme shown in Equation 10 to mask the square-scale-multiplier in our construction for $\mathbb{F}_2^8$ inverter. In fact, this masking scheme fulfills 1-SNI, which can be verified by maskVerif [BBC+19]. It is enough for the scheme to be 1-SNI when $u_1' = f_1 + r_1$ and $u_2' = f_2 + r_1$, i.e., without adding r_0 to u_1' and u_2'. However, we still add the additional mask r_0 for u_1' and u_2' to ensure that r_0 cancels out in the expression of u_0 and u_1. Therefore, the subsequent operations on u_0 and u_1 is independent of r_0

and r_0 can be from an unrelated state bits instead of fresh randomness to save online randomness, which will be detailed in Section 4.

$$\begin{array}{llll} f_0 = x_0y_0 + \mathrm{SqSc}(x_0 + y_0) & \rightarrow & f_0{+}r_0 = [u'_0] & \\ f_1 = x_0y_1 & \rightarrow & f_1{+}r_0 + r_1 = [u'_1] & u'_0 + u'_1 = u_0 \\ \hline f_2 = x_1y_0 & \rightarrow & f_2{+}r_0 + r_1 = [u'_2] & \\ f_3 = x_1y_1 + \mathrm{SqSc}(x_1 + y_1) & \rightarrow & f_3{+}r_0 = [u'_3] & u'_2 + u'_3 = u_1 \end{array} \tag{10}$$

In the meanwhile, we have also tested the security of the direct masking scheme of $\mathbb{F}_2^4$ square-scale-multiplier without any fresh random bit (see in Equation 10 without the gray parts) with Prover. It is reported it also achieves a uniform output sharing. As a result, when considering masking the square-scale-multiplier alone, even in the absence of the random bits r_1 in Equation 10, the remaining circuit maintains a uniform sharing without requiring any correction terms.

Masking $\mathbb{F}_2^4$ inverter. We use the TI of $\mathbb{F}_2^4$ inversion detailed in Subsubsection 3.4.2 for our construction of $\mathbb{F}_2^8$ inversion, which requires 12 random masks. Since the input shares of the $\mathbb{F}_2^4$ inverter (i.e., the output of the square-scale-multiplier) have been refreshed by r_1 in Equation 10, the output shares (and its glitch-extended probing set) remains independent of the shares of x and y. Therefore, for any $i, j, k \in \{0, 1\}$, $\rho(x_i) \cup \rho(v_j) \cup \rho(y_k) = \{x_i, y_k\} \cup \rho(v_j)$ remains jointly uniform, which enables x_i, v_j, y_k to be safely used as inputs to the final two multiplications without storing them in registers.

Masking the final two multiplications. For the final two multiplications, although we can directly take the designs in [YCW+24], we indeed find a more efficient solution, inspired by the idea used in TI of AND3 detailed in Subsubsection 3.4.1. We observe that if a and b are shared independently, then $(c_0, c_1) = (a_0b + b_0, a_1b + b_0)$ makes a uniform sharing of $c = ab$ even when a, b, a_0, a_1, b_0, b_1 are all in $\mathbb{F}_2^4$. Therefore, we use the masking scheme presented in Equation 11. Interestingly, the scheme without the gray parts is of a similar form of the masked two-input AND gate introduced in [SM21, §2.4] and presented in [WFW+25, Equation 2].

$$\begin{array}{lll} f_0 = a_0b_0 + b_0 & \rightarrow f_0{+}r_0 = [c'_0] & \\ f_1 = a_0b_1 & \rightarrow f_1{+}r_0 = [c'_1] & c'_0 + c'_1 = c_0 \\ \hline f_2 = a_1b_0 + b_0 & \rightarrow f_2{+}r_0 = [c'_2] & \\ f_3 = a_1b_1 & \rightarrow f_3{+}r_0 = [c'_3] & c'_2 + c'_3 = c_1 \end{array} \tag{11}$$

However, it should be noted that x and y should not be placed on the port a in Equation 11 simultaneously in the TI of $q = x \times v$ and $p = y \times v$. Otherwise, both the output p and q is masked by v_0, which will make the sharing of p and q to be not jointly uniform. In our implementation, both x and y are placed on the port b. As a result, $\rho(q_0) = \{v_0x_0 + x_0{+}r_0, v_0x_1{+}r_0\}$, $\rho(q_1) = \{v_1x_0 + x_0{+}r_0, v_1x_1{+}r_0\}$, $\rho(p_0) = \{v_0y_0 + y_0{+}r_1, v_0y_1{+}r_1\}$, and $\rho(p_1) = \{v_1y_0 + x_0{+}r_1, v_1y_1{+}r_1\}$. Since $((p_0, q_0), (p_1, q_1))$ makes a uniform sharing of (p, q), we have that for any $i, j \in \{0, 1\}$, $\rho(p_i) \cup \rho(q_j)$ follow a jointly uniform distribution thanks to Proposition 4.

Formal Verification of $\mathbb{F}_2^8$ Inverter. There are many formal verification tools to verify the security of masked hardware designs. However, only the results of SILVER [KSM20] and Prover [ZCF25] are accurate while the other tools such as maskVerif [BBC+19] and CocoAlma [GHP+21] suffer from false positive issues [ZCF25]. Since our designs are similar to the designs in [YCW+24] and [SM21], whose security can not be verified with maskVerif and CocoAlma, we choose Prover, which is a improved tool based on SILVER, to verify the security of the whole $\mathbb{F}_2^8$ inverter. We also use maskVerif to verify the security of the square-scale-multiplier since it fulfills a stronger security notion – SNI.

For the masked $\mathbb{F}_2^8$ inverter, it took about 22 days for Prover to verify its standard and glitch-extended probing security, as well as uniformity.

To verify our design for $\mathbb{F}_2^8$ inverter more efficiently, we have split our design of $\mathbb{F}_2^8$ inverter into two smaller parts. The first part is the square-scale-multiplier in the first stage while the second part is the rest of the $\mathbb{F}_2^8$ inverter. We utilize maskVerif to verify that the first part of our design is 1-SNI. Therefore, it can be ensured that the input sharing of the second part (the input and output shares of the square-scale-multiplier) is jointly uniform. Then we use Prover to verify the security of the second part. As the second part is smaller compared to the whole $\mathbb{F}_2^8$ inverter, Prover can verify its security in several minutes.

4 Low-Latency First-Order AES Implementations

In this section, we present two compact low-latency first-order Threshold Implementations for AES-128 encryption with low randomness. The AES-128 encryption can be partitioned into three parts, the initial round key addition, 9 normal rounds, and a final round [AES01]. The normal rounds have four operations, namely SubBytes, ShiftRows (SR for short), MixColumns (MC for short), and AddRoundKey (AK for short). Compared to the normal rounds, the final round does not include the MixColumns operation.

In the following subsections, we first introduce techniques to reduce latency and then techniques to reduce randomness.

4.1 Reducing Register Stages

There are four register stages in our design for S-Box. If we use a round-based architecture for the masked AES implementations, with an additional register stage to store the state of round results, 5 cycles are needed to complete the computation of one round. Therefore, the total latency of AES encryption will be 50 cycles.

In the next, we will introduce a two-step latency reduction for our masked AES, each saving one cycle per round. This yields two designs: a 40-cycle variant and a 31-cycle variant, corresponding to the optimizations detailed in the following sections.

4.1.1 The First Step: A 40-cycle Design

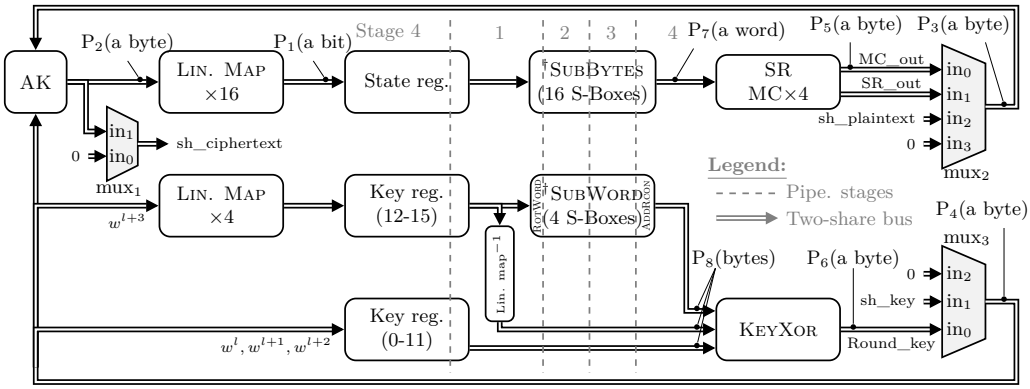


Figure 2: Round-based AES architecture with 40-cycle latency. The SubBytes and SubWord operations marked with † excludes the input linear mapping of the S-Box.

We can merge the register stage that stores the round results and the one that stores the results of input mapping of the S-Box into one register stage, saving one cycle per round. We formalized this idea as follows.

Many cryptographic algorithms can be seen as a iterative function, $E = F^{(n)} = F \circ \cdots \circ \cdots \circ F$, which applies the round function F for n times. Typically, the round results are stored in registers[2] . The round function $F = L_2 \circ N \circ L_1$ is often composed of three layers where L_1, L_2 are the linear layers and N is the non-linear layer (N is assumed to have s register stages). If the result of the first linear layer L_1 is not stored in registers, non-completeness issue might occur in the nonlinear layer N (e.g., [ABP+18, DRB+16]). As the a result, a total latency of $n(s+2)$ cycles is needed to complete the computation of E. However, if we implement the round as $F' = L_1 \circ L_2 \circ N$ (and the final round $F'' = L_2 \circ N$[3]), then the cryptographic algorithm can be implemented as $E = F'' \circ F'^{(n-1)} \circ L_1$. When we store the round results of F' or F'' (instead of results of F), which is necessary for security, together with the s register stages in N, the total latency of E becomes $n(s+1)$ cycles, saving n cycles. The trick has been employed by many works [ABP+18, GKM+24, ZCFY25].

The 40-cycle AES design using this trick is depicted in Figure 2. The latency is defined as the number of intermediate register stages between when the inputs and outputs are available. If the two-shared plaintext is fed to the AES core at cycle c, and the two-shared ciphertext is available in cycle $c + l$, then the latency is l-cycle. However, some works might use a different definition of latency. They refer the latency in our definition as $l+1$.

In our case, suppose the AES core receives the plaintext shares and key shares in cycle 0, then it will produce the ciphertext shares in cycle 40.

At cycle 0, the AES core receives the plaintext shares sh_plaintext and the key shares sh_key. Rather than loading sh_plaintext and sh_key directly into the state and key registers, the core first performs all linear operations up to (but excluding) the nonlinear $\mathbb{F}_2^8$ inversion, and only then stores the intermediate results in registers. These linear operations comprise the initial round-key addition and the input linear mappings of SubBytes and SubWord.

After the initial round key addition, 10 cipher rounds are executed in the next cycles. Each round starts with the input linear mapping, and then the SubBytes without the input linear mapping, and next the ShiftRows and MixColumns (optional), and ends with AddRoundKey. A multiplexer (mux_2 in Figure 2) is used to select the signals that are routed to the AK module. In cycle 0, the plaintext shares sh_plaintext is selected. In cycle $4i$ ($1 \le i \le 9$), the output of MixColumns is selected. In cycle 40, the output of ShiftRows is selected. In the other cycles, constant 0 is selected. Feeding all S-Box instances with the constant 0 in the other cycles can circumvent transitional leakage [HB25].

Following the notations in [AES01], we use w^l, w^{l+1}, w^{l+2} and w^{l+3} to denote the four round key words used in the $\frac{l}{4}$-th round (l ranges over the multiples of 4 between 0 and 36). The round key for the next round is denoted by $w^{l+4}, w^{l+5}, w^{l+6}$ and w^{l+7}. As shown in Equation 12, the next round key can be generated from $w^l, w^{l+1}, w^{l+2}, w^{l+3}$, and $F(w^{l+3})$ where F denotes the composite operation of RotWord, SubWord, and AddRcon.

$$\begin{aligned} w^{l+4} &= w^l \oplus F(w^{l+3}) \\ w^{l+5} &= w^{l+1} \oplus w^{l+4} = w^{l+1} \oplus w^l \oplus F(w^{l+3}) \\ w^{l+6} &= w^{l+2} \oplus w^{l+5} = w^{l+2} \oplus w^{l+1} \oplus w^l \oplus F(w^{l+3}) \\ w^{l+7} &= w^{l+3} \oplus w^{l+6} = w^{l+3} \oplus w^{l+2} \oplus w^{l+1} \oplus w^l \oplus F(w^{l+3}) \end{aligned} \tag{12}$$

The AES core executes the key expansion process through the SubWord and KeyXor blocks, along with the Lin. Map and its inverse[4] blocks in Figure 2. The multiplexer

[2]However, storing intermediate round results is more of an implementation convention than a security requirement. For example, some PINI-based AES designs [CGM+24] store round results, whereas others do not [VDB+25].

[3]F' and F'' can be implemented with the same circuit, since we can use a multiplexer to skip L_1 in the final round.

[4]Since the key registers stores the normal basis of the key word w^{l+3}, we need to convert them back

mux_3 selects the signals to the AK module. In cycle 0, it selects the key shares sh_key. In cycle $4i$ $(1 \leq i \leq 10)$, it selects the generated round key. In the other cycles, it selects constant 0 to avoid the transitional leakage.

At cycle 40, the ciphertext shares are available, the multiplexer mux_1 selects thec ciphertext shares as the output sh_ciphertext. In the other cycles, the AES core set the output sh_ciphertext to zero.

4.1.2 The Second Step: A 31-cycle Design

To further reduce latency, we can defer the compression of the final two multipliers in Figure 1 to the next round. Instead of compressing the results of the 4 component functions of p and q (which are stored in registers) into two shares, and then applying the subsequent operations on the two shares, we directly apply the subsequent linear operations[5] over the four shares (without any intermediate register stages) and store the four shares of the results as round results. This will further reduce the latency of our AES implementation by one cycle per round. However, the chip area for the linear operations is doubled. On the contrary, the chip area for registers is reduced, since the number of pipeline register stages is reduced. Similar trick is employed in [ZSS+21] for the implementations of Keccak.

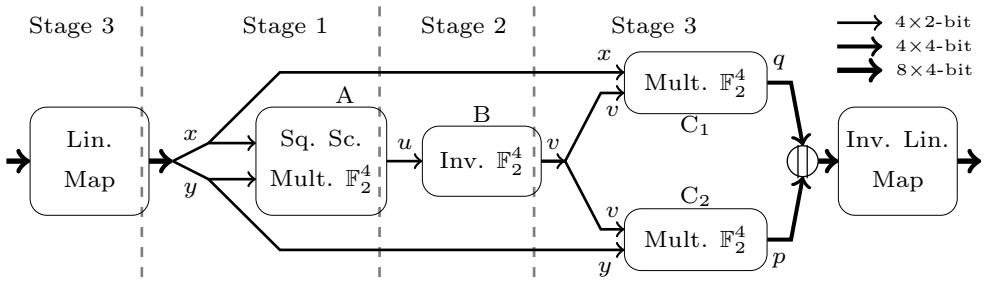


Figure 3: Masked AES S-Box with three stages: The input linear mapping receives four input shares and the four shares of its output are stored in registers. Then the four shares are compressed into two shares of x, y in stage 1. After compression, stage 1 and stage 2 are the same as the four-staged S-Box. The outputs of C_1 and C_2 have four shares which are routed to output linear mapping.

The three-stage S-Box and the 31-cycle AES design are depicted in Figure 3 and Figure 4, respectively. In this design, the inputs and outputs of all the linear operations in this design have four shares. The $\mathbb{F}_2^4$ square-scale-multiplier A and $\mathbb{F}_2^4$ inversion B receive two input shares and output two shares while the final two multiplications C_1 and C_2 receives two shares but output four shares.

For sake of convenience, we associate each share with a domain, which corresponds to the share index of the shared variable. The two shares of inputs and outputs of square-scale-multiplier and $\mathbb{F}_2^4$ inversion, as well as the two shares of the inputs of the multiplier C_1 and C_2, are associated with domain 0 and 1, respectively. For the four output share of C_1 and C_2, the output share computed from v_i in domain i and p_j (or q_j) in domain j is assigned to a composite domain (i, j) where $i, j \in \{0, 1\}$. Subsequent linear operations are performed separately on the four shares within their respective (i, j) domains, with their inputs and outputs remaining associated with the same domain. After all the linear operations are performed, the four shares are stored in registers, then compressed into

into polynomial basis for the computation of w^{l+7} in Equation 12.

[5]Including the remaining linear operations in this round (output mapping of S-Box, ShiftRows, MixColumns, AddRoundKey) and the linear operations of the next round (the input mapping of the S-Box).

two shares and used as two input shares of the $\mathbb{F}_2^8$ inversion. Specifically, the two shares in domain $(i, 0)$ and $(i, 1)$ are compressed into one share in domain i for $i \in \{0, 1\}$.

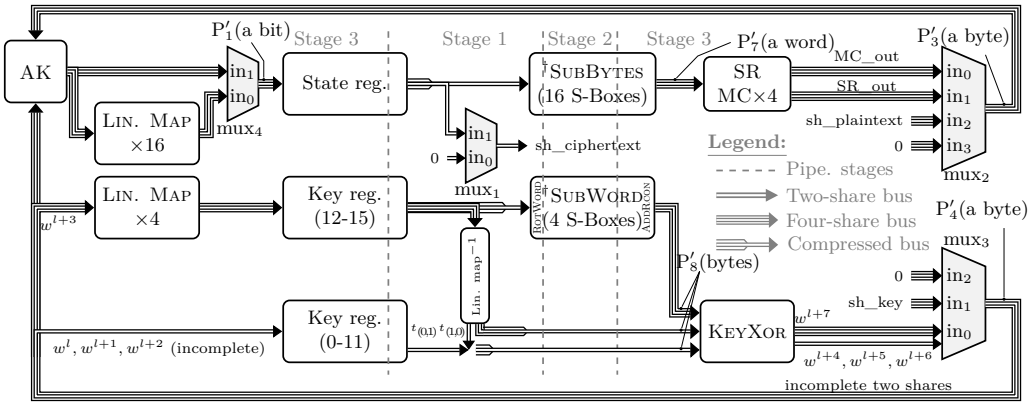


Figure 4: Round-based AES architecture with 31-cycle latency. The SubBytes and SubWord operations marked with † excludes the input linear mapping of the S-Box. The four shares of the data are compressed into two shares through XOR operations when they are carried on compressed bus.

At the start of encryption, AddRoundKey is performed. Since it is a linear operation and requires four input shares while the plaintext and key have only two shares, we need to expand the two-shared sh_plaintext (P_0, P_1) and sh_key (K_0, K_1) into four shares – $P_{(0,0)} = P_0, P_{(0,1)} = 0, P_{(1,0)} = 0$, and $P_{(1,1)} = P_1$; $K_{(0,0)} = K_0, K_{(0,1)} = 0, K_{(1,0)} = 0$, and $K_{(1,1)} = K_1$.

In cycle 30, the four shares of ciphertext (the results of the AK module) are ready. However, they can not be compressed directly in this cycle. The AES core has to wait for a cycle for the compression and finally produces the two shares of the ciphertext. As a result, the multiplexer mux_4 is added to skip linear mapping after the AK operation of the final round.

For key scheduling, it is not necessary for all the key words to store all four shares in registers. We observe that for the key words that are not involved in SubWord, it is enough to only store the two shares in domain $(0, 0)$ and $(1, 1)$ in registers, since the other two shares in domains $(1, 0)$ and $(0, 1)$ are the same as the key word w^{l+3}, which is involved in SubWord. We explain this in the following.

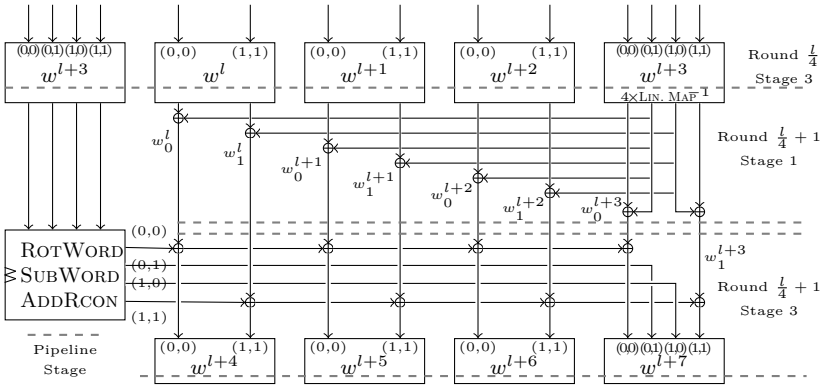


Figure 5: Key Scheduling (the upper left block for w^{l+3} is only repeated for presenting the constructions better)

In stage 1 of the key scheduling, each key word is compressed into two shares, denoted as $w_0^{l+i} = w_{(0,0)}^{l+i} \oplus w_{(0,1)}^{l+i}$ and $w_1^{l+i} = w_{(1,0)}^{l+i} \oplus w_{(1,1)}^{l+i}$ where $0 \leq i \leq 3$. Let $t = F(w^{l+3})$ where F denotes the composite operation of RotWord, SubWord, and AddRcon. We denote the four shares of t produced in stage 3 as $t_{(0,0)}, t_{(0,1)}, t_{(1,0)}, t_{(1,1)}$. The round key for the $\frac{l}{4}+1$-th round can be calculated following Equation 13. We can see that each key word has four separate parts that must be stored in different registers[6], which is indicated by the $[\cdot]$ in Equation 13. However, the four key words have two common shares, i.e., $t_{(0,1)}$ and $t_{(1,0)}$. Therefore, instead of storing 4 copies of $t_{(0,1)}$ and $t_{(1,0)}$ into registers for the four key words, we only need to store only one copy into registers. The implementation for key scheduling is shown in Figure 5. As a side note, we would like to highlight that if we do not compress the four shares of the key words w^l, w^{l+1}, w^{l+2} into two shares in stage 1, they will not share the two common shares with w^{l+3} thus the saving on register area will not be possible.

$$
\begin{aligned}
w^{l+4} &= w^l + F(w^{l+3}) = [w_0^l + t_{(0,0)}] + [w_1^l + t_{(1,1)}] + [t_{(0,1)}] + [t_{(1,0)}] \\
w^{l+5} &= w^{l+1} + w^{l+4} = [w_0^{l+1} + w_0^l + t_{(0,0)}] + [w_1^{l+1} + w_1^l + t_{(1,1)}] + [t_{(0,1)}] + [t_{(1,0)}] \\
w^{l+6} &= [w_0^{l+2} + w_0^{l+1} + w_0^l + t_{(0,0)}] + [w_1^{l+2} + w_1^{l+1} + w_1^l + t_{(1,1)}] + [t_{(0,1)}] + [t_{(1,0)}] \\
w^{l+7} &= [w_0^{l+3} + w_0^{l+2} + w_0^{l+1} + w_0^l + t_{(0,0)}] + [w_1^{l+3} + w_1^{l+2} + w_1^{l+1} + w_1^l + t_{(1,1)}] + [t_{(0,1)}] + [t_{(1,0)}]
\end{aligned}
\tag{13}
$$

4.2 Reducing Randomness by Guards

Our masked S-Box consumes a total of $8 + 12 + 8 = 28$ random bits. However, there are 24 bits that are eliminated in the expression of the output shares. The remaining four bits are used to maintain a uniform sharing of the outputs.

Since the eliminated 24 bits do not affect the distribution of the output shares (and thus will not affect the distributions of results of the subsequent operations over the output shares), we can use unrelated state bytes, i.e., one input share of other S-Boxes, to replaced them. We refer to the unrelated input share from other S-Boxes as *guards*. This shall significantly reduce the randomness overhead in our designs. Similar idea of saving randomness by reusing unrelated state bits has been reported more than a decade ago [BDN+13], it was later formalized to the method *changing of the guards* (COTG) [Dae17]. This method is widely deployed in literature [WM18, Sug18, BDZ20, DSM22, ADN+22].

Our method is different from the changing of the guards technique presented in [Dae17]. The changing of the guards technique solves the uniformity of the sharing, while we use the guards for refreshing to ensure glitch-extended probing security of combinations of output shares from different coordinate functions. It is more similar to the COTG-based concept proposed in [GKM+24].

Super box representation of a round. In cryptanalysis, researchers use the super box representation for two rounds of the AES such that the two rounds have only one nonlinear layer [DR20]. Borrowing the idea, we present that one round of AES can be divided into four smaller super boxes that do not share common inputs. This idea has also been utilized in [GKM+24]. However, they did not provide a formal description for the super boxes, which is covered by us in the next.

Let the round input be $\{s^i | 0 \leq i \leq 15\}$ (16 bytes) and the round key be w^l, w^{l+1}, w^{l+2}, and w^{l+3} (four words). Let $\mathcal{L}_1$ and $\mathcal{L}_2$ be the input linear mapping and output linear mapping of Canright's S-Box. Then we define the word-level operation for $\mathcal{L}_1$ and $\mathcal{L}_2$ - ILMapWord and OLMapWord, which means applying $\mathcal{L}_1$ and $\mathcal{L}_2$ over the four bytes in a word individually, respectively. And we use InvWord to denote applying the $\mathbb{F}_2^8$ inversion over the four bytes in a word. Then the AES round function can be divided into four super boxes which implement the function $w^{l+i}\oplus$ (MixColumns $\circ$ OLMapWord

[6] Otherwise, if any two parts are XORed together before saving them into registers, leakage will exhibit.

$\circ$ InvWord $\circ$ ILMapWord(C^i)) with $i \in \{0, 1, 2, 3\}$. And the inputs to the four super boxes will be $C^0 = (s^0, s^5, s^{10}, s^{15})$, $C^1 = (s^4, s^9, s^{14}, s^3)$, $C^2 = (s^8, s^{13}, s^2, s^7)$, and $C^3 = (s^{12}, s^1, s^6, s^{11})$, respectively. Note that the composition OLMapWord $\circ$ InvWord $\circ$ ILMapWord equals four SubBytes, and ShiftRows is implemented through the byte indexing in $C^i (0 \leq i \leq 3)$.

Guards employment for round function. It is observed that by placing one probe in the circuit depicted in Figure 2 and Figure 4, the adversary have access to the internal state of *either* exactly one S-Box *or* all the four S-Boxes inside a super box. Specifically, if the adversary place a probe at stage 1 or stage 2 of both designs, or stage 3 of the 40-cycle design, the adversary have access to the shared data of only one S-Box. If the adversary place a probe in stage 3 of the 31-cycle design or stage 4 of the 40-cycle design, they have access to internal state of four S-Boxes in the same super boxes due to the diffusion property of the MixColumns.

When the adversary have access to only one S-Box, one input share of the remaining 15 S-Boxes can be used as guards of this S-Box since input sharing of the S-Boxes is jointly uniform[7]. When the adversary have access to one super box, one share of the rest super boxes can be used as guards of this super box. Furthermore, it should be ensured that any two S-Boxes in the same super box should not employ the same guards.

Table 1: Guards employed for each S-Box

S-Box index	Stage 1	Stage 2	Stage 3	S-Box index	Stage 1	Stage 2	Stage 3
0	$s_0^{01}[7:4]$	$s_0^{08}, s_0^{12}[3:0]$	s_0^{04}	8	$s_0^{09}[7:4]$	$s_0^{00}, s_0^{04}[3:0]$	s_0^{12}
5	$s_0^{06}[7:4]$	$s_0^{13}, s_0^{01}[3:0]$	s_0^{09}	13	$s_0^{14}[7:4]$	$s_0^{05}, s_0^{09}[3:0]$	s_0^{01}
10	$s_0^{11}[7:4]$	$s_0^{02}, s_0^{06}[3:0]$	s_0^{14}	2	$s_0^{03}[7:4]$	$s_0^{10}, s_0^{14}[3:0]$	s_0^{06}
15	$s_0^{00}[7:4]$	$s_0^{07}, s_0^{11}[3:0]$	s_0^{03}	7	$s_0^{08}[7:4]$	$s_0^{15}, s_0^{03}[3:0]$	s_0^{11}
4	$s_0^{05}[7:4]$	$s_0^{12}, s_0^{00}[3:0]$	s_0^{08}	12	$s_0^{13}[7:4]$	$s_0^{04}, s_0^{08}[3:0]$	s_0^{00}
9	$s_0^{10}[7:4]$	$s_0^{01}, s_0^{05}[3:0]$	s_0^{13}	1	$s_0^{02}[7:4]$	$s_0^{09}, s_0^{13}[3:0]$	s_0^{05}
14	$s_0^{15}[7:4]$	$s_0^{06}, s_0^{10}[3:0]$	s_0^{02}	6	$s_0^{07}[7:4]$	$s_0^{14}, s_0^{02}[3:0]$	s_0^{10}
3	$s_0^{04}[7:4]$	$s_0^{11}, s_0^{15}[3:0]$	s_0^{07}	11	$s_0^{12}[7:4]$	$s_0^{03}, s_0^{07}[3:0]$	s_0^{15}

On the other hand, two consecutive stages should not employ the same guards. Consider the case when the two output x and y of Equation 4 are multiplied in a next step. If the masked multiplication $x \times y$ use the same guarding random bits r_0 as in the masked multiplication $x = ad + bc + bd$, then a probe placed at the result wire of $x_0y_1 + r_0$ will enable the adversary to obtain $\{f_0 + r_0, f_1 + r_0, g_2 + r_1, g_3 + r_1, r_0\}$. As a result, the adversary can use r_0 to unmask $f_0 + r_0$ and $f_1 + r_0$ and obtain $\{f_0, f_1\}$. However, the security of the set $\{f_0, f_1\}$ is not covered by the four properties presented in Subsection 3.3.

Finally, we employ the guards for each S-Box as shown in Table 1, which is suitable for both AES designs. The employment for guards is not unique; however, careful consideration must be given during a employment that works for both designs due to the facts that (a) the adversary have access to a super box in stage 3 of the 31-cycle design and (b) they have access to a super box in stage 4 of 40-cycle design. Note that when the adversary place a probe at stage i, it has access to the shared data that is protected by the guards employed in the previous stage. Therefore the *guards* employed for a S-Box in the previous stage of stage 3 and stage 4 (namely stage 2 and stage 3) must come from the other three super boxes rather than S-Boxes within the same super box. For stage 1, we use one share of a neighboring S-Box as guards.

[7]This ensures that the employed guards satisfied the assumption that $\mathcal{R}^{(j)}$ (guards) are uniform bits independent of all $\mathcal{U}^{(i)}$ (unmasked observations of the S-Box protected by guards) and $\mathcal{V}$ (secret inputs of the protected S-Box) in Proposition 2.

Guards employment for key scheduling. The guards we employed for the S-Boxes in key scheduling is shown in Table 2. After SubWord and AddRcon, all XOR operations shown in Equation 12 are performed sharewisely in the last stage of a round. As a result, the guards used in the previous stage should not be the key bytes that participate in same key word XORing as the protected byte, which are shown in the last column of Table 2. For example, the first output byte of the SubWord is XORed with k^0, k^4, k^8, k^{12}. Therefore, the shares of k^0, k^4, k^8, k^{12} should not be used as guards of the third stage of S-Box of the 13-th key byte, which is the first input byte to SubWord.

Table 2: Guards employed for Key Scheduling

S-Box		Stage			Related Bytes
index	input	1	2	3	
0	k^{13}	$k_0^6[3:0]$	$k_0^6[7:4], k_0^7$	k_0^5	k^0, k^4, k^8, k^{12}
1	k^{14}	$k_0^7[3:0]$	$k_0^7[7:4], k_0^8$	k_0^6	k^1, k^5, k^9, k^{13}
2	k^{15}	$k_0^8[3:0]$	$k_0^8[7:4], k_0^9$	k_0^7	k^2, k^6, k^{10}, k^{14}
3	k^{12}	$k_0^5[3:0]$	$k_0^5[7:4], k_0^6$	k_0^4	k^3, k^7, k^{11}, k^{15}

Remark 5. Although operations are performed over the guards in another S-Box (or super box), this will not break the statistical independence of the observations ($\mathcal{W}$ in Proposition 2) and secret inputs ($\mathcal{V}$ in Proposition 2). This is because the adversary have only one probe in a first-order attack. Once they place the probe in one S-Box (or super box), they do not have a second probe to place at the S-Box (or super box) where the guards come from to gain information about the guards.

4.3 Round Security Analysis of our Design

In this section, we provide the security analysis of one round in AES. First, we introduce a stronger but simplified probing ability of the adversary in our analysis. Then we provide security proofs of our designs based on this probing ability.

4.3.1 Bit-level, Byte-level, and Word-level Probing

Under the glitch-extended probing model, probing a wire enables the adversary to gain all the registered signals that contribute to the computation of the value this wire carries. This requires us to recursively track the dependent input bits of an output bit for a operation *in combinational logic* until the input bits are stored in registers. The analysis shall be much more difficult if we use a precise dependency of the bits. Therefore we adopt some simplifications to ease the analysis. Although the simplifications make the analysis imprecise, it is correct and useful when giving the security proofs of our designs.

We find that in AES, the *linear* operations can be categorized into three levels: (i) bit-level, (ii) byte-level, and (iii) word-level (a word consists of four bytes). The simplifications of the dependencies for each level are listed below.

- Bit-level. For the AK operation, one output bit only depends on two bits of the inputs, i.e., one bit of plain state and one bit of round key. For ShiftRows operation, one output bit only depends on the same bit, since the ShiftRows operation only changes the index of bits.

- Byte-level. We assume if the adversary have access to one or several output bits of a byte-level operation *in combinational logic*, they will gain the access to all bits of the input byte of this operation by *glitches* . Byte-level operations include the input/output linear mapping of the Canright's S-Box.

- Word-level. We assume if the adversary have access to one or several output bits of a word-level operation *in combinational logic*, they will gain the access to all bits of the input word of this operation by *glitches*. Word-level operations include MixColumns.

4.3.2 Round Security of Our 40-cycle Design

We conclude the security of 40-cycle design in two cases.

Security of the first three stages. The adversary can either probe the inverse of the input linear mapping in the first stage that is used for converting the bytes in w^{l+3} back into polynomial basis or the first three stages of only one $\mathbb{F}_2^8$ inverter. The inverse of input linear mapping is a linear operation and thus trivially secure when it is implemented in a sharewise manner. The security of the first three stages of the $\mathbb{F}_2^8$ inverter is verified by formal verification tools, as shown in Subsection 3.5. The only differences is that we used one of the shares from the other S-Boxes (or super box) to replace fresh randomness generated from a PRNG. Since the input sharing of the round state are jointly uniform, the *guards* from the other S-Boxes can also play the role of *fresh randomness* to the probed $\mathbb{F}_2^8$ inverter.

Security of stage 4. Under the probing ability assumptions, if the adversary place a probe P_1 at one input bit of the state registers[8] in Figure 2, then this probe propagates to the position of P_2, where the adversary have access to a byte. When this byte propagates backwards, the adversary have access to one output byte of the multiplexers mux_2 and mux_3, i.e., the probing positions P_3 and P_4. In the middle rounds of AES[9], P_3 propagates to an output byte of MixColumns (P_5), enabling the adversary to gain access to one share of the output bytes of four S-Boxes (P_7). In the meanwhile, P_4 propagates to a byte of the round key P_6 and then at most one byte of $F(w^{l+3}), w^l, w^{l+1}, w^{l+2}, w^{l+3}$, and w^{l+3} (cf. Equation 12), which is marked as P_8.

Since input sharing of the data stored in state registers and key registers are jointly uniform, it suffices to conclude the security of stage 4 by proving that the observations given by P_7 and P_8 are jointly uniform, respectively.

Table 3: Glitch-extended probing set of a byte in w^{l+7} with b^0 the least significant byte (refer to Subsection 3.5 for definitions of x, y, v, p, and q). $S(k^i)$ denote the S-Box that takes shares of k^i as inputs.

b	$\rho_i^{\oplus}(b)$	$\rho_i^{S}(b)$	$\rho_{(i,j)}^{S}(b)$
b^0	$\{k_i^0, k_i^4, k_i^8, k_i^{12}\}$	$\rho(p_i, q_i)$ in $S(k^{13})$	$\rho(v_i) \cup \{x_j, y_j\}$ in $S(k^{13})$
b^1	$\{k_i^1, k_i^5, k_i^9, k_i^{13}\}$	$\rho(p_i, q_i)$ in $S(k^{14})$	$\rho(v_i) \cup \{x_j, y_j\}$ in $S(k^{14})$
b^2	$\{k_i^2, k_i^6, k_i^{10}, k_i^{14}\}$	$\rho(p_i, q_i)$ in $S(k^{15})$	$\rho(v_i) \cup \{x_j, y_j\}$ in $S(k^{15})$
b^3	$\{k_i^3, k_i^7, k_i^{11}, k_i^{15}\}$	$\rho(p_i, q_i)$ in $S(k^{12})$	$\rho(v_i) \cup \{x_j, y_j\}$ in $S(k^{12})$

Observations given by P_7 follow a jointly uniform distribution. If the domain of the probed bit by P_1 is i ($i \in \{0, 1\}$), then by probe P_7, the adversary have access to $\rho(p_i, q_i) = \rho(p_i) \cup \rho(q_i)$ in the four S-Boxes. Since each $\rho(p_i, q_i)$ are jointly uniform (see in Subsection 3.5) and the four S-Boxes are independently shared, the union of the observations given by the four S-Boxes are jointly uniform. *Observations given by P_8 follow a*

[8]The adversary can also place the probe at the input wire of the key registers. However, the probing set obtained by such probes would be a subset of the probing set of a probe placed at a corresponding state register under our simplified probing ability for linear operations. Therefore, it suffices to consider placing the probe at the input wire of the state registers only.

[9]The probes in the first round and the last round are subsets of probes in the middle rounds (in the sense that the public signals like round constants are excluded from the probes) and thus less powerful. Therefore, it suffices to cover the case of middle rounds only.

jointly uniform distribution. By Equation 12, we observe that any glitch-extended probing set of one share of a byte in $w^{l+4}, w^{l+5}, w^{l+6}$ is a subset of glitch-extended probing set of the share of the byte in the same position in w^{l+7} associated with the same domain. In other words, for any $j \in \{4,5,6\}$, any domain $i \in \{0,1\}$ (or $i \in \{(0,0),(0,1),(1,0),(1,1)\}$), and the pair of two bytes (b, b') taken from w^{l+j} and w^{l+7} at the same position within the word, we have $\rho(b_i) \subseteq \rho(b'_i)$. Therefore it suffices to only consider the glitch-extended probing set of the shares of each byte in w^{l+7} – if observation set given by each byte in w^{l+7} is jointly uniform, then any subsets of it (observations given by the other bytes) will be jointly uniform.

We list the partition of the glitch-extended probing of each byte in w^{l+7} in Table 3. The observations given by the key word XORing are given in column $\rho_i^{\oplus}(b)$. The observations given by the S-Box in SubWord are listed in columns $\rho_i^{S}(b)$ (for our 40-cycle design) and $\rho_{(i,j)}^{S}(b)$ (for our 31-cycle design). If a byte b in domain i is probed, then $\rho(b)$ equals $\rho_i^{\oplus}(b) \cup \rho_i^{S}(b)$ in our 40-cycle design. From Table 3, we can conclude that for all the four bytes, $\rho(b)$ follows a jointly uniform distribution since variables in both $\rho^{\oplus}(b)$ and $\rho^{S}(b)$ follow a individual jointly uniform distribution and they are statistically independent of each other.

Remark 6. The RotWord operation is critical to the security of the AES key expansion. If RotWord is not a part of key expansion and thus not applied before feeding w^{l+3} to SubWord, then $\rho_i^{S}(b^0)$ will be $\rho(p_i, q_i)$ for the S-Box with input k^{12}, along with k^{12} in $\rho_i^{\oplus}(b^0)$, the observations $\rho(b^0) = \rho_i^{S}(b^0) \cup \rho_i^{\oplus}(b^0)$ will leak information about k^{12}.

4.3.3 Round Security of Our 31-cycle Design

The security analysis of our 31-cycle design is similar to the analysis of 40-cycle design. We only provide the differences here.

Placing a probe P'_1 at the input bit of the state registers gives P'_7 and P'_8, as shown in Figure 4. If the domain of the probed bit by P'_1 is (i,j) $(i,j \in \{0,1\})$, then by probe P'_7, the adversary have access to $\rho(v_i) \cup \rho(x_j) \cup \rho(y_j) = \rho(v_i) \cup \{x_j, y_j\}$ (refer to Subsection 3.5 for definitions of x, y, v) in the four S-Boxes, which is shown to be jointly uniform in Subsection 3.5. Therefore, the observations in the four independently shared S-Boxes given by P'_7 follow a uniform distribution.

As for the probe P'_8, suppose the domain of the probed byte b is (i,j). Then if $i \neq j$, $\rho(b) = \rho_{(i,j)}^{S}(b)$ otherwise $\rho(b) = \rho_i^{\oplus}(b) \cup \rho_{(i,j)}^{S}(b)$ (cf. Figure 5 or Equation 13). In both cases, we can conclude that for any of the four bytes in w^{l+7}, $\rho(b)$ follows a jointly uniform distribution.

Compared to stage 1 of the 40-cycle design, there is an additional compression before the inputs are fed to the S-Boxes in our 31-cycle design, as shown by the compressed bus in stage 1 of Figure 4. Therefore the adversary can access the uncompressed data when placing a probe in stage 1. Still, the uncompressed data follow a jointly uniform distribution. This is because the uncompressed data $\rho(p_i, q_i)$ of stage 3 in the 40-cycle design are jointly uniform. The 31-cycle design just skip the register storage of these data and applying linear transformations over the two uncompressed component of p_i and q_i individually. Since all the linear transformations have a full rank and thus are bijective, the results are still jointly uniform and can be safely compressed. Therefore the compression in stage 1 will not cause security issues.

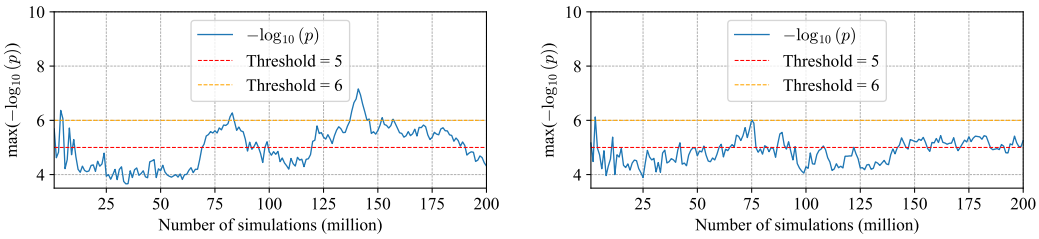
5 Security Evaluations and Comparison

We implemented our AES designs based on the source code provided in Compress, in which the fresh random bits are provided by a Trivium PRNG. The code of our designs and the evaluation scripts are made publicly available at Low-Latency-First-Order-AES.

In this section, we present the evaluation results of our designs and compare our constructions for AES and its S-Box with state-of-the-art implementations.

5.1 Security Evaluations

We evaluated the security of our designs using both PROLEAD and TVLA. **Simulation-Based Leakage Detection using PROLEAD.** PROLEAD [MM22] is a simulation-based leakage assessment tool that evaluates the statistical independence of probing sets under robust probing models. It computes p-values using a fixed-vs-random G-test over a specified number of simulations, and outputs the maximum leakage observed over all probing sets, typically reported as $-\log_{10}(p)$. The authors suggesting a typical threshold of $p = 10^{-5}$. However, due to the large number of probing sets in our case (1.01×10^5 and 1.31×10^5 probing sets in the 31-cycle design and 40-cycle design, respectively), there is a significant probability (approximately 64% and 73%) of encountering false positives [MM22]. Therefore, we adjusted the threshold to 10^{-6}, corresponding to a false positive probability of approximately 10% and 12%.



(a) Results for 31-cycle AES-128

(b) Results for 40-cycle AES-128

Figure 6: PROLEAD results with 200 million simulations.

PROLEAD computes the required total number of traces by solving a non-central χ^2 power equation for the worst (largest) probing set. In *compact mode*, a probing set with b bits yields $K = b + 1$ Hamming-weight classes, and with two groups (fixed vs. random, $G = 2$) the degrees of freedom are

$$\mathrm{df} = (G - 1)(K - 1) = b. \tag{14}$$

Given a per-test significance α, effect size w (Cohen's w/Cramér's V), and Type-II error probability β, the required traces N (across both groups) are obtained from

$$c = \chi^2_{1-\alpha,\mathrm{df}}, \qquad F_{\chi^2_{\mathrm{df}}(\lambda)}(c) = \beta, \qquad N = \left\lceil \frac{\lambda}{w^2} \right\rceil, \tag{15}$$

where $F_{\chi^2_{\mathrm{df}}(\lambda)}(\cdot)$ is the CDF of the non-central χ^2 distribution with non-centrality λ.

In our designs, the largest probing sets contain $b = 414$ (31-cycle) and $b = 254$ (40-cycle) bits, hence $\mathrm{df} = 414$ and $\mathrm{df} = 254$, respectively. Choosing $\alpha = \beta = 10^{-6}$ and $w = 0.1$, the resulting required number of total traces N are 35872 and 29867, i.e., approximately $N/2$ traces per group. Because these N are computed for the maximum probing-set size, they are conservative for all smaller sets; if some Hamming weights are unreachable (fewer than b+1 classes), the effective df decreases and so does the required N.

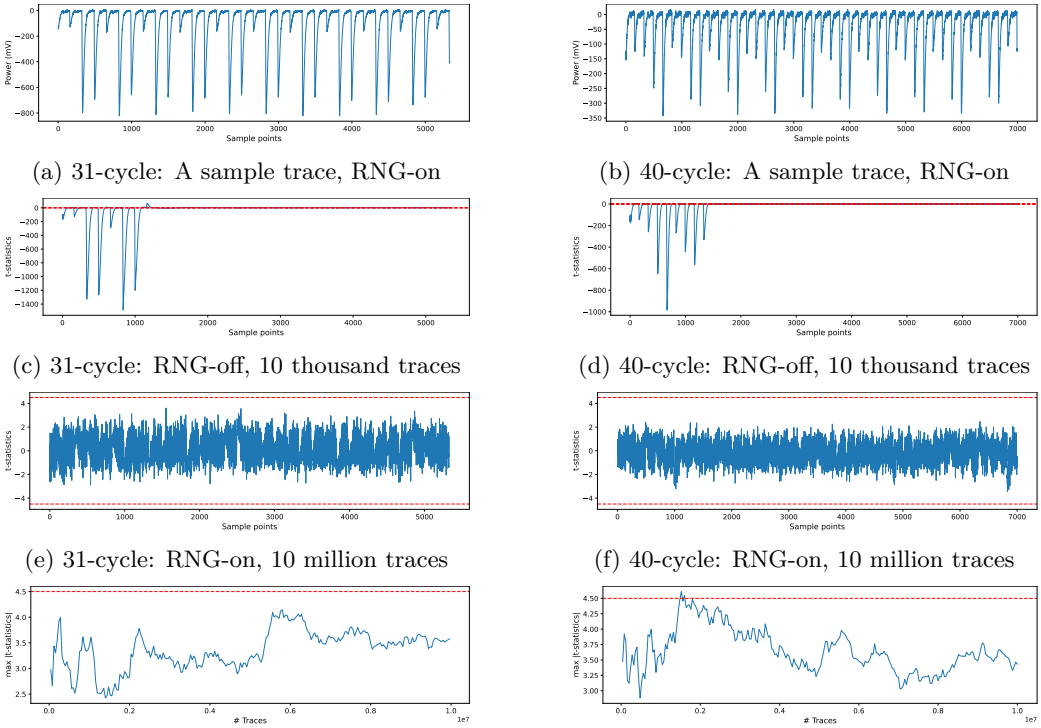
We evaluated our two full AES designs using PROLEAD in *compact mode*, simulating the entire encryption operation across 31 and 40 clock cycles respectively. For each design, we performed 200 million simulations, which is significantly more than the required number of traces, under the $(1, 1, 0)$-robust probing model [FGM+18], which consider the physical defaults including both glitches and transitions.

The results show that in both cases, most $-\log_{10}(p)$ values remain below 6, and no significant upward trend was observed. This behavior is illustrated in Figure 6, and aligns with the interpretation suggested by Müller [Mül25], where the decisive indicator of leakage is not isolated threshold violations, but rather consistent growth of the $-\log_{10}(p)$ values as simulation count increases.

Hence, our experiments provide strong evidence that both implementations maintain statistical independence of the secret inputs under the robust probing model, and exhibit no detectable leakage across the entire AES computation even at high simulation counts.

Practical Leakage Assessment using TVLA. We also employed the Test Vector Leakage Assessment [GGJR11] to validate the security of our round-based AES-128 designs.

We performed practical evaluations for our implementations using a first-order t-test on the SAKURA-G evaluation board connected to a PicoScope 3206D at 500 Ms/s sampling rate (2 ns sampling interval). The hardware design operates at a clock frequency of 3 MHz. The complete AES-128 encryption are measured.



(a) 31-cycle: A sample trace, RNG-on

(b) 40-cycle: A sample trace, RNG-on

(c) 31-cycle: RNG-off, 10 thousand traces

(d) 40-cycle: RNG-off, 10 thousand traces

(e) 31-cycle: RNG-on, 10 million traces

(f) 40-cycle: RNG-on, 10 million traces

(g) 31-cycle: RNG-on, max |t-statistics|

(h) 40-cycle: RNG-on, max |t-statistics|

Figure 7: TVLA results of our 31-cycle and 40-cycle masked AES implementations.

The results of TVLA are shown in Figure 7. The two figures in the first row show a sample trace of our designs. The figures in the second row show the t-test results of first-order TVLA when the initial sharing is disabled (but the Trivium PRNG providing online fresh randomness for our AES designs is not disabled) after acquiring 10 thousand traces. We can see that the t-values are significantly larger than 4.5 in the first cycles, indicating leakage. The third row show the results of TVLA when both the initial sharing and the Trivium PRNG are enabled after 10 million traces. The figures in the last row show the maximum t-value of the sample points with increasing number of acquired traces[10]. We

[10]Please refer to Appendix A for the experiments that confirm the peaks above 4.5 in Figure 7h are false positives.

did not observe evidence for first-order leakage with up to 10 million traces, as the t-value for all the sample points are within the threshold ± 4.5.

5.2 Comparison over Round-based First-order AES-128

We followed a similar synthesis flow of [HB25] to synthesize our designs. We use Yosys 0.33 [WGK13] to elaborate, optimize and synthesize our implementations. Then we use ABC [SG07] to technology-map the circuits with NanGate45 as the target library.

The implementation costs of our designs are listed in the last two rows of Table 4. As discussed in Subsection 4.1, reducing registers leads to a smaller register area but a doubling of linear operation area. Since the increase outweighs the savings, the 31-cycle design incurs a 8.3% area overhead over 40-cycle design, while achieving a 22.5% reduction in latency. We have employed the Trivium PRNG suggested by [CMM+24] to provide 80 random bits per cycle for the AES core. The area without or with the PRNG is listed in the fifth and sixth columns of the last two rows of Table 4.

Several state-of-the-art works [CGM+24, ZCFY25] include the area cost of random number generation in their AES designs and do not provide the area footprint excluding the PRNG in their paper. However, the source code of both works are open-sourced. Therefore, we re-synthesized the designs using the method described in their repository to generate the area footprint with or without a PRNG.

In contrast, other works [VDB+25, KM22] exclude the PRNG when providing the area footprint. We therefore estimate the area of PRNG for such designs for comparison over the area including PRNG. Following [CMM+24], we assume an unrolled Trivium PRNG to estimate the area of generating one bit of randomness per cycle. Specifically, we divide the area of a fully unrolled Trivium by the unrolling factor (512), yielding an estimate of 39.4 GE/bit under the NanGate45 library.

The design in [SBHM20] have already employed a PRNG based on Keccak-f and the area for the PRNG is also provided. Therefore we extracted the area footprint from the paper directly. The design in [ADN+22] have adopted a generalized changing of the guards method and thus does not require a PRNG to provide online randomness.

Table 4: Comparison to state-of-the-art low-latency first-order AES

Designs	S-Box			Full AES					
	Area (kGE)	Rand. *bps*†	Lat. (Cycl.)	Area w/o (kGE)		Area w (kGE)		Rand. *bpr*‡	Lat. (Cycl.)
[SBHM20]$^{\nabla,\bullet}$	3.48	36	1	157.5	−66.9% −69.5%	172.3[1]	−66.9% −68.8%	720	10
[VDB+25]$^{\nabla,\circ}$	4.02	46	2	91.8	−43.2% −47.6%	128.0[2]	−55.4% −58.0%	920	20
[ZCFY25]$^{\triangle,\circ}$	1.63	28	4	49.4	5.5% −2.6%	70.4[3]	−18.9% −23.6%	560	41
[ADN+22]$^{\nabla,\circ}$	5.84	40*	3	115.7	−55.0% −58.4%	115.7	−50.6% −53.5%	0	42
[CGM+24]$^{\triangle,\circ}$	1.95	36	4	56.1	−7.1% −14.3%	84.8[3]	−32.7% −36.6%	720	51
[ZCFY25]$^{\triangle,\circ}$	1.67	24	5	48.6	7.2% −1.0%	67.4[3]	−15.3% −20.2%	480	51
[KM22]$^{\nabla,\circ}$	1.85	66	4	44.6	16.8% 7.8%	98.2[2]	−41.9% −45.2%	1360	55
This work$^{\triangle,\circ}$	1.52	4	3	52.1	−−− −7.7%	57.1[3]	−−− −5.8%	80	31
This work$^{\triangle,\circ}$	1.67	4	4	48.1	8.3% −−−	53.8[3]	6.1% −−−	80	40

$^{\nabla}$ Synthesized with Synopsys $^{\triangle}$ Synthesized with Yosys
$^{\circ}$ NanGate45 design kit $^{\bullet}$ GF 28nm design kit
† bits per S-Box ‡ bits per round * reused by all the S-Boxes in all rounds
[1] PRNG based on Keccak-f [2] Estimated by 39.4 GE per random bit [3] Trivium PRNG

Finally, we summarize the costs of state-of-the-art low-latency first-order AES-128 implementations in Table 4. The two percentage values to the right of the area data

indicate the relative area overhead of the design compared to ours (how much area are decreased by our design compared to the other design, for example, -66.9% means our designs are 66.9% smaller than the compared design), computed as $1 - S_1/S'$ (upper value) and $1 - S_2/S'$ (lower value), where S' is the area of the compared design, and S_1, S_2 are the areas of our 31-cycle design (larger but faster) and 40-cycle design (smaller but slower), respectively.

Only two designs achieve lower latency than ours, namely the LMDPL-based implementation [SBHM20] and the TSM-based design [VDB+25]. Although these designs are faster, our implementation achieves significantly better area (excluding PRNG) efficiency: compared to LMDPL, our designs reduce area by 66.9% and 69.5%, and compared to TSM, the reductions are 43.2% and 47.6%, respectively. Moreover, we use significantly less random bits than these designs. Therefore, when the PRNG is included, the area savings remain comparable.

The designs in [ADN+22] and [ZCFY25] have a comparable latency to our 40-cycle design. However, our 40-cycle design occupies significantly less chip area than the design in [ADN+22] even when the area of PRNG is included. In the meanwhile, although the design provided in [ZCFY25] is almost as compact as ours, it consumes significantly more random bits than ours. Therefore, when the area of PRNG is included, our 40-cycle design reduces the area by 23.6% compared to this design.

As a result, we conclude that our designs offer an excellent trade-off between latency, area, and randomness consumption.

6 Conclusion

In this work, we propose a new construction for first-order $d + 1$ Threshold Implementations, utilizing both the property of Boolean functions and randomness to achieve secure combination of the expansion layer of different coordinate functions. This results in $d + 1$ Threshold Implementations with single-cycle latency. Since the randomness used in expansion layer cancels out in the expression of output shares, we can use the shares from unrelated modules (such as the other S-Boxes) as the random masks. Our constructions is applicable to quadratic functions, cubic functions and potentially functions with even higher degree. As a case study, we apply our techniques on the AES S-Box, which only consumes 4 fresh random bits per S-Box. Furthermore, we proposed two low-latency AES implementations with a latency of 31 cycles and 40 cycles respectively based on our S-Box and a modified round-based architecture provided in [CGM+24]. Although the designs in both [SBHM20] and [VDB+25] are faster than ours, they are significantly larger than our 31-cycle design with higher demands on online randomness. Our designs achieves an excellent trade-off in terms of area, latency, and randomness demands.

Future work includes further reducing the randomness demands of our designs or developing more efficient formal verification tools to verify the designs based on our techniques.

Acknowledgment

This work is supported by the National Natural Science Foundation of China (No. 62172395) and the Beijing Natural Science Foundation (No. L234085).
The authors sincerely thank the anonymous reviewers for their valuable comments, which significantly improved the quality of the paper.

References

[ABP+18] Victor Arribas, Begül Bilgin, George Petrides, Svetla Nikova, and Vincent Rijmen. Rhythmic Keccak: SCA security and low latency in HW. *IACR TCHES*, 2018(1):269–290, 2018.

[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.

[AES01] Advanced Encryption Standard (AES). National Institute of Standards and Technology, NIST FIPS PUB 197, U.S. Department of Commerce, November 2001.

[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 Schneider, and Peter Y. A. Ryan, editors, *ESORICS 2019, Part I*, volume 11735 of *LNCS*, pages 300–318. Springer, Cham, September 2019.

[BDN+13] Begül Bilgin, Joan Daemen, Ventzislav Nikov, Svetla Nikova, Vincent Rijmen, and Gilles Van Assche. Efficient and first-order DPA resistant implementations of keccak. In Aurélien Francillon and Pankaj Rohatgi, editors, *Smart Card Research and Advanced Applications - 12th International Conference, CARDIS 2013, Berlin, Germany, November 27-29, 2013. Revised Selected Papers*, volume 8419 of *Lecture Notes in Computer Science*, pages 187–199. Springer, 2013.

[BDZ20] Tim Beyne, Siemen Dhooghe, and Zhenda Zhang. Cryptanalysis of masked ciphers: A not so random idea. In Shiho Moriai and Huaxiong Wang, editors, *ASIACRYPT 2020, Part I*, volume 12491 of *LNCS*, pages 817–850. Springer, Cham, December 2020.

[CFW04] Lusheng Chen, Fang-Wei Fu, and Victor K.-W. Wei. On the constructions and nonlinearity of binary vector-output correlation-immune functions. *J. Complex.*, 20(2-3):266–283, 2004.

[CGLS20] Gaëtan Cassiers, Benjamin Grégoire, Itamar Levi, and François-Xavier Standaert. Hardware private circuits: From trivial composition to full verification. Cryptology ePrint Archive, Report 2020/185, 2020.

[CGM+24] Gaëtan Cassiers, Barbara Gigerl, Stefan Mangard, Charles Momin, and Rishub Nagpal. Compress: Generate small and fast masked pipelined circuits. *IACR TCHES*, 2024(3):500–529, 2024.

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

[CMM+24] Gaëtan Cassiers, Loïc Masure, Charles Momin, Thorben Moos, Amir Moradi, and François-Xavier Standaert. Randomness generation for secure hardware masking - unrolled trivium to the rescue. *CiC*, 1(2):4, 2024.

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

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

[DR20] Joan Daemen and Vincent Rijmen. *The Design of Rijndael - The Advanced Encryption Standard (AES), Second Edition.* Information Security and Cryptography. Springer, 2020.

[DRB+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, *CHES 2016*, volume 9813 of *LNCS*, pages 194–212. Springer, Berlin, Heidelberg, August 2016.

[DSM22] Siemen Dhooghe, Aein Rezaei Shahmirzadi, and Amir Moradi. Second-order low-randomness d + 1 hardware sharing of the AES. In Heng Yin, Angelos Stavrou, Cas Cremers, and Elaine Shi, editors, *ACM CCS 2022*, pages 815–828. ACM Press, November 2022.

[FGM+18] Sebastian Faust, Vincent Grosso, Santos Merino Del Pozo, Clara Paglialonga, and François-Xavier Standaert. Composable masking schemes in the presence of physical defaults & the robust probing model. *IACR TCHES*, 2018(3):89–120, 2018.

[GGJR11] Benjamin Jun Gilbert Goodwill, Josh Jaffe, and Pankaj Rohatgi. A testing methodology for side-channel resistance validation. In *NIST non-invasive attack testing workshop*, volume 7, pages 115–136, 2011.

[GHP+21] Barbara Gigerl, Vedad Hadzic, Robert Primas, Stefan Mangard, and Roderick Bloem. Coco: Co-design and co-verification of masked software implementations on CPUs. In Michael Bailey and Rachel Greenstadt, editors, *USENIX Security 2021*, pages 1469–1468. USENIX Association, August 2021.

[GIB18] Hannes Gross, Rinat Iusupov, and Roderick Bloem. Generic low-latency masking in hardware. *IACR TCHES*, 2018(2):1–21, 2018.

[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 TCHES*, 2024(1):309–335, 2024.

[GMK16] Hannes Gross, Stefan Mangard, and Thomas Korak. Domain-oriented masking: Compact masked hardware implementations with arbitrary protection order. Cryptology ePrint Archive, Report 2016/486, 2016.

[GMO01] Karine Gandolfi, Christophe Mourtel, and Francis Olivier. Electromagnetic analysis: Concrete results. In Çetin Kaya Koç, David Naccache, and Christof Paar, editors, *CHES 2001*, volume 2162 of *LNCS*, pages 251–261. Springer, Berlin, Heidelberg, May 2001.

[HB25] Vedad Hadzic and Roderick Bloem. Efficient and composable masked AES s-box designs using optimized inverters. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2025(1):656–683, 2025.

[KJJ99] Paul C. Kocher, Joshua Jaffe, and Benjamin Jun. Differential power analysis. In Michael J. Wiener, editor, *CRYPTO'99*, volume 1666 of *LNCS*, pages 388–397. Springer, Berlin, Heidelberg, August 1999.

[KM22] David Knichel and Amir Moradi. Low-latency hardware private circuits. In Heng Yin, Angelos Stavrou, Cas Cremers, and Elaine Shi, editors, *ACM CCS 2022*, pages 1799–1812. ACM Press, November 2022.

[KSM20] David Knichel, Pascal Sasdrich, and Amir Moradi. SILVER - statistical independence and leakage verification. In Shiho Moriai and Huaxiong Wang, editors, *ASIACRYPT 2020, Part I*, volume 12491 of *LNCS*, pages 787–816. Springer, Cham, December 2020.

[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, *CHES 2014*, volume 8731 of *LNCS*, pages 580–597. Springer, Berlin, Heidelberg, September 2014.

[Mat94] Mitsuru Matsui. Linear cryptanalysis method for DES cipher. In Tor Helleseth, editor, *EUROCRYPT'93*, volume 765 of *LNCS*, pages 386–397. Springer, Berlin, Heidelberg, May 1994.

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

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

[Mül25] Nicolai Müller. Prolead - wiki - results. https://github.com/ChairImpSec/PROLEAD/wiki/Results, 2025. Accessed: July 14, 2025.

[NGPM22] Rishub Nagpal, Barbara Gigerl, Robert Primas, and Stefan Mangard. Riding the waves towards generic single-cycle masking in hardware. *IACR TCHES*, 2022(4):693–717, 2022.

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

[RBN+15] Oscar Reparaz, Begül Bilgin, Svetla Nikova, Benedikt Gierlichs, and Ingrid Verbauwhede. Consolidating masking schemes. In Rosario Gennaro and Matthew J. B. Robshaw, editors, *CRYPTO 2015, Part I*, volume 9215 of *LNCS*, pages 764–783. Springer, Berlin, Heidelberg, August 2015.

[SBHM20] Pascal Sasdrich, Begül Bilgin, Michael Hutter, and Mark E. Marson. Low-latency hardware masking with application to AES. *IACR TCHES*, 2020(2):300–326, 2020.

[SG07] Berkeley Logic Synthesis and Verification Group. Abc: a system for sequential synthesis and verification, release 70930, 2007.

[SM21] Aein Rezaei Shahmirzadi and Amir Moradi. Re-consolidating first-order masking schemes. *IACR TCHES*, 2021(1):305–342, 2021.

[Sug18] Takeshi Sugawara. 3-share threshold implementation of AES s-box without fresh randomness. *IACR TCHES*, 2019(1):123–145, 2018.

[SZZ95] Jennifer Seberry, Xian-Mo Zhang, and Yuliang Zheng. Relationships among nonlinear criteria (extended abstract). In Alfredo De Santis, editor, *EUROCRYPT'94*, volume 950 of *LNCS*, pages 376–388. Springer, Berlin, Heidelberg, May 1995.

[UHA17] Rei Ueno, Naofumi Homma, and Takafumi Aoki. Toward more efficient DPA-resistant AES hardware architecture based on threshold implementation. In Sylvain Guilley, editor, *COSADE 2017*, volume 10348 of *LNCS*, pages 50–64. Springer, Cham, April 2017.

[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 TCHES*, 2024(3):249–272, 2024.

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

[WFW+25] Lixuan Wu, Yanhong Fan, Weijia Wang, Bart Preneel, and Meiqin Wang. Extending randomness-free first-order masking schemes and applications to masking-friendly s-boxes. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2025(1):340–366, 2025.

[WGK13] Clifford Wolf, Johann Glaser, and Johannes Kepler. Yosys-a free verilog synthesis suite. In *Proceedings of the 21st Austrian Workshop on Microelectronics (Austrochip)*, volume 97, 2013.

[WM18] Felix Wegener and Amir Moradi. A first-order SCA resistant AES without fresh randomness. In Junfeng Fan and Benedikt Gierlichs, editors, *COSADE 2018*, volume 10815 of *LNCS*, pages 245–262. Springer, Cham, April 2018.

[YCW+24] Fu Yao, Hua Chen, Yongzhuang Wei, Enes Pasalic, Feng Zhou, and Limin Fan. Optimizing AES threshold implementation under the glitch-extended probing model. *IEEE Trans. Comput. Aided Des. Integr. Circuits Syst.*, 43(7):1984–1997, 2024.

[ZCF25] Feng Zhou, Hua Chen, and Limin Fan. Prover - toward more efficient formal verification of masking in probing model. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2025(1):552–585, 2025.

[ZCFY25] Feng Zhou, Hua Chen, Limin Fan, and Junhuai Yang. Domain-oriented masking revisited: More efficient AES implementations with arbitrary protection order. *IACR Cryptol. ePrint Arch.*, page 1358, 2025.

[ZSS+21] Sara Zarei, Aein Rezaei Shahmirzadi, Hadi Soleimany, Raziyeh Salarifard, and Amir Moradi. Low-latency Keccak at any arbitrary order. *IACR TCHES*, 2021(4):388–411, 2021.

A Explanation and additional experiments regarding to the peaks in Figure 7h

The overall value of the TVLA test in Figure 7f is below 4.5. However, when inspecting the trace series of the development of the TVLA value in Figure 7h, it seems there is a 'detectable' leakage in case of using approximately 1.8 million to 1.9 million traces.

In fact, it is not surprising that the t-value occasionally exceeds the threshold. The evolution of maximum t-values across the number of traces is also reported in [ADN+22, Figure 10], where all three designs in [ADN+22] exhibit occasional peaks above 4.5 as well.

Here, we provide a possible explanation for these (false) peaks. The false-positive rate for a single sample point in the t-test is approximately 0.001%. Assuming the sample points behave independently, the false-positive rate for an N-point trace becomes $\beta = 1 - 0.99999^N$. In our 40-cycle design, $N = 6666$, which yields $\beta \simeq 6.4\%$, i.e., several orders of magnitude larger than the single-point false-positive rate.

To confirm the peaks above 4.5 in Figure 7h are indeed false positives, we conducted eight additional TVLA experiments. Each experiment collected 2 million traces, and t-values were computed every 40,000 traces.

The results are summarized in Table 5 below. Each row reports the experiment index, the maximum absolute t-value exceeding the threshold, the corresponding clock cycle, and the number of traces at which this t-value occurs. Importantly, the final maximum t-values of all eight experiments remain below 4.5.

Table 5: Results of additional experiments

No.	max |t-statistics|	Cycle	# Traces
1	4.5003	30	320,000
	4.81	4	960,000
2	4.57	12	600,000
	4.8	12	640,000
	4.9	12	680,000
	4.57	12	720,000
	4.69	12	760,000
3	-	-	-
4	4.507	19	760,000
5	-	-	-
6	4.64	39	80,000
7	4.56	25	920,000
8	4.575	16	200,000

It can be observed that none of the supposedly 'leaky' clock cycles is reproducible across experiments, which indicates these peaks are false-positives.

Finally, since we computed t-values $50 \times 8 = 400$ times in total, and the per-test false-positive rate is $\beta = 6.4\%$, the expected number of false positives is approximately $400 \times 6.4\% = 25.6$. In our experiments, we observed 11 such cases, which is well within the statistical expectations.