A General Randomness Recycling Framework for First-Order Masking with Application to AES

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

[1] TCA Laboratory, Institute of Software, Chinese Academy of Sciences, Beijing, China
[2] University of Chinese Academy of Sciences, Beijing, China
[3] Zhongguancun Laboratory

Abstract. Masking is a principal countermeasure against side-channel attacks, yet its practical deployment is often constrained by the high cost of randomness. Existing approaches for reducing randomness overhead generally follow two directions. The first focuses on designing low-randomness gadgets, which often introduces considerable area and latency overheads for complex boolean functions. The second relies on architecture-level randomness reuse, but securely managing the resulting algebraic dependencies typically still requires additional fresh randomness or extra initial randomness. In this work, we introduce a dependency-tracking abstraction, termed unique randomness guard (URG), for reasoning about randomness reuse in masked hardware circuits. We develop a randomness recycling methodology that eliminates historical randomness dependencies before safely reusing the historical randomness to subsequent computations. This enables secure randomness reuse without requiring additional fresh randomness. To demonstrate the practicality of the proposed methodology, we apply it to first-order masked AES hardware implementations in both parallel and serial architectures. The security of these architectures is proven under the robust probing model and practically validated via TVLA on an FPGA platform. Furthermore, hardware synthesis results demonstrate that our implementations achieve highly competitive area and latency performance compared to state-of-the-art low-randomness designs, while using randomness solely for input encoding.

Keywords: Hardware Masking · Randomness Reuse · AES · Provable Security

1 Introduction

As Internet of Things (IoT) devices proliferate across domains from smart homes to critical infrastructure, the security of their embedded cryptographic applications has become a cornerstone of digital trust. Classical cryptography, however, often assumes an idealized "black-box" model where algorithms are only vulnerable to mathematical attacks. In practice, this assumption is fundamentally challenged by Side-Channel Attacks (SCAs), which exploit

physical leakages such as power consumption, electromagnetic emissions, and timing information to extract secret keys directly from hardware implementations [KJJ99,Koc96].

Among the various countermeasures, masking has emerged as a leading provably secure technique. Based on secret sharing, it splits sensitive variables into multiple shares such that any subset below a threshold reveals no information about the secret [Mes01,GP99,CJRR99]. The security of such schemes was formalized by Ishai, Sahai, and Wagner in the probing model, which defines an implementation as d-th order secure if any combination of up to d intermediate values is statistically independent of the secret [ISW03]. While this model provides a rigorous foundation, practical hardware effects—notably glitches—introduce additional leakage [NRR06], motivating stronger frameworks such as the robust probing model [FGM$^+$18].

A central milestone in this context is Threshold Implementation (TI), which enforces correctness, non-completeness, and uniformity to provide provable resistance against glitches [NRR06]. However, these guarantees often come with substantial cost: the original construction required $td + 1$ shares to protect a function of algebraic degree t at security order d [DNR19]. For complex nonlinear functions such as the AES S-box ($t = 7$), this leads to prohibitive area overhead. The standard solution is to decompose complex functions into quadratic gadgets [MPL$^+$11,BGN$^+$15,GMK17,DRB$^+$16,UHA17,WM18]. This strategy, while managing the share count, amplifies the need for fresh randomness, as the uniformity property—a prerequisite for secure composition—must be restored after each non-linear stage.

In parallel, a separate lineage of masking schemes [RP10,CPRR14,RBN$^+$15,DRB$^+$16,GMK17], evolving from [ISW03] achieved higher share-count efficiency, culminating in constructions requiring only the theoretical minimum of $d+1$ shares. The trade-off for their share-count efficiency, however, is a significant reliance on fresh randomness.

Thus, while starting from different principles, both major masking lineages converge on a common bottleneck in their practical application: a substantial dependence on fresh, online randomness. Approaches to reduce the randomness footprint of masked implementations can be broadly categorized into two synergistic philosophies: optimizing the internal design of cryptographic gadgets and managing randomness at the architectural level across multiple components.

To reduce the randomness consumption, one approach focuses on the meticulous, component-level construction of the masked primitives, aiming to make them inherently efficient and negate the need for randomness [WM18,Sug18,SM21,BDRS22,**?**]. Such designs often involve intricate algebraic manipulations to eliminate the dependency on randomness, which can be challenging for complex non-linear components or leading to significant area and latency overheads. Alternatively, an architectural-level strategy reduces randomness costs by reusing it across the entire system. The most prominent technique in this category is Changing of the Guards (COTG), which substitutes fresh randomness with shares of other, unrelated state

variables—referred to as guards [CPRR14]. The methodology of randomness reuse has evolved, expanding from techniques tailored to functions with specific algebraic properties, such as bijections, to more systematic strategies applicable to a wider range of complex cryptographic structures [GKM+24]. The selection of data for reuse must be carefully managed in accordance with the algorithm's internal structure and diffusion properties to prevent introducing new security vulnerabilities. Also, to achieve provable multi-round security generally requires the periodic injection of fresh randomness [WM18,Sug18,GKM+24,YCW+24], or relies on some extra initial randomness beyond randomness for input encoding [CPRR14]. Existing studies have usually combined the two philosophies, integrating fine-grained component optimizations with global reuse strategies. While such hybrid approaches achieve notable efficiency gains, their applicability is often constrained by the structural details of the underlying components [YCW+24,WM18,Sug18,GKM+24]. Ultimately, their security guarantee over multiple rounds still relies on the availability of either initial or online randomness. However, in practical hardware implementations, generating high-quality randomness—whether through initial injection or an online random number generator (RNG)—typically entails significant overhead in terms of area. This is especially pronounced for resource-constrained devices like IoT nodes and smart cards. Therefore, ensuring the security of cryptographic implementations under severely limited randomness resources has become a critical problem to be solved.

Contributions

In this work, we make three main contributions:

1. **A dependency-tracking representation for reasoning about randomness reuse.** We introduce a dependency-oriented abstraction for reasoning about randomness reuse in masking schemes, explicitly capturing the evolution of dependencies beyond uniformity-based analyses.
2. **A new methodology for randomness dependency management.** Building upon this abstraction, we develop a methodology for managing randomness dependencies in masked implementations. The central idea is to eliminate historical randomness dependencies before reusing the historical randomness in subsequent computations. Based on this principle, we construct randomness recycling strategies for both parallel and serial round-based architectures, enabling secure reuse of existing randomness without requiring additional online randomness.
3. **Low-randomness AES implementations with security analysis.** We apply our strategy to AES algorithm and evaluate their security. The resulting construction achieves first-order masking security against glitches while using no additional randomness beyond that required for masking the plaintext and key. Furthermore, our design shows competitive area and latency performance compared to state-of-the-art low-randomness AES masking implementations.

2 Preliminaries

This section lays the theoretical groundwork for our security analysis. We first formalize computations as circuits and fix the notation used throughout the paper. Then, we introduce probing models that capture adversarial capabilities in both idealized and physically-aware settings, culminating in an information-theoretic security criterion that serves as our formal target.

2.1 Circuit Model and Notation

We employ a computational model adapted from [ISW03], where a computation is formalized as a circuit C. Specifically, a circuit in our context consists of the following components:

- **Inputs and outputs.** Inputs are collected in the set of random variables $\mathcal{X}$, and outputs in $\mathcal{Y}$. We assume all inputs are secret.
- **Gates.** Elementary Boolean operators (AND, XOR, NOT) that form combinational logic.
- **Wires.** Directed edges that carry values between gates, collected in the set $\mathcal{W}$. At runtime, each wire $w_i \in \mathcal{W}$ carries a random variable.
- **Registers.** Sequential elements that separate combinational blocks in time.

Formally, C is represented as a directed acyclic graph (DAG) at the level of combinational blocks separated by registers. Notational conventions: caliphic letters $(\mathcal{X}, \mathcal{Y}, \mathcal{A}, \mathcal{B}, \mathcal{Q}, \mathcal{R})$ denote random variables on wires; lowercase letters (x, y, a, b, q, r) denote specific values of these random variables; uppercase letters denote sets, e.g., $\mathcal{X}$ for inputs, $\mathcal{R}$ for randomness sources, $\mathcal{W}$ for all wires, $\mathcal{Q} \subseteq \mathcal{W}$ for candidate probing points, and $\mathcal{Y}$ for outputs.

2.2 Probing Models for Security Analysis

As outlined in the introduction, probing models serve as the standard formalism for evaluating the resistance of masked circuits to side-channel attacks. This section recalls their formal definitions, following a notation similar to that used in [DBR19], which will be employed in the subsequent analysis.

In the d-probing model, the adversary may simultaneously observe up to d variables from W, the set of all wire-associated random variables of the masked circuit $\hat{\mathcal{C}}$. A single probe corresponds to some $Q \in W$, and a set of d probes corresponds to $\mathcal{P} = \{Q_1, \ldots, Q_d\} \subseteq W$.

Definition 1 (d-Probing Security [ISW03]). *A masked circuit $\hat{\mathcal{C}}$ is d-probing secure if, for any $P \subseteq W$ with $|P| = d$, the joint distribution of the probed variables is statistically independent of any sensitive input $x \in \mathrm{Dom}(X)$. Formally,*

$$\Pr(P \mid x) = \Pr(P),$$

which ensures that no information about X can be inferred from the observation of at most d wires.

Although this model provides a clean abstraction, it neglects hardware-level effects such as glitches, i.e., transient switching behaviors that may cause a wire' s instantaneous value to depend on all combinational inputs. To account for this, the robust probing model extends the semantics of a probe: instead of observing only one variable, a probe on a wire includes both the output and all its driving inputs. Each probe is therefore associated with a subset $Q \subseteq W$ determined by the structure of the circuit—for instance, a combinational gate induces a set consisting of its output together with all its input variables, while registers or latches delimit such sets in sequential logic. Probing d wires then reveals the union $P_{\text{robust}} = \bigcup_{i=1}^{d} Q_i$.

Definition 2 (Robust d-Probing Security [FGM+18]). *A masked circuit is robust d-probing secure if, for any collection of d extended probes, the joint distribution of all revealed variables is statistically independent of the sensitive input X. That is,*

$$\Pr(P_{\text{robust}} \mid X) = \Pr(P_{\text{robust}}) .$$

It is immediate that robust d-probing security implies standard d-probing security, as each extended probe covers at least one wire variable. The robust model therefore enforces a strictly stronger requirement.

3 Unique Randomness Guards

For first-order masked implementations, existing composability frameworks often rely on the uniformity of intermediate variables to guarantee security. However, in randomness reuse scenarios, uniformity alone does not explicitly capture how randomness dependencies evolve throughout the computation. As a result, it becomes difficult to reason about whether reused randomness remains algebraically independent from the data it protects.

To facilitate such analysis, we introduce a notion termed unique randomness guard(URG). Rather than serving as a universal characterization of masking schemes, URG is intended as a lightweight dependency-tracking abstraction tailored to the randomness recycling scenarios considered in this work. The central purpose of this representation is to explicitly track the association between variables and the randomness that protects them, thereby enabling reasoning about dependency elimination and safe randomness reassignment during computation.

Although URG does not capture all possible masking scenarios, it is sufficient for the architecture-level randomness reuse strategies developed in this paper. This abstraction will serve as the basis for the randomness recycling methodology introduced in the following sections.

3.1 Unique Randomness Guard and Its Properties

Notation. Let the original inputs of the circuit C be denoted by $\mathcal{X}$, and let the set of random variables be

$$\mathcal{R} = \mathcal{R}_{\text{in}} \cup \mathcal{R}_{\text{cal}},$$

$\mathcal{R}_{\text{in}}$ denotes randomness consumed by the encoder to mask the inputs, and $\mathcal{R}_{\text{cal}}$ denotes randomness introduced during the calculation inside the masked circuit $\hat{C}$. Let $\mathcal{A} = \{a^1, \ldots, a^n\}$ denote the internal variables (signals) of the masked circuit.

We use the symbol $\mathtt{F}$ to denote a boolean function

$$\mathtt{F} : \mathbb{F}_2^{|\mathcal{S}|} \to \mathbb{F}_2,$$

where $\mathcal{S}$ is a source random variable vector. When applied to a concrete assignment of $\mathcal{S}$, denoted by $\mathcal{S}^*$, $\mathtt{F}$ returns the corresponding Boolean value. When the subscript $\mathcal{S}$ is attached, i.e. $\mathtt{F}_{\mathcal{S}}$, it denotes the random variable obtained by evaluating $\mathtt{F}$ on the random vector $\mathcal{S}$.

In particular, each $a \in \mathcal{A}$ can be written as

$$a = \mathtt{F}(\mathcal{X}, \mathcal{R}),$$

meaning that a is the random variable generated by some boolean function of the coordinates in $\mathcal{X} \cup \mathcal{R}$.

Definition 3 (Unique Randomness Guard (URG)). *Let $m := |\mathcal{R}_2|$ and $\mathcal{R}_2 \subseteq \mathbb{F}_2^m$. Suppose there exists a function*

$$\mathtt{G} : \mathbb{F}_2^m \longrightarrow \mathbb{F}_2^n$$

such that each element $a^i \in \mathcal{A}$ can be expressed as

$$a^i = \mathtt{F}^i(\mathcal{X}, \mathcal{R} \setminus \mathcal{R}') \oplus \mathtt{G}^i(\mathcal{R}'),$$

where $\mathtt{G}^i(\cdot)$ denotes the i-th coordinate function of $\mathtt{G}$.

We say that $\mathcal{A}$ possesses a set of unique randomness guards or $\mathcal{A}$ holds the URG property if $\mathtt{G}$ is balanced, we denote this by $\mathtt{G} \sim \mathcal{U}(\mathbb{F}_2^n)$. That is, for all $y \in \mathbb{F}_2^n$, the cardinality of the preimage $|\mathtt{G}^{-1}(y)|$ is constant (specifically 2^{m-n}).

Theorem 1 (URG Implies Joint Uniformity). *Let $\mathcal{A} = \{a^1, \ldots, a^n\}$ possess the URG property with guard randomness $\mathcal{R}'$. Then, the variables $a^1, \ldots, a^n$ are jointly uniform.*

Proof. Let $m := |\mathcal{R}'|$ and let $\Lambda := \mathcal{X} \cup (\mathcal{R} \setminus \mathcal{R}')$ denote the set of non-guard variables. Fix an arbitrary assignment Λ^*. This determines a constant vector $\mathcal{C}^* \in \mathbb{F}_2^n$ defined by $\mathcal{C}^* = \mathtt{F}_{\Lambda^*}$.

For any specific assignment $\mathcal{A}^* \in \mathbb{F}_2^n$, the conditional probability is derived via the counting measure on the randomness space $\mathbb{F}_2^m$:

$$\begin{aligned}
\Pr[\mathcal{A} = \mathcal{A}^* \mid \Lambda = \Lambda^*] &= \frac{1}{2^m} \cdot |\{\mathcal{R}'^* \in \mathbb{F}_2^m \mid \mathcal{C}^* \oplus \mathtt{G}(\mathcal{R}'^*) = \mathcal{A}^*\}| \\
&= \frac{1}{2^m} \cdot |\{\mathcal{R}'^* \in \mathbb{F}_2^m \mid \mathtt{G}(\mathcal{R}'^*) = \mathcal{A}^* \oplus \mathcal{C}^*\}| \\
&= \frac{1}{2^m} \cdot \left|\mathtt{G}^{-1}(\mathcal{A}^* \oplus \mathcal{C}^*)\right|.
\end{aligned}$$

By the URG definition, $\mathtt{G}$ is balanced, implying $|\mathtt{G}^{-1}(\mathcal{V}^*)| = 2^{m-n}$ for any vector $\mathcal{V}^* \in \mathbb{F}_2^n$. Substituting this into the equation:

$$\Pr[\mathcal{A} = \mathcal{A}^* \mid \Lambda = \Lambda^*] = \frac{2^{m-n}}{2^m} = 2^{-n}.$$

Since this probability is constant for all $\mathcal{A}^*$, the joint distribution factorizes into the product of marginal probabilities:

$$\Pr[\mathcal{A} = \mathcal{A}^*] = 2^{-n} = \prod_{i=1}^{n} \frac{1}{2} = \prod_{i=1}^{n} \Pr[a^i = \mathcal{A}_i^*],$$

where $\mathcal{A}_i^*$ denotes the i-th coordinate of the assignment $\mathcal{A}^*$. Thus, the variables $a^1, \ldots, a^n$ are mutually independent.

Consequently, the URG property is strictly stronger than uniformity. However, in standard masked circuit designs, independent randomness guards are typically introduced for each variable (for example, generating a masked share a_0 can be represented as $a_0 = a \oplus r_a$ with r_a as a unique randomness guard). Therefore, in such cases, the inputs can be easily represented in URG form.

Having established that the URG property rigorously guarantees the joint uniformity and independence of variables, the next critical step is to investigate whether this structural property can be preserved throughout the actual computation of a cryptographic algorithm. Block ciphers are predominantly constructed from alternating linear and non-linear layers. We first examine the behavior of the URG property under linear transformations. It is crucial to note that the URG property is not universally preserved under arbitrary linear operations. As demonstrated in the following theorem, it is inherently maintained if the linear mapping is surjective. In practice, standard cryptographic linear layers are typically bijective, thus naturally satisfying this requirement.

Theorem 2 (Preservation of URG Property under Surjective Linear Mappings). *Let the set of variables $\mathcal{A}$ possess a URG of the form:*

$$\mathcal{A} = \mathtt{F}(\mathcal{X} \cup (\mathcal{R} \setminus \mathcal{R}')) \oplus \mathtt{G}(\mathcal{R}')$$

Suppose the map $\mathtt{H} : \mathbb{F}_2^n \to \mathbb{F}_2^k$ is both linear and surjective. Then the new set of variables $\mathcal{A}' = \mathtt{H}(\mathcal{A})$ holds the URG property.

Proof. Since the function $\mathtt{H}$ is linear, its application distributes over the XOR sum. This allows the new variable set $\mathcal{A}'$ to be expressed using the composed functions:

$$\mathcal{A}' = \mathtt{H} \circ (\mathtt{F}(\mathcal{X} \cup (\mathcal{R} \setminus \mathcal{R}')) \oplus \mathtt{G}(\mathcal{R}')) = (\mathtt{H} \circ \mathtt{F})(\mathcal{X} \cup (\mathcal{R} \setminus \mathcal{R}')) \oplus (\mathtt{H} \circ \mathtt{G})(\mathcal{R}').$$

Let $\mathtt{F}' := \mathtt{H} \circ \mathtt{F}$ and $\mathtt{G}' := \mathtt{H} \circ \mathtt{G}$. To show that $\mathcal{A}'$ satisfies the URG property, it suffices to prove that the composed guard function $\mathtt{G}' : \mathbb{F}_2^m \to \mathbb{F}_2^k$ is balanced.

The function $\mathtt{G}$ is balanced by the definition of URG for $\mathcal{A}$. This implies that for any specific value $\mathcal{Y}^* \in \mathbb{F}_2^n$, the preimage size is constant:

$$|\mathtt{G}^{-1}(\mathcal{Y}^*)| = 2^{m-n}.$$

Similarly, since $\mathtt{G}$ is a surjective linear map, it is inherently balanced. For any target value $\mathcal{Z}^* \in \mathbb{F}_2^k$, the set of its preimages under $\mathtt{G}$ has a constant size determined by the kernel dimension:

$$|\mathtt{G}^{-1}(\mathcal{Z}^*)| = 2^{n-k}.$$

Now, consider the preimage size of the composite function $\mathtt{G}'$ for an arbitrary output $\mathcal{Z}^* \in \mathbb{F}_2^k$:

$$\begin{aligned}
|\mathtt{G}'^{-1}(\mathcal{Z}^*)| &= \sum_{\mathcal{Y}^* \in \mathtt{G}^{-1}(\mathcal{Z}^*)} |\mathtt{G}^{-1}(\mathcal{Y}^*)| \\
&= \sum_{\mathcal{Y}^* \in \mathtt{G}^{-1}(\mathcal{Z}^*)} 2^{m-n} \\
&= |\mathtt{G}^{-1}(\mathcal{Z}^*)| \cdot 2^{m-n} \\
&= 2^{n-k} \cdot 2^{m-n} = 2^{m-k}.
\end{aligned}$$

Since the preimage size is constant for all $\mathcal{Z}^*$, the function $\mathtt{G}'$ is balanced. Therefore, $\mathcal{A}'$ holds the URG property.

Following Theorem 2, surjective linear components can be computed share-wise, inherently preserving the URG property without consuming fresh randomness. However, achieving a first-order masked implementation for non-linear components (such as S-box) is considerably more complex. The necessary cross-share multiplications inherently disrupt the initial independence and entangle the algebraic expressions with internal randomness. Consequently, restoring the uniformity and independence of the outputs—especially under constrained randomness budgets—becomes a central challenge. To intuitively understand how this randomness dependency is managed, the following subsection re-examines several classical randomness reduction techniques for first-order AND gates through the lens of the URG property.

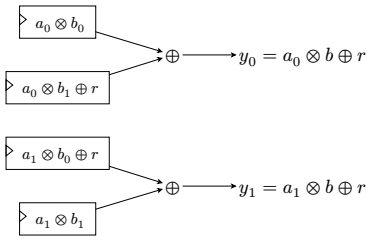
3.2 An intuitive interpretation of previous randomness reduction techniques through the lens of URG

In this subsection, we apply the URG perspective to analyze classical randomness reduction strategies, primarily focusing on the first-order AND gate. It should be noted that while the security of these original constructions is traditionally analyzed based on the macroscopic property of uniformity, the URG framework imposes a stronger assumption on the specific format of the inputs. We adopt this formalism not to provide a universally exhaustive analytical method, but rather to offer a concise and intuitive perspective. Given the requisite assumptions

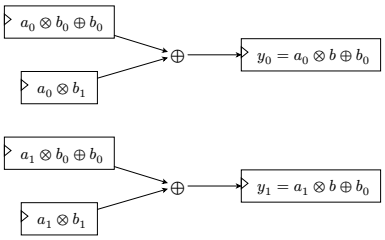
about the input format, URG serves as a convenient tool to trace the flow of randomness. We begin with the strict assumption that the two inputs of the AND gate initially possess unique, independent randomness guards. For visual clarity, the specifically chosen randomness guards are highlighted in blue. The inputs can be represented as follows:

$$\begin{cases} a_0 = a \oplus r_a \\ a_1 = r_a \end{cases} \qquad \begin{cases} b_0 = b \oplus r_b \\ b_1 = r_b \end{cases}$$

To understand the evolution of these techniques, we first establish the baseline. In conventional first-order masked AND gates, securing cross-domain multiplications strictly requires the injection of a fresh random bit to prevent leakage.



The hardware construction proposed by Shahmirzadi and Moradi [SM21] alters this approach by reusing the existing input mask (e.g., r_b) to protect the computation, thereby eliminating the need for an online PRNG. As illustrated by the hardware dataflow as follows:



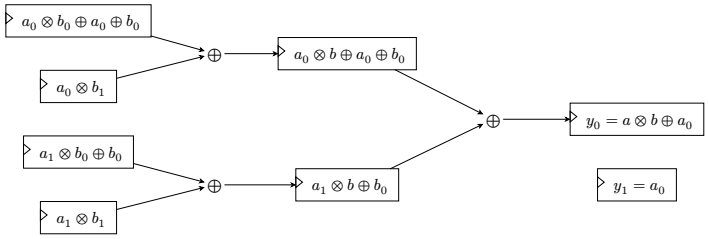
Algebraically, the output shares produced by this hardware architecture can be expressed in the URG format as follows:

$$\begin{cases} y_0 = a_0 \otimes b \oplus b_0 & = (a \oplus r_a \oplus 1) \otimes b \oplus r_b \\ y_1 = a_1 \otimes b \oplus b_0 & = r_a \otimes b \oplus r_b \end{cases}$$

While this design avoids fresh randomness by utilizing r_b as the guard, the outputs retain a dependency on the other input's randomness, r_a. Consequently,

when this component is composed with other variables that also depend on r_a, caution is required to avoid unintended correlations that could potentially expose the secret.

Subsequent masking schemes, such as those introduced by De Cnudde et al. and Gross et al. [GSDM+19,?], propose a different construction for the AND gate. The hardware implementation for such a design is shown below:



Correspondingly, the mathematical representation of its output shares evaluates to the following expressions:

$$\begin{cases} y_0 = a \otimes b \oplus a_0 & = a \otimes b \oplus a \oplus r_a \\ y_1 = a_0 & = a \oplus r_a \end{cases}$$

In these expressions, the dependency on r_b is completely removed from the output shares. From the perspective of the URG, this signifies that the output inherits r_a as its unique and independent guard. This structure reduces the risk of leakage when interacting with variables that do not share r_a. Furthermore, since r_b is entirely excluded from the output state, it is effectively released and can be safely recycled in subsequent computation steps without creating algebraic dependencies.

However, despite the progress represented by these improved gadgets, existing hardware implementations have not yet fully realized the generalized principles underlying the elimination of algebraic dependencies. Existing implementations largely treat the elimination of algebraic dependencies as a local, gadget-specific optimization rather than a fundamental architectural principle.

The underlying mechanism for reducing randomness cost remains under-exploited at the system level. This lack of a principled methodology leads to significant hardware costs, such as prohibitive area overhead and high latency. This gap motivates the formalization of our recycling methodology in the following sections.

3.3 The URG Restoration Gadget: Securing Randomness Recycling

In essence, the logic behind these low-randomness designs is that the elimination of algebraic dependency on a specific input mask (e.g., r_b) decouples that variable

from the current computational state, thereby creating the formal possibility for its reuse. By explicitly formulating this underlying mechanism, we provide the theoretical foundation to move beyond specific gadget-level and establish a principled methodology for randomness recycling.

Consider a target function $y = \mathtt{F}^y(\mathcal{X})$, where the unmasked result y is purely a deterministic function of the secret input $\mathcal{X}$. Without loss of generality, the output of a secure two-share, first-order masked component for this function can be expressed as $y^i = \mathtt{F}^i(\mathcal{X}, \mathcal{R})$. Due to the multiplication of masked shares, each individual share (y_0 and y_1) typically retains a complex algebraic dependency on the entire set of historical random variables $\mathcal{R}$ used within the circuit. If any random variable from $\mathcal{R}$ were to be reused in a subsequent computation involving y_0 or y_1, unintended correlations could emerge, leading to potential security flaws. Notice, however, that as long as the masking is correct, the unmasked evaluation $y = y_0 \oplus y_1 = \mathtt{F}^y(\mathcal{X})$ remains a deterministic function completely independent of any internal randomness. This algebraic identity implies that while individual shares are deeply entangled with $\mathcal{R}$, their joint sum provides a clean foundation for eliminating the randomness dependency. This insight motivates the design of a specialized gadget that can operationalize this decoupling. We formalize an atomic cryptographic primitive termed the **Recycle Gadget** as follows:

$$\begin{cases} y'_0 = [[y_0 \oplus r'] \oplus y_1] = \mathtt{F}^y(\mathcal{X}) \oplus r' \\ y'_1 = r' \end{cases}$$

As shown in the algebraic derivation, the resulting shares y'_0 and y'_1 depend exclusively on the deterministic function $\mathtt{F}^y(\mathcal{X})$ and the new randomness guard r'. Under the condition that r' is independent of the historical randomness pool $\mathcal{R}$, the complex dependencies of the initial shares are effectively collapsed into the unmasked result y. By substituting the entangled internal state with such an independent guard, the algebraic ties to $\mathcal{R}$ are removed. This property ensures that variables within $\mathcal{R}$ can be subsequently recycled without introducing unintended correlations that would lead to security flaws.

The improved AND gate design discussed in Section 3.2 can be viewed as a specific instance of this general principle. In that construction, the y_0 calculation internally merges the components necessary to compute y, which effectively eliminates the dependency on the input mask.

However, these previous designs remained confined to the localized optimization of individual components. By formalizing this dependency elimination mechanism, the focus of gadget design can be fundamentally reassessed: instead of specifically constructing components that operate without randomness, we can instead utilize randomness to reduce gadget cost—such as area and latency—provided that the resulting dependencies are recycled at the output boundary.

Furthermore, the principle of the recycling gadget can be extended to high-order masking schemes. For instance, in a d-share, d-th order masked implementation, the output of each share can be expressed as a function of the input and the entire randomness pool. By applying a similar recombination

of each share securely, we can effectively decouple the output from the historical randomness, enabling the reuse of random variables in subsequent computations.

4 Randomness Recycling Strategy for Most First-Order Round-Based Ciphers Masking

Previously, we detailed the algebraic properties of the unique randomness guard (URG) and introduced the recycle gadget to enable local randomness recycling. Building upon this theoretical foundation, this section further explores how to extend recycle gadget to protect complete modern cryptographic algorithms in hardware implementations.

For round-based cryptographic algorithms, our core objective is to ensure that the global state of the algorithm maintains the URG property entirely after each encryption round, without consuming any online fresh randomness. This is crucial because the output state of the current round can only be securely utilized as the input for the next iteration if it preserves the URG property.

Since surjective linear layers naturally preserve the URG property without incurring additional randomness overhead, the primary architectural challenge resides in the non-linear layer. Targeting the two typical architectures in hardware implementations, this section first distills a general lemma for randomness reuse, and subsequently presents specific implementation strategies for both parallel and serial architectures.

4.1 Basic Principles of Randomness Reuse

When applying the unique randomness guard (URG) abstraction to cryptographic hardware, the sequential nature of actual circuits must be considered. Hardware registers update with clock signals, and physical wires and computational resources are frequently reused across cycles. Therefore, secure randomness reuse requires tracking the dynamic evolution of data across multiple clock cycles. This section explains how such reused randomness can achieve equivalence to independent fresh randomness under the first-order robust probing model.

To accurately characterize the dynamic evolution across clock cycles, we incorporate time indices into our notation. Let $\mathrm{Reg}^{(t)}$ denote a register variable at clock cycle t, and let $\mathrm{Guard}(\mathrm{Reg}^{(t)})$ represent its associated randomness source. In practice, a computational component may receive states from different registers as randomness sources across various cycles. We collectively denote the set of reused guards introduced by such a component as $\mathcal{G}_{reuse}$.

Based on these concepts, we formalize the fundamental principle for secure randomness reuse in sequential circuits as follows:

Lemma 1 (Lemma of dynamic randomness reuse). *For a hardware computational component subject to repeated reuse, as long as the values in the reused guard set $\mathcal{G}_{reuse}$ are distinct across different clock cycles and do*

not conflict with the component's own input guards, the computation process is strictly equivalent to using independent fresh randomness under the first-order robust probing model.

Proof. Under the first-order robust probing model, registers serve as physical barriers that prevent glitch propagation, confining potential leakage to a single clock cycle. Since $\mathcal{G}_{reuse}$ consists of distinct guards, the properties of URG guarantee that the involved variables are joint uniform distribution as a whole. Furthermore, since cross-cycle algebraic correlations cannot be observed under first-order robust probing model, this overall uniformity ensures that the reused variables are statistically equivalent to independent fresh randomness.

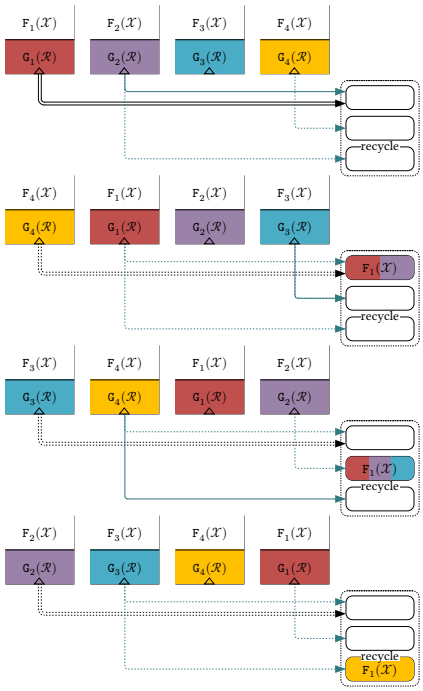


Fig. 1: The dataflow of randomness borrowing and recycling in a single component across four consecutive clock cycles. In each register, the upper half represents the masked data state, while the lower half indicates the associated independent guard ($\mathbf{G}_i(\mathcal{R})$, distinguished by different colors). Solid and dashed arrows denote the active and inactive data paths in the current cycle for current computation, respectively. The recycle gadget illustrates the re-masking process of old states with new guards.

Figure 1 provides an intuitive example of the aforementioned lemma, illustrating a dynamic reuse mechanism by detailing how $F_1(\mathcal{X})$ is evaluated over consecutive clock cycles. As mentioned in lemma 1, the key to this mechanism is that, although the computational component uses data from the register array through a fixed physical wiring, the cyclic shifting of the register states ensures that the guards fetched on these fixed wires (denoted by $\mathbf{G}_i(\mathcal{R})$ in the figure) are continuously updated over successive clock cycles. For instance, as shown in the figure, although Reg_2 is utilized twice during the computation, its associated guards are entirely different: $\mathrm{Guard}(\mathrm{Reg}_2^{(1)}) = \mathtt{G}_2(\mathcal{R})$, whereas $\mathrm{Guard}(\mathrm{Reg}_2^{(3)}) = \mathtt{G}_4(\mathcal{R})$. Since the fetched variables in $\mathcal{G}_{reuse}$ are demonstrably distinct across cycles, this specific wiring and shifting combination satisfies lemma 1, ensuring the component receives equivalent fresh randomness under the first-order robust probing model at zero extra cost. Moreover, the above figure primarily illustrates the reuse strategy under conditions of sufficient random bits. In most cases, this strategy is adequate. However, under certain extreme circumstances, a shortage of random bits may occur. To address this, we can introduce additional recycle gadgets during the computation process to eliminate dependencies of previously used random bits. This allows a single random bit to be reused multiple times, thereby further reducing the demand for random bits.

Furthermore, the figure also depicts a recycle gadget to illustrate how randomness dependencies are eliminated during the reuse process. By introducing the recycle gadget, the previously reused randomness is completely decoupled from the input data of the current computation. This ensures that the current operation can safely reuse prior randomness without introducing security vulnerabilities. This mechanism will be detailed further in the subsequent architectural designs.

4.2 Randomness Recycling for Parallel Architecture

This subsection primarily explores, through an intuitive example, how to utilize a rational randomness recycling strategy within parallel hardware architecture to ensure that the output state of a concurrent S-Box array fully inherits the URG property, assuming the initial inputs already possess the property.

Maintain URG Property through Cross-Guard Recycling. The core principle for the parallel S-box architecture is formulated as follows: the i-th S-box extracts the randomness from the input of the $\pi(i)$-th S-box to serve as the new guard for its output state. As long as the mapping π is a bijection without fixed points (i.e., $\forall i, \pi(i) \neq i$), this design ensures that the randomness used by each component is algebraically independent of its input data. Crucially, this bijective mapping also dictates the global preservation of the URG property. Because π is strictly bijective, it acts as a permutation over the finite set of existing input guards. This ensures that no guard is duplicated or discarded during the reassignment. Consequently, the resulting output array inherits the exact same complete set of mutually independent guards, merely in a permuted order.

This rigorous preservation guarantees that the collective output of the entire non-linear layer inherently maintains the macroscopic URG property, securing the overall state for subsequent computational stages without generating online randomness. Additionally, any randomness utilized for the internal computations of the masked S-box must be strictly disjoint from the guards recycled at the output boundaries.

Security Analysis Regarding the security analysis of the parallel architecture, the randomness fetching strategy of an individual S-Box can directly reuse the aforementioned local wiring strategies. Under the assumption of the first-order robust probing model, the observation scope of a single probe is strictly restricted, rendering it incapable of simultaneously extracting the internal states of two concurrent components. Given this physical isolation, the global security proof simplifies to a matter of local verification. Provided that each component independently satisfies the security criteria, the first-order security of the overall non-linear layer is guaranteed.

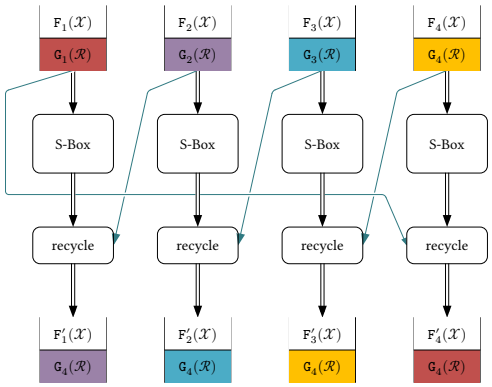


Fig. 2: Dataflow of randomness recycling in parallel masked S-box architecture.

Figure 2 illustrates an example of four parallel masked S-boxes (internal randomness wiring for each S-box is omitted). As shown in the figure, each masked S-box is connected to a recycle gadget at its output. These gadgets do not consume fresh randomness; instead, they directly extract inherent randomness guard from the inputs of other S-boxes (denoted as $\mathtt{G}_1(\mathcal{R}) \sim \mathtt{G}_4(\mathcal{R})$ in the figure). Concretely, the recycle gadget of the first S-box uses the input guard from the second S-box, while the fourth S-box use its guard from the first S-box's input.

4.3 Randomness Recycling for Serial Architecture

In serial pipeline architectures, we likewise proceed from the premise that the initial global state strictly possesses the URG property. Distinct from

the single-cycle global updates in parallel architectures, the core of a serial architecture relies on time-division multiplexing. This introduces a unique theoretical challenge: how to maintain the macroscopic URG property of the entire register array across consecutive clock cycles while only updating a fractional subset of the data at any given time.

Maintain URG Property through Dynamic Replacement. The design space for serial randomness recycling heavily depends on the permissible hardware overhead. If one were to use extra registers to temporarily buffer guards, the recycling strategy would become trivially flexible. However, to rigorously preserve the low-area advantage of serial architectures, we deliberately constrain our framework to use limited registers for guard management. The fundamental principle is that whenever a newly computed state is written back into the array, it must directly inherit the exact randomness guard of the old state that is simultaneously being discarded or overwritten. By maintaining a perfect "one-in, one-out" instantaneous substitution, the total collection of variables within the register array maintain URG property. When extra registers are used, the design can be further relaxed to allow for more flexible mappings, as long as the overall URG property is preserved.

Furthermore, it is important to note a key distinction in randomness sourcing between the two architectures. While the parallel architecture can flexibly fetch the randomness from other S-boxes, the serial architecture relies on fixed hardware wiring. However, based on the preserved relative order within the register array, this internal randomness can be securely borrowed. By assigning randomness using different fixed offsets, it is straightforward to ensure that the entire set of guards fetched for internal operations is mutually distinct, and completely avoids algebraic collisions with both the current input guard and the guard used for recycling. Since the entire data array shifts cyclically over time, any borrowing rule based on a fixed offset that is secure for a specific data index i will inherently remain secure for all subsequent data entries. This mapping satisfies the dynamic reuse lemma, simplifying the hardware logic while ensuring the rigorous security of the time-multiplexed computation with zero online randomness consumption.

Security Analysis Regarding the security analysis of the serial architecture, physical wires are reused, but the input guards fetched on these wires remain independent across different clock cycles. Synchronous registers placed between pipeline stages act as physical boundaries under the first-order robust probing model, preventing the propagation of glitches. This temporal isolation prevents an attacker from utilizing a single probe to simultaneously observe cross-cycle algebraic correlations. Since the extracted guards in the reused set are demonstrably distinct across cycles, this specific temporal multiplexing mechanism is equivalent to operating with independent fresh randomness, guaranteeing the first-order security.

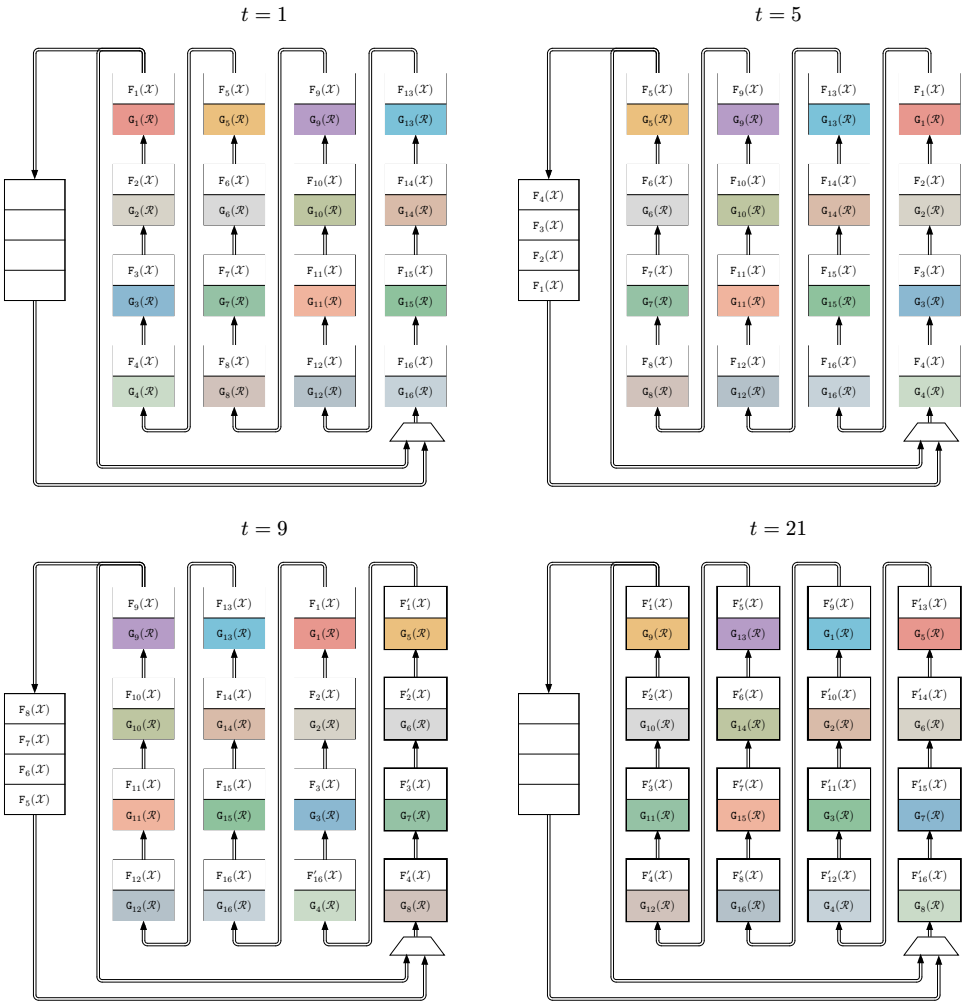


Fig. 3: Dataflow of randomness recycling in serial masked S-box architecture.

Figure 3 illustrates how this serial architecture operates. Since the masked S-box and its associated recycle gadget inherently introduce a fixed computation latency of four clock cycles, the initial data elements are fed into the pipeline and simultaneously routed back into the shift-register array to maintain the complete set of guards. This fixed four-cycle latency directly dictates the physical wiring for secure, no-extra-register randomness recycling: when the computation result F_i' for the i-th data entry is ready, the shift-register array has exactly shifted four times. At this precise clock cycle, the data entry being shifted out is the one originally located at index $(i + 4) \bmod 16$. Consequently, the hardware is hardwired to extract the guard from this specific position. For instance, at $t = 9$,

F_1' is written back exactly when F_5 is shifted out; through fixed wiring, F_1' natively inherits the guard $\mathcal{G}_5(\mathcal{R})$, completing the seamless URG preservation without consuming any online randomness or extra register buffers.

4.4 Discussion

Generalizability and Design Flexibility The randomness reuse strategies presented for the parallel and serial architectures serve as archetypal instantiations rather than rigid blueprints. They demonstrate how to maintain the URG property. This core principle—guaranteeing that the global state inherits the URG property after each round—is broadly applicable to a wide range of round-based cryptographic algorithms. Grounded in this theoretical perspective, designers can flexibly tailor routings to meet specific hardware constraints, algorithmic structures, or practical engineering requirements.

Superiority over Traditional COTG Compared to traditional Changing of the Guards (COTG) mechanisms, a defining advantage of our URG-based methodology is its extended scope. While traditional COTG typically focuses on state re-masking at component boundaries, our approach systematically secures the reuse of randomness for the internal non-linear operations of the masked S-boxes as well. By formally tracking and eliminating algebraic dependencies both internally and externally, this framework empowers the implementation to operate with strictly zero online randomness consumption.

Extension to Transition Leakage Resilience While our baseline time-multiplexed serial design effectively mitigates spatial glitches under the standard robust probing model, the URG perspective also provides a natural pathway to mitigate transition leakages. To achieve first-order security under a probing model that considers transition leakage, it is imperative that the guards utilized by constituent registers remain strictly distinct over consecutive clock cycles. Guided by the URG framework, this can be engineered by rationally restructuring the datapath. For instance, designers can partition the input data into two independent groups. By ensuring that the randomness sourced for both internal computation and output recycling strictly originates from the same isolated group, the hardware naturally alternates these sources over time. This interleaved routing strategy guarantees that adjacent pipeline stages invariably process data with different guards, thereby resisting transition leakages while preserving the elegant zero-online-randomness property.

5 Case Study: Masked AES Implementation

In this section, we move from theory to practice by applying the strategies established in section 4 to the Advanced Encryption Standard (AES). Our primary objective is to demonstrate the practical feasibility of constructing

a first-order secure, masked hardware implementation that operates with the initial randomness used for input encoding.

The remainder of this section is organized as follows: Subsection 5.1 details the hardware design of the S-box and the specific datapath routing strategies used to achieve global conflict-free randomness recycling in both the parallel and serial architectures. Subsection 5.2 rigorously evaluates the security of our designs, combining security analysis with practical Test Vector Leakage Assessment (TVLA) experiments on an FPGA platform. Subsection 5.3 presents the synthesis results. It benchmarks our hardware area and latency against previous low-randomness designs, confirming that our approach successfully reduces the cost of randomness while maintaining highly competitive performance.

5.1 Implementation Overview

Implementation of the Masked S-Box To demonstrate the practical efficiency of our broadly applicable randomness reuse strategy, we implement a high-performance first-order AES S-box based on the time sharing masking (TSM) [**?**]. We chose this specific component primarily for its excellent latency performance, as it requires only two clock cycles per evaluation. This efficiency effectively meets the low-latency requirements of modern masked implementations. It requires 38 bits of randomness in the first clock cycle and 8 bits of randomness in the second clock cycle. The 46 bits of randomness can be easily fetched from other input shares of the state and key arrays. We did not modify the TSM S-box design to accommodate our randomness reuse strategy, but rather directly applied our general rules to the original TSM design. This demonstrates the versatility and practicality of our approach, as it can be seamlessly integrated with existing high-performance masked S-box implementations without requiring any modifications.

Parallel Architecture Design This subsection describes the round-based, fully parallel hardware architecture designed to achieve first-order security with zero online randomness. The design can be partitioned into two concurrent datapaths: the encryption datapath and the key-expansion datapath. A central feature is the strategic management of masked data and the internal sourcing of randomness for non-linear operations.

Figure 4 illustrates the datapath of the parallel architecture AES implementation. Double-lined arrows represent data flows containing two shares, while single-lined arrows denote the $0-th$ share used exclusively for providing randomness. The encryption datapath and the key-expansion datapath execute computations synchronously. For all linear operations, a simple share-wise implementation is adopted. Furthermore, we use recycling gadgets at the output of the S-boxes. The specific randomness reuse strategy is detailed as follows.

As illustrated, figure 5 defines the strategy for randomness reusing and recycling. Red nodes represent the target bytes processed by the S-box. Blue

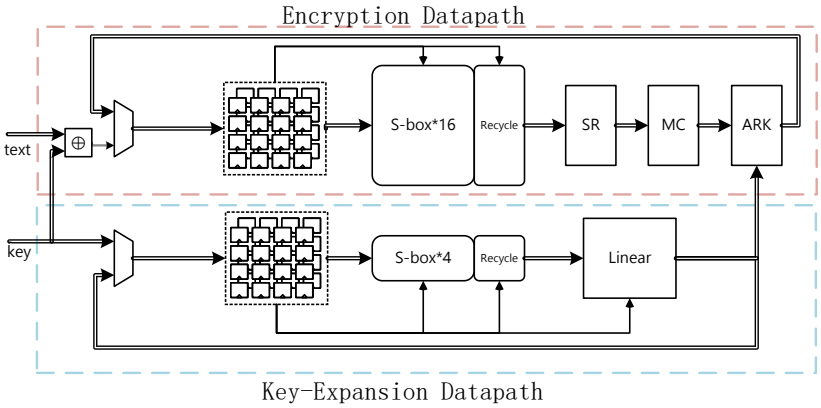


Fig. 4: Dataflow diagram: Parallel AES hardware architecture

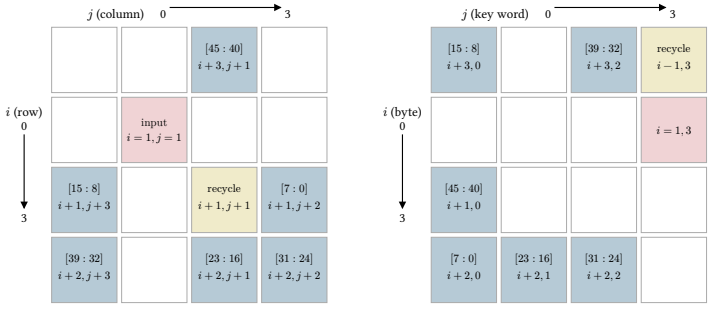


Fig. 5: Randomness reusing and recycling strategy for parallel AES.

nodes are data sources providing randomness for internal non-linear operations, with brackets (e.g., [15:8]) specifying the exact bit intervals supplied. If a blue source requires fewer than 8 bits (e.g., [45:40]), the hardware simply extracts the lower bits and discards the rest. Yellow nodes identify the randomness sources for the recycling gadgets. All components fetch randomness via fixed offsets (e.g., $i + 1, j + 1$). This design guarantees that the guards for the recycling gadgets remain globally conflict-free, maintaining the register array's urg property without online randomness.

In the parallel architecture, the S-box computation for each round takes 4 clock cycles, followed by 1 clock cycle to write the results of the linear operations back to the registers. Consequently, a single complete round requires 5 clock cycles. Including 1 clock cycle for initialization, the entire 10-round encryption process requires a total of 51 clock cycles.

Serial Architecture Design This subsection presents a specific implementation of the serial AES architecture. The overall datapath is illustrated as follows.

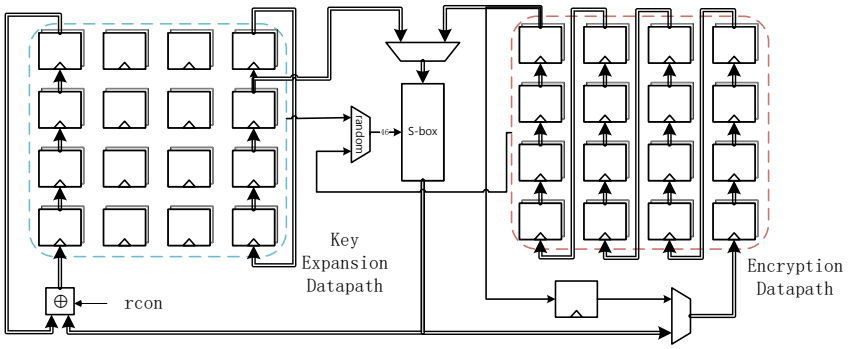


Fig. 6: Dataflow diagram: Serial AES hardware architecture

Figure 6 illustrates the datapath of the serial architecture. The design utilizes a shift-register array to achieve time-division multiplexing, allowing the encryption and key-expansion paths to share a single central S-box. Double-lined arrows represent data flows containing two shares, while single-lined arrows denote the 0th share used exclusively for providing randomness. All linear operations within the architecture are implemented share-wise. To maintain the clarity of the diagram, the datapath for the linear operations' feedback is omitted.

During the non-linear computation, the key and state data are sequentially shifted into the S-box cycle by cycle. To prevent side-channel leakage, when the multiplexer in the feedback loop selects its input, the old data that is about

to be removed from the array is first buffered in an independent register for one clock cycle. This register severs the direct logical connection between this to-be-removed data and the new S-box output, avoiding glitch leakages caused by both possessing the exact same guards. At the S-box output, the recycling gadget relies on the fixed latency of the pipeline, using hardwired connections to directly extract the Guard of the to-be-removed data for re-masking.

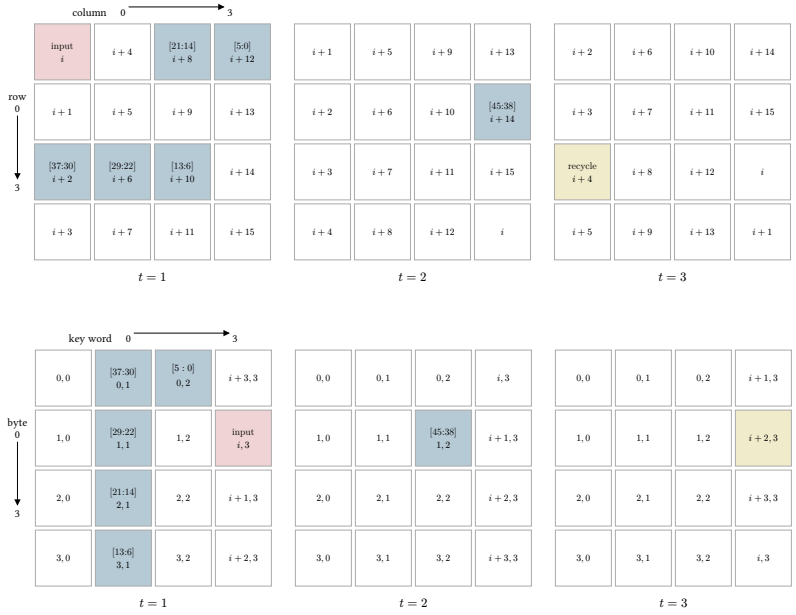


Fig. 7: Randomness reusing and recycling strategy for serial AES.

Figure 7 details the randomness routing and recycling strategy for the serial architecture across consecutive clock cycles (t). The node colors (red, blue, yellow) carry the same meaning as in the parallel architecture, representing the computation input, randomness for S-box computation, and recycling gadgets, respectively.Note that although the complete non-linear computation of the S-box requires 4 clock cycles, randomness is only consumed during the first 3 cycles; thus, the figure only illustrates the array states from $t = 1$ to $t = 3$. Due to the continuous shifting of the registers, the guards distribution within the array rotates dynamically over time (subscripts like i and $i + 1$ in the figure intuitively denote the changes in their relative positions).Taking the state data as an example: at $t = 1$, data i enters the S-box. At $t = 3$, when the recycling gadget needs to fetch randomness, the hardware directly extracts the guard from the physical location (yellow node) holding the old data originally at the relative position $i + 4$. Due to the 4-cycle latency of the pipeline, this specific

data is exactly the one that will be shifted out when the current computation finishes and the result is written back. The key-expansion datapath follows the same timing mapping, but differs in specific physical coordinates. As for randomness used for S-box computations, the system utilizes the property that certain columns in the key schedule (e.g., Column 1 and 2 in the figure) do not shift, serving as stable randomness sources. During the re-masking extraction phase at $t = 3$, if the input coordinate at $t = 1$ was $(i, 3)$, the Guard index of the old data at this physical location has changed to $i+2$ due to the cyclic shift logic of the specific column. Therefore, the hardware directly extracts the guard here (yellow node) for recycling. The above mapping rules ensure that the various randomness sources do not conflict.

In the serial architecture, each round sequentially performs the key-related S-box computations followed by the encryption-related S-box computations. Given the 4-cycle latency of the S-box pipeline, a single round takes a total of 24 clock cycles. The write-back operation for the linear transformations during encryption is scheduled at the beginning of the subsequent round. Accounting for one cycle for initialization and one cycle for final write-back operation after the last round, the entire process requires a total of 242 clock cycles.

5.2 Security Analysis

This section presents the formal security arguments for the implemented parallel and serial AES hardware architectures, demonstrating that they satisfy the security requirements of the first-order robust probing model with exactly zero online randomness consumption.

Security Proof for Parallel Architecture In the instantiation of the parallel architecture, the global security of the system is founded upon the independent verification of each masked S-box. Under the first-order robust probing model, the 20 masked S-boxes within the encryption and key-expansion datapaths operate in parallel with no shared combinational logic , which restricts the observation scope of a single probe within an individual component. This physical isolation allows the global security proof of the entire non-linear layer to simplify directly to the local independent verification of each masked S-box. The fixed coordinate offset wiring adopted in this design satisfies the theoretical requirement of a bijective mapping without fixed points. This ensures that for both the state array in the encryption datapath and the array in the key-expansion datapath, the guard used at the output of each masked S-box for recycling is algebraically independent of its own input data and any randomness involved in internal computations. Moreover, all linear operations in this architecture are implemented share-wise and the results are written back to registers. Since these cryptographic linear transformations in AES are bijective, they naturally and preserve the URG property. Consequently, both the state array and the key-expansion array flawlessly maintain the global URG property after completing the computations of both the non-linear and linear layers, achieving first-order security without consuming any online randomness.

Security Proof for Serial Architecture The security argument for the serial architecture focuses on proving that its time-division multiplexing mechanism complies with the theoretical requirements of dynamic randomness reuse. This architecture sequentially feeds data into a single masked S-box pipeline via a shift-register array. The continuous rotation of the shift-register array guarantees that the reused randomness sets introduced across different clock cycles are entirely distinct and non-conflicting. According to Lemma 1 (the lemma of dynamic randomness reuse), this explicit time separation renders the entire time-multiplexed computational process equivalent to operating with statistically independent fresh online randomness under the first-order robust probing model. Furthermore, for all linear operations in this architecture, since they are similarly implemented share-wise without any resource multiplexing , these bijective linear transformations do not disrupt the URG property. When a non-linear computation result is ready to be written back, the system precisely utilizes the fixed pipeline latency to extract the protective randomness associated with the old data being shifted out or overwritten for re-masking. This one-in, one-out same-cycle substitution mechanism ensures that the total collection of protective randomness within the shift-register array remains constant during continuous updates. Therefore, the system naturally and continuously maintains the complete URG property during continuous computations, fundamentally proving the security of this time-multiplexed architecture under the premise of zero online randomness consumption.

Experimental Evaluation To practically verify the effectiveness of our implemented scheme against side-channel attacks, we conducted an experimental evaluation using the TVLA framework [GJJR11]. TVLA is a widely adopted, statistics-based methodology that tests for distinguishable differences in a device's power consumption when processing different classes of input data. The framework's null hypothesis (H_0) posits that no such statistical difference exists. A resulting Welch's t-statistic that remains within the established confidence threshold of ± 4.5 indicates that no significant side-channel leakage is detected.

Experimental Platform Our crypto-implementation was deployed on Sakura-G. We captured power traces using a PicoScope 6404D oscilloscope over a shunt resistor on the FPGA's power line. It is noteworthy that the evaluation board used does not integrate any dedicated hardware random number generator instances.

Evaluation Methodology Our specific testing methodology was the fixed-versus-random t-test. We created two distinct sets of power traces: a "fixed set" by repeatedly encrypting a constant plaintext, and a "random set" by encrypting a new random plaintext for each trace. A constant encryption key was used throughout all measurements. To eliminate time-varying environmental factors, the encryptions for both sets were performed in a interleaved manner. In our evaluation, we collected 10^7 traces to test the parallel architecture and 5×10^6 traces to test the serial architecture.

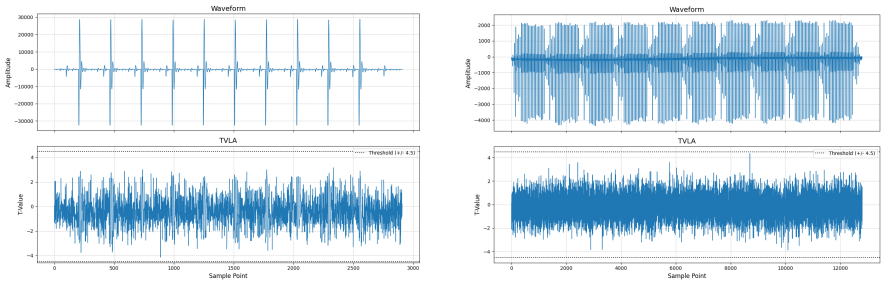


Fig. 8: TVLA t-test results of the parallel (left) and serial (right) architectures.

Results Our experimental results clearly demonstrate the effectiveness of the implementations. As shown in Figure 8, the t-statistic results do not exceed the significance threshold of ± 4.5. This held true across the 10^7 traces collected for the parallel architecture, as well as the 5×10^6 traces collected for the serial architecture. This result provides strong experimental evidence that our implementation can effectively resist first-order attacks.

5.3 Implementation Results and Comparison

To evaluate the hardware area overhead of our proposed AES implementation, we first utilize the sv2v [Sno25] tool to convert the SystemVerilog implementation into synthesizable Verilog code. Subsequently, we employ the open-source synthesis tool Yosys [20225] and synthesize the design based on the NanGate 45nm [The] open-source standard cell library to measure its gate area.

As shown in Table 1, our proposed implementations are the first hardware implementations to achieve first-order security with randomness exclusively used for input encoding. Our performance comparison primarily focuses on AES implementations with low randomness consumption. All prior state-of-the-art hardware implementations either rely on a static pool of tens to hundreds of random bits [WM18,Sug18,ADN+21], or necessitate fresh randomness [YCW+24,**?**]. Additionally, a prior proposal introduces the use of 2-bit randomness (including randomness for input encoding); however, this approach has only been validated in software, and its theoretical hardware implementation is estimated to incur an impractical latency of 24 clock cycles per S-box [GSDM+19]. Furthermore, a recent work [**?**] proposed a randomness-free AES implementation; however, its S-box requires 9 clock cycles per evaluation and suffers from a relatively large area cost.

In contrast, our designs substantiate the feasibility of implementing first-order secure AES masking using solely the initial randomness required for plaintext and key sharing. Both the parallel and serial architectures maintain a competitive overall balance, offering acceptable area cost, low latency, and zero online randomness. It should be emphasized that this work serves primarily as an initial validation of our core URG methodology. Further architectural

Table 1: Comparison of hardware implementation overheads for different first-order side-channel countermeasures

Design	S-box Instances	Area (kGE)		Latency (cycles)		S-box share	Randomness (bits)		
		S-box	AES	S-box	AES		S-box	AES	Fixed*
[WM18]	–	–	–	16	2804	4	0	0	384
[Sug18]	1	3.50	17.10	4	266	3	0	0	264
[ADN+21] Design1	20	2.71	63.91	5	62	2	0	0	54
[ADN+21] Design2	20	4.33	102.44	4	52	2	0	0	36
[ADN+21] Design3	20	5.84	115.73	3	42	2	0	0	40
[YCW+24] Design1	1	–	–	6	236	2	4	800	0
[YCW+24] Design2	2	–	–	6	148	2	4	800	0
[**?**]	1	4.17	8.805	4	290	2	4	800	0
Parallel	**20**	**5.481**	**116.084**	**2+2**	**51**	**2**	**0**	**0**	**0**
Serial	**1**	**5.718**	**16.41**	**2+2**	**242**	**2**	**0**	**0**	**0**

* "Fixed Randomness" denotes all other initial random numbers required by the design, excluding those used for the initial masking of the plaintext and key. "S-box Randomness" refers to the randomness consumed during one S-box computation process."AES Randomness" represents the total randomness consumed throughout the entire encryption process."S-box" area contains the area of the recycling gadget."

optimizations are expected to enhance latency performance and continue to reduce the area footprint, particularly for the serial implementation. Moreover, the URG framework itself can be extended to other cryptographic algorithms, especially lightweight ciphers like PRESENT or SKINNY, where zero online randomness can be achieved to reduce the overall area cost as the random number generator is eliminated.

6 Conclusion and Future Work

Managing the high randomness cost while maintaining provable security remains a central challenge in hardware masking. This paper addresses this challenge by introducing a novel theoretical framework centered on the unique randomness guard (URG) property and recycling gadget. Grounded on this cornerstone property, we derive a systematic methodology for randomness reuse and propose specific design schemes tailored for parallel and serial hardware implementations.

We instantiated our URG-based framework in a first-order secure AES hardware implementation, designing and realizing both parallel and serial architectures. This design achieves provable security with zero extra randomness beyond the initial masking required for plaintext and key sharing, all while maintaining an acceptable balance in overall hardware performance. Its security is validated through rigorous formal proofs and extensive Test Vector Leakage Assessment (TVLA) evaluations on an FPGA platform. Experimental results demonstrate that no first-order leakage is detected after collecting 10^7 power traces for the parallel architecture and 5×10^6 power traces for the serial architecture. This work demonstrates that efficient randomness reuse and rigorous provable security can be achieved simultaneously, opening up a versatile new approach for secure hardware design in resource-constrained environments.

This work lays a foundation for future research in the following directions. As discussed, more fine-tuned designs can be explored to achieve security considering the transition leakage. Moreover, the core design philosophy of our proposed recycling gadget is naturally applicable to higher-order masking implementations, by ensuring that the randomness dependencies introduced by reuse are properly managed and eliminated. Therefore, this work opens a direct path to developing low-randomness implementations for higher-order masking schemes, which is a critical area for future research.

Acknowledgements

We used generative AI tools to assist in drafting and refining the manuscript. AI-generated content was reviewed and edited to ensure accuracy and to reflect our original work.

References

20225. YosysHQ/yosys. Yosys Headquarters, October 2025.

ADN+21. Amund Askeland, Siemen Dhooghe, Svetla Nikova, Vincent Rijmen, and Zhenda Zhang. Guarding the First Order: The Rise of AES Maskings, 2021.

BDRS22. Tim Beyne, Siemen Dhooghe, Adrián Ranea, and Danilo Sijacic. A low-randomness second-order masked AES. In Riham AlTawy and Andreas Hülsing, editors, *SAC 2021*, volume 13203 of *LNCS*, pages 87–110. Springer, Cham, September / October 2022.

BGN+15. Begül Bilgin, Benedikt Gierlichs, Svetla Nikova, Ventzislav Nikov, and Vincent Rijmen. Trade-Offs for Threshold Implementations Illustrated on AES. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 34(7):1188–1200, July 2015.

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.

CPRR14. Jean-Sébastien Coron, Emmanuel Prouff, Matthieu Rivain, and Thomas Roche. Higher-order side channel security and mask refreshing. In Shiho Moriai, editor, *FSE 2013*, volume 8424 of *LNCS*, pages 410–424. Springer, Berlin, Heidelberg, March 2014.

DBR19. Lauren De Meyer, Begül Bilgin, and Oscar Reparaz. Consolidating security notions in hardware masking. *IACR TCHES*, 2019(3):119–147, 2019.

DNR19. Siemen Dhooghe, Svetla Nikova, and Vincent Rijmen. Threshold implementations in the robust probing model. Cryptology ePrint Archive, Report 2019/1005, 2019.

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.

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.

GJJR11. G. Goodwill, Benjamin Jun, J. Jaffe, and P. Rohatgi. A testing methodology for side channel resistance validation. 2011.

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

GMK17. Hannes Groß, Stefan Mangard, and Thomas Korak. An efficient side-channel protected AES implementation with arbitrary protection order. In Helena Handschuh, editor, *CT-RSA 2017*, volume 10159 of *LNCS*, pages 95–112. Springer, Cham, February 2017.

GP99. Louis Goubin and Jacques Patarin. DES and differential power analysis (the "duplication" method). In Çetin Kaya Koç and Christof Paar, editors, *CHES'99*, volume 1717 of *LNCS*, pages 158–172. Springer, Berlin, Heidelberg, August 1999.

GSDM+19. Hannes Gross, Ko Stoffelen, Lauren De Meyer, Martin Krenn, and Stefan Mangard. First-Order Masking with Only Two Random Bits. In *Proceedings of ACM Workshop on Theory of Implementation Security*

Workshop, TIS'19, pages 10–23, New York, NY, USA, November 2019. Association for Computing Machinery.

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

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.

Koc96. Paul C. Kocher. Timing attacks on implementations of Diffie-Hellman, RSA, DSS, and other systems. In Neal Koblitz, editor, *CRYPTO'96*, volume 1109 of *LNCS*, pages 104–113. Springer, Berlin, Heidelberg, August 1996.

Mes01. Thomas S. Messerges. Securing the AES finalists against power analysis attacks. In Bruce Schneier, editor, *FSE 2000*, volume 1978 of *LNCS*, pages 150–164. Springer, Berlin, Heidelberg, April 2001.

MPL+11. Amir Moradi, Axel Poschmann, San Ling, Christof Paar, and Huaxiong Wang. Pushing the limits: A very compact and a threshold implementation of AES. In Kenneth G. Paterson, editor, *EUROCRYPT 2011*, volume 6632 of *LNCS*, pages 69–88. Springer, Berlin, Heidelberg, May 2011.

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.

RP10. Matthieu Rivain and Emmanuel Prouff. Provably secure higher-order masking of AES. In Stefan Mangard and François-Xavier Standaert, editors, *CHES 2010*, volume 6225 of *LNCS*, pages 413–427. Springer, Berlin, Heidelberg, August 2010.

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

Sno25. Zachary Snow. Sv2v: SystemVerilog to Verilog, October 2025.

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

The. The-OpenROAD-Project. NANGATE: The NanGate 45nm Open Cell Library. https://github.com/The-OpenROAD-Project/OpenROAD-flow-scripts/tree/master/flow/platforms/nangate45.

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.

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 Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 43(7):1984–1997, July 2024.