IACR Transactions on Cryptographic Hardware and Embedded Systems
ISSN 2569-2925, Vol. 2022, No. 3, pp. 290–329. DOI:10.46586/tches.v2022.i3.290-329

Side-Channel Masking with Common Shares

Weijia Wang[1,2,3], Chun Guo[1,2,4], Yu Yu[5,6,7], Fanjie Ji[1] and Yang Su[1]

[1] School of Cyber Science and Technology, Shandong University, Qingdao, China, wjwang@sdu.edu.cn, chun.guo@sdu.edu.cn
[2] Key Laboratory of Cryptologic Technology and Information Security, Ministry of Education, Shandong University, Qingdao, China
[3] Quan Cheng Shandong Laboratory, Jinan, China
[4] Shandong Research Institute of Industrial Technology, Jinan, China
[5] Department of Computer Science and Engineering, Shanghai Jiao Tong University, Shanghai, China yuyu@yuyu.hk
[6] Shanghai Qi Zhi Institute, Shanghai, China
[7] Shanghai Key Laboratory of Privacy-Preserving Computation, Shanghai, China

Abstract. To counter side-channel attacks, a masking scheme randomly encodes key-dependent variables into several *shares*, and transforms operations into the masked correspondence (called *gadget*) operating on shares. This provably achieves the de facto standard notion of *probing security*.

We continue the long line of works seeking to reduce the overhead of masking. Our main contribution is a new masking scheme over finite fields in which shares of different variables have a part in common. This enables the reuse of randomness / variables across different gadgets, and reduces the total cost of masked implementation. For security order d and circuit size ℓ, the randomness requirement and computational complexity of our scheme are $\tilde{O}(d^2)$ and $\tilde{O}(\ell d^2)$ respectively, strictly improving upon the state-of-the-art $\tilde{O}(d^2)$ and $\tilde{O}(\ell d^3)$ of Coron et al. at Eurocrypt 2020.

A notable feature of our scheme is that it enables a new paradigm in which many intermediates can be precomputed before executing the masked function. The precomputation consumes $\tilde{O}(\ell d^2)$ and produces $\tilde{O}(\ell d)$ variables to be stored in RAM. The cost of subsequent (online) computation is reduced to $\tilde{O}(\ell d)$, effectively speeding up e.g., challenge-response authentication protocols. We showcase our method on the AES on ARM Cortex M architecture and perform a T-test evaluation. Our results show a speed-up during the online phase compared with state-of-the-art implementations, at the cost of acceptable RAM consumption and precomputation time.

To prove security for our scheme, we propose a new security notion intrinsically supporting randomness / variables reusing across gadgets, and bridging the security of *parallel* compositions of gadgets to *general* compositions, which may be of independent interest.

Keywords: Side-Channel Attack · Masking · Cost Amortization · Precomputation

1 Introduction

Side-channel attacks that exploit leakage emitting from devices pose an important threat for cryptographic implementations. Masking [CJRR99, ISW03] is one of the most investigated protection techniques. The core idea is to randomly split each secret-dependent variable in finite fields (e.g., $\mathbb{F}_2$) into a vector of $d+1$ shares called *sharing*, and then implement the cryptographic algorithm over sharings instead of the raw secrets. This ensures that the initial secret cannot be rebuilt from any d intermediate variables in the implementation, which is called *d-private security* (a.k.a., *d-probing security*). While the leakages of all the

Received: 2022-01-15 Accepted: 2022-03-15 Published: 2022-06-08

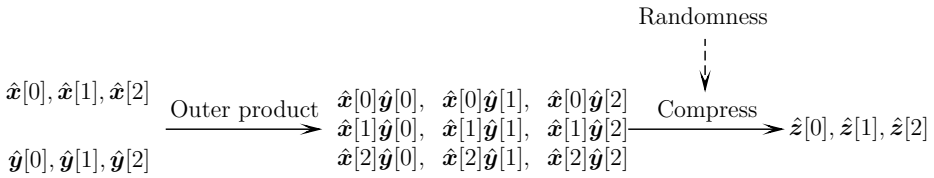


Figure 1: ISW multiplication with $d = 2$.

$d + 1$ shares enable the reconstruction of secrets theoretically, the intrinsic noise in the leakages renders secret recovery infeasible in practice, if the leakages of all the shares are independent [CJRR99, DDF14, GS18, PR13].

To have secure functionalities with shares as input and output, a masking scheme, sometimes called a *private circuit compiler*, firstly constructs *gadgets* over sharings for various elementary calculations, and then composes the gadgets to obtain the desired functionality. In 2003 [ISW03], Ishai, Sahai, and Wagner introduced the first d-probing secure gadget for the multiplication over $\mathbb{F}_2$, which is called ISW multiplication[1]. The scheme takes the additive sharings where the XOR of the shares gives the secret, and outputs the additive sharing of the product. In a nutshell, the ISW multiplication is performed through two steps. First, it calculates the outer product of the input sharings, resulting in a $(d+1) \times (d+1)$ matrix, where the XOR of entries equals the secret output. The second step compresses the entries with some randomness into the output shares. We showcase the procedure with $d = 2$ in Figure 1[2], where input sharings are $\hat{\boldsymbol{x}}$ and $\hat{\boldsymbol{y}}$, and output sharing is $\hat{\boldsymbol{z}}$.

The ISW multiplication requires $(d + 1)d/2$ random bits and runs with complexity (defined by the size of circuit that computes it) $O(d^2)$ [ISW03], making any function of circuit size ℓ admits a masked implementation using $O(\ell d^2)$ bits of randomness, and running with complexity $O(\ell d^2)$. Subsequently, plenty of works have emerged to make the ISW scheme (or its variants) more practical by, e.g., reducing the randomness complexity [BBP+16, BBP+17, KR18, CS19, CS20, BGR18] and ensuring independent leakages of all the shares [GMK16, RBN+15, MPL+11, FGP+18]. Recently, besides optimizing a single gadget, the community has noticed the benefits of *amortization* techniques that reduce the averaged complexity for several masked operations. This roughly follows two directions, namely *randomness reuse* and *code-based masking*.

The first direction aims at reusing random bits in different gadgets. Faust et al. introduced a method allowing secure reuse of some random bits among different gadgets for the threshold implementation [FPS17]. Ishai et al. proposed to expand the randomness using the so-called robust pseudorandom generator (PRG) [IKL+13], where the number of random bits (that should be generated from True Random Number Generator) for seeds is independent of the circuit size. A recent work of Coron et al. describes a very practical construction, making the number of random bits to be only $\tilde{O}(d^2)$ and independent of the circuit size, at the cost of computational complexity growing to $\tilde{O}(\ell d^3)$ due to the runs of PRG [CGZ20].

As for the second direction, recently, Wang et al. showed that a generic type of masking called code-based masking is able to encode multiple secrets together into one codeword and calculate parallel operations over these secrets together in the masked domain, which naturally enables the cost amortization [WMCS20]. Their work resulted in a scheme using $\tilde{O}\big((d+\ell)^2\big)$ random bits with computational complexity $\tilde{O}\big(\ell(d+\ell)^2\big)$, which is then optimized to $\tilde{O}(d^2)$ and $\tilde{O}(\ell d^2)$ respectively in follow-up work [WGS+20]. However, this direction only allows amortization across *parallel* operations, which largely restricts the

[1] Later in [RP10], ISW multiplication was generalized to the case over any finite field.
[2] For completeness, we recall in Appendix A the full description.

application.

1.1 Our Contributions

1.1.1 Cost Amortized Masking with Common Shares

We propose a new masking scheme that could effectively combine the advantages of the two aforementioned directions. In our new scheme, a part of shares of different key-dependent variables is the same. Thereby, randomness and intermediate variables can be reused among different operations, achieving cost amortization. Concretely, as shown in Figure 2 with second-order security in probing model, let $(\hat{\boldsymbol{x}}_i, \hat{\boldsymbol{y}}_i)$ and $(\hat{\boldsymbol{x}}_j, \hat{\boldsymbol{y}}_j)$ be input sharings of two multiplication gadgets over a finite field. Before calculating the outer products, our scheme first refreshes the sharings to $(\hat{\boldsymbol{x}}'_i, \hat{\boldsymbol{y}}'_i)$ and $(\hat{\boldsymbol{x}}'_j, \hat{\boldsymbol{y}}'_j)$ such that the last two shares of $\hat{\boldsymbol{x}}'_i$ and $\hat{\boldsymbol{x}}'_j$ (resp., $\hat{\boldsymbol{y}}'_i$ and $\hat{\boldsymbol{y}}'_j$) are the same, making the lower right 2×2 sub-matrices of outer products the same as well. Then, randomness in the compression and refreshing can be reused across the two multiplications.

Our temporary sharings $\hat{\boldsymbol{x}}'_i$ and $\hat{\boldsymbol{x}}'_j$ cannot be additive (which is different from the ISW multiplication), as otherwise the leakage of $\{\hat{\boldsymbol{x}}'_i[0], \hat{\boldsymbol{x}}'_j[0]\}$ gives rise to $x \oplus y \equiv \hat{\boldsymbol{x}}'_i[0] \oplus \hat{\boldsymbol{x}}'_j[0]$, where x and y are the secret variables of the sharings $\hat{\boldsymbol{x}}'_i$ and $\hat{\boldsymbol{x}}'_j$ respectively, and $\oplus$ is the field addition. This effectively violates probing security. Similarly, the temporary sharings $(\hat{\boldsymbol{y}}'_i, \hat{\boldsymbol{y}}'_j)$ cannot be additive either. To address this issue, we follow the idea of inner product masking [BFG15] and construct the refreshing in such a way that $x \equiv \langle \hat{\boldsymbol{x}}'_i, \boldsymbol{a}_i \rangle$ (resp., $y \equiv \langle \hat{\boldsymbol{y}}'_i, \boldsymbol{a}_i \rangle$) and $x \equiv \langle \hat{\boldsymbol{x}}'_j, \boldsymbol{a}_j \rangle$ (resp., $x \equiv \langle \hat{\boldsymbol{y}}'_j, \boldsymbol{a}_j \rangle$), where $\langle \cdot, \cdot \rangle$ denotes the inner product. Probing security is guaranteed if vectors $\boldsymbol{a}_i$ and $\boldsymbol{a}_j$ are linearly independent. The above adaptions finally contribute to a new masking scheme using $\tilde{O}(d^2)$ random bits with computational complexity $\tilde{O}(\ell d^2)$, where ℓ is the circuit size.

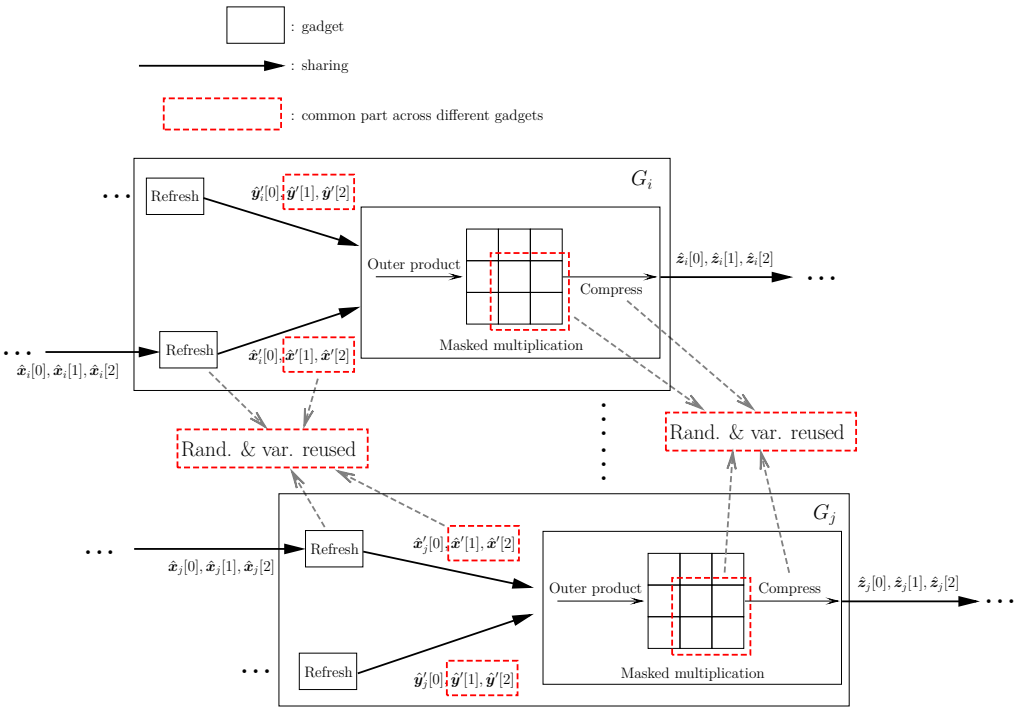


Figure 2: Concept of our scheme with security order $d = 2$.

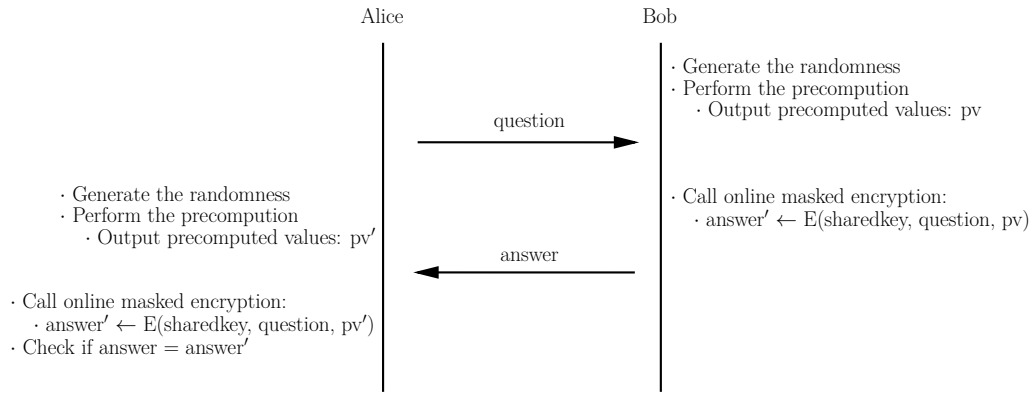


Figure 3: Precomputation-based design paradigm for challenge–response authentication protocols.

1.1.2 Precomputation-based Design Paradigm for Masking

A promising feature of our scheme is that, most of the intermediate variables can be precomputed only by randomness. For example, in Figure 2, the intermediate variables related to the common part should be independent of the input and only determined by randomness. Thus, these variables and their functions can be precomputed for each call of the cryptographic function. Concretely, the masked implementation of a function can be represented as $\hat{f}\big(\hat{\boldsymbol{x}}, \hat{g}()\big)$, where $\hat{\boldsymbol{x}}$ is the sharing of input, and $\hat{f}$ and $\hat{g}$ are deterministic and randomized functions respectively. In such a way, our masking scheme with common shares can be divided into two phases: precomputation and online computation. The former phase uses $\tilde{O}(d^2)$ random bits, runs in $\tilde{O}(\ell d^2)$, and outputs $\tilde{O}(\ell d)$ precomputed variables required to be stored in RAM. The online computation takes the input sharings and precomputed variables and performs the functionality of masked cryptographic algorithm in $\tilde{O}(\ell d)$.

This property can significantly lift our masking scheme in many scenarios. For example, Figure 3 shows the challenge-response authentication protocols (e.g., the authentication and key agreement protocol as main building blocks securing 3/4G and some 5G networks) where Alice presents a question (aka., the "challenge") and Bob must provide a valid answer (aka., the "response") to be authenticated. The answer can be calculated by encryption from the question and a pre-shared symmetric key. To prevent the side-channel attack, the encryptions are protected by masking. Both Alice and Bob can call the function $\hat{g}$ to produce precomputed variables before the answer or question completely arrive, and then they call the $\hat{f}$. This strategy is quite practical since the transferring of answers/questions is relatively slow compared with encryption (even with masking countermeasure).

1.1.3 New Security Notion for Proofs: From Parallel to General Compositions

The naive method of d-private proof is to show by enumeration that the joint distributions of possible tuples of any d intermediate variables are independent of the secret. However, such an enumeration becomes intricate as the size of the function grows, and it is only feasible for small circuits such as a single multiplication gadget. This naturally motivates the composition approach, i.e., proving that under certain conditions, a large circuit built upon d-private gadgets is d-private. In this respect, several composable security notions have been introduced, such as Non-Inference (NI), Strong Non-Inference (SNI) [BBD+16] and Probe-Isolating Non-Inference (PINI) [CS19]. Thanks to those security notions, a composition of gadgets can be proven to be globally d-private secure. However, in our

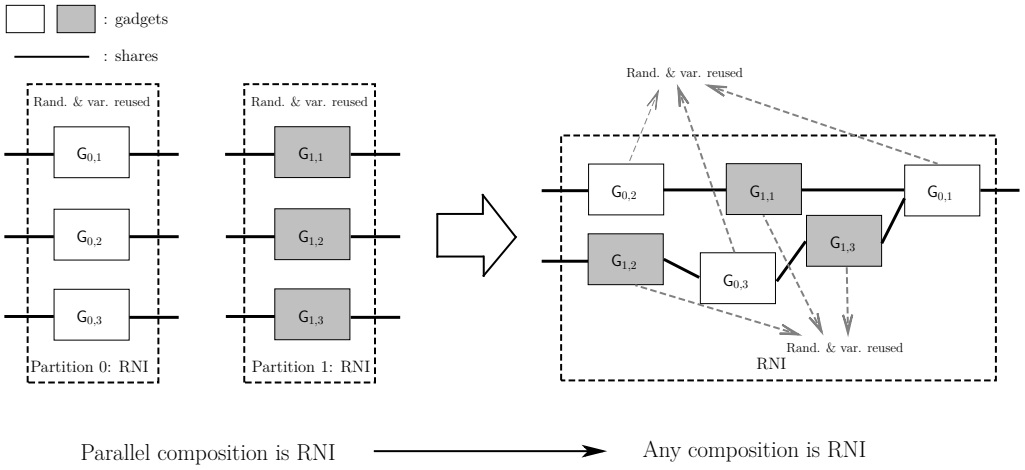


Figure 4: An arbitrary composition (on the right) that can be described a bipartite graph is RNI, as long as the parallel compositions (on the left) of gadgets in each partition is RNI.

scheme, due to the common shares, many intermediate variables and randomness are reused across different gadgets, and thus the previous security notions for single gadgets are not quite suitable. In this regard, new security notions are needed.

Note that security for the case of parallel composition, in which there is no input-to-output connection between different gadgets, is easier to prove than the case of general composition. Inspired by this, first, we put forward a new security notion named *Randomness Reusable Non-Inference (RNI)* [3]. Then, we consider the situation that the gadgets can be divided into two disjoint sets such that: (a) every cross-gadget input-output-connection crosses the two sets, and (b) randomness/intermediate variable-reuse is limited within each set, as illustrated in Figure 4. In other words, the composed gadget is a *bipartite graph* with gadgets as vertexes and connections of gadgets as edges. With the above, any composition that can be described as such a bipartite graph is RNI, as long as the parallel composition of gadgets in each partition is RNI. This enables deducing the proof for any masked implementation to the parallel composition of our new gadgets. Compared with other well-know composable security notions such as NI/SNI, RNI intrinsically supports randomness/variables reusing.

1.1.4 Application to AES

Finally, to show the practical relevance of the new masking, we describe an application of our countermeasure to the block cipher AES-128 in the precomputation-design paradigm. We implement masked AES on the ARM Cortex M architecture, and report the performance results. It shows that our scheme contributes to a speed-up for the online phase compared with the state-of-the-art implementations. Notably, when the security order is $d = 8$, our implementation achieves a gain of up to 141% in the timing, at the cost of 11KB of RAM for storing precomputed intermediates.

We provide a T-test evaluation for our implementation, which validates the security order, but also shows that the more complex algebraic structure of our masking (than the Boolean one) makes it more robust to some lapses (e.g., transitional leakage) in implementation.

[3] Strictly speaking, the new notion is associated with a function called share-scale function defined in Section 3.2. Nevertheless, in introduction, we focus on showing the concept, and omit the technical details.

Noted that, the implementation secure in the probing model should be regarded as the first necessary step to a provably side-channel secure implementation. We provide our result only for illustrative purposes. Many works (e.g., [BDF+17, DFS15, CPR07, MPW21]) have shown that it still challenging to integrate a secure implementation in probing model to a side-channel secure one for a particular platform or micro-architecture. E.g., the implementation should involve sufficient noise, which we deem out of scope and do not investigate in this (already quite long) paper. But, we believe the application result shows the practical relevance of the new masking w.r.t. the efficiency.

1.2 Related Works on Masking with Common Shares

The idea of using common shares in masking has been put forward by Coron et al. in [CGPZ16]. The work first studied the case of two multiplications $z \cdot x$ and $z \cdot y$ over a finite field with a common operand z, and proved that the ISW multiplication gadget can still be secure even if half of the shares of x and y are the same. This saved half field multiplications. Then, the work generalized the case to multiple parallel multiplications without a common operand and saved roughly 1/4 multiplications. Later in [CRZ18], this technique was adopted for the countermeasure of look-up table proposed in [Cor14], and brought a significant speed-up.

Compared with these prior works, our scheme is both more general and more practical. Notably, in our scheme, d shares (out of $d+1$ shares in a sharing) are common across multiple sharings that do not need to be in parallel. That is, considering the case of two multiplications $z_1 \leftarrow x_1 \cdot y_1$ and $z_2 \leftarrow x_2 \cdot y_2$, our scheme allows that

1. d shares of x_1 and x_2 (resp., y_1 and y_2) are the same, and,
2. most importantly, $x_2 = z_1$ or $y_2 = z_1$, i.e., the parallelism of the multiplications is not necessary.

1.3 Organization

Below we first present notations and backgrounds in Section 2. We then introduce our new security notion and related composability rules in Section 3. We describe the new masking scheme and discuss the concrete security in Section 4. Finally, Section 5 presents the application to AES.

2 Preliminaries

2.1 Notations

In this paper, we denote by $\mathbb{F}_q$ a finite field with $q = p^m$ for any prime p and positive integer m, and denote field elements by lower-case letters. For any $x \in \mathbb{F}_q$, let $-x$ be the additive inverse of x. We use $\oplus$ to denote plus over the finite field and $\langle \cdot, \cdot \rangle$ to denote the inner product. Let calligraphies (e.g., $\mathcal{I}$) be sets, and $|\mathcal{I}|$ denote the length of the set $\mathcal{I}$. Let $\bigcup_{k=1}^{n} \mathcal{I}_k$ (resp., $\bigcap_{k=1}^{n} \mathcal{I}_k$) be the union (resp., intersection) of sets $\mathcal{I}_1, \ldots, \mathcal{I}_n$. For any integers $i \leq j$, we use $[i\!:\!j]$ to denote the set of integers {i,...,j}. Let bold lower cases (e.g., $\boldsymbol{x}$) be the vectors over $\mathbb{F}_q^{|\boldsymbol{x}|}$, where $|\boldsymbol{x}|$ denotes the length of the vector, $\boldsymbol{x}[i]$ denotes the i^{th} element of vector $\boldsymbol{x}$, and $\boldsymbol{x}[i\!:\!j]$ denotes the vector made up of i^{th} to j^{th} elements of vector $\boldsymbol{x}$. Unless otherwise noted, we assume the vectors are row vectors, and the column vectors are denoted as $\boldsymbol{x}^{\mathrm{T}}$. We use $\sum \boldsymbol{x}$ to denote the summation of all the elements of vector $\boldsymbol{x}$.[4] Let $\boldsymbol{x} \odot \boldsymbol{y}$ be the element-wise multiplication between two vectors $\boldsymbol{x}$ and $\boldsymbol{y}$ of the same size. Let bold capital letters (e.g., $\boldsymbol{A}$) be the matrices in $\mathbb{F}_q^{r \times c}$ (or $r \times c$ matrix), for row and

[4] As the symbol sums all the elements, we omit writing bounds for the sake of brevity.

column counts being r and c respectively. Let $[\boldsymbol{a};\boldsymbol{b}]$ be the matrix that is the concatenation of vectors $\boldsymbol{a}$ and $\boldsymbol{b}$ in column, and $[\boldsymbol{a},\boldsymbol{b}]$ be the vector that is the concatenation of vectors $\boldsymbol{a}$ and $\boldsymbol{b}$ in row. We use $\sum \boldsymbol{A}$ to denotes the summation of all the rows of matrix $\boldsymbol{A}$, resulting in a vector. Similarly, $\sum \boldsymbol{A}^{\mathrm{T}}$ denotes the summation of all the rows of matrix $\boldsymbol{A}^{\mathrm{T}}$ that is the transpose of $\boldsymbol{A}$. A set of variables can be represented as $\{x_k\}_{k=1}^{n} \stackrel{\text{def}}{=} \{x_1,\ldots,x_n\}$, $\{x_k\}_{k\in\mathcal{K}} \stackrel{\text{def}}{=} \{x_k \text{ for } k\in\mathcal{K}\}$ or $x_{\mathcal{K}} \stackrel{\text{def}}{=} \{x_k \text{ for } k\in\mathcal{K}\}$, and these representations can be abused for an arbitrary set of vectors or matrices. Let $\tilde{O}(n) \stackrel{\text{def}}{=} O(nlog^c n)$ with c a constant.

2.2 Private Circuits

We view an arithmetic circuit (shorted as a circuit in the rest of this paper) C as a directed acyclic graph with gates and wires being vertices and edges respectively. The wires carry variables in $\mathbb{F}_q$ and the gates are elementary calculations over $\mathbb{F}_q$. A randomized circuit is a circuit augmented with random gates. Variables carried in the wires of a circuit C are called intermediate variables of C. The size (aka, the complexity) of a circuit is the number of gates. Particularly, a Boolean circuit is a circuit whose wires carry variables in $\mathbb{F}_2$ and the gates are elementary calculations over $\mathbb{F}_2$. A probe to a circuit is an intermediate variable whose value is assumed to be revealed to any adversary. For a circuit C with input $\boldsymbol{x}\in\mathbb{F}_q^n$, $\mathsf{C}(\boldsymbol{x})$ returns the output $\boldsymbol{y}\in\mathbb{F}_q^{n'}$, which we denote $\boldsymbol{y}=\mathsf{C}(\boldsymbol{x})$. And for a set $\mathcal{P}$ of probes, $\mathsf{C}_{\mathcal{P}}(\boldsymbol{x})$ returns the values of the probes by feeding $\boldsymbol{x}$ as the input of C. We call a set (or vector) of variables (say, $\boldsymbol{x}$) over $\mathbb{F}_q$ independently distributed of the other vector of variables (say, $\boldsymbol{y}$) if $\Pr(\boldsymbol{x}=\alpha,\boldsymbol{y}=\beta)=\Pr(\boldsymbol{x}=\alpha)\Pr(\boldsymbol{y}=\beta)$ for any value α of $\boldsymbol{x}$ and any value β of $\boldsymbol{y}$, where the probability is taken over the random coins used to generate these vectors.

Definition 1 (Private circuit compiler [ISW03])**.** A private circuit compiler for a circuit C with input in $\mathbb{F}_q^n$ and output in $\mathbb{F}_q^{n'}$ is defined by a triple $(\mathsf{I},\mathsf{T},\mathsf{O})$ where

- $\mathsf{I}:\mathbb{F}_q\to\mathbb{F}_q^{d+1}$, is an encoder that randomly maps each input $x\in\mathbb{F}_q$ to a sharing $\hat{\boldsymbol{x}}\in\mathbb{F}_q^{d+1}$ such that there exists a constant vector $\boldsymbol{\alpha}\in\mathbb{F}_q^{d+1}$ such that $\langle\hat{\boldsymbol{x}},\boldsymbol{\alpha}\rangle=x$. In this paper, the indices of shares in a sharing start from 0. Particularly, a sharing $\hat{\boldsymbol{x}}$ of x is called an additive sharing, if $\sum_{i=0}^{d}\hat{\boldsymbol{x}}[i]=x$.
- T is a circuit transformation whose input is circuit C, and output is a randomized circuit C' with n sharings as the input, and n' sharings as the output.
- $\mathsf{O}:\mathbb{F}_q^{d+1}\to\mathbb{F}_q$, is a decoder that maps each output sharing $\hat{\boldsymbol{z}}\in\mathbb{F}_q^{d+1}$ to the corresponding output $z\in\mathbb{F}_q$, i.e., $z\leftarrow\langle\hat{\boldsymbol{z}},\boldsymbol{\alpha}\rangle$ for a constant vector $\boldsymbol{\alpha}\in\mathbb{F}_q^{d+1}$.

We say that $(\mathsf{I},\mathsf{T},\mathsf{O})$ is a private circuit compiler and C' is a *d-private circuit* (or *d-probing secure*) if the following requirements hold:

- Correctness: for any input $\boldsymbol{x}\in\mathbb{F}_q^n$, $\mathsf{O}^{\circ}\big(\mathsf{C}'(\mathsf{I}^{\circ}(\boldsymbol{x}))\big)=\mathsf{C}\big(\boldsymbol{x}\big)$, where I° (resp., O°) is a canonical encoder (resp., decoder) that encodes (resp., decodes) each element of input secrets $\boldsymbol{x}$ (resp., each sharing of output sharings) by repeatedly calling I (resp., O).
- Privacy: for any input $\boldsymbol{x}\in\mathbb{F}_q^n$ and any set of probes $\mathcal{P}$ such that $|\mathcal{P}|\le d$, $\mathsf{C}'_{\mathcal{P}}\big(\mathsf{I}(\boldsymbol{x})\big)$ are independent of the input $\boldsymbol{x}$, where d is called the security order.

The index of an input or output share (say, $\hat{\boldsymbol{x}}_k[i]$) of gadget is defined as the index of the share in the sharing, i.e., the integer i. The position of an input or output share (say, $\hat{\boldsymbol{x}}_k[i]$) of gadget is defined as the pair of indices (i,k). We consider the circuit transformation T realized by the composition of gadgets. A *gadget* is a randomized circuit whose inputs and outputs are sharings. We say that a gadget G implements a function $f:\mathbb{F}_q^n\to\mathbb{F}_q^{n'}$, if and

only if $\mathsf{O}^{\circ}\Big(\mathsf{G}\big(\mathsf{I}^{\circ}(\boldsymbol{x})\big)\Big) = f(\boldsymbol{x})$ for any $\boldsymbol{x} \in \mathbb{F}_q^n$, where I° (resp., O°) encodes (resp., decodes) each input (resp., output). In the rest of the paper, we usually write a gadget in the form of $\mathsf{G}^{\mathcal{R}}_{\mathcal{H}}$, where G is the name, subscript $\mathcal{H}$ denotes the parameters related to the gadget's functionality, and superscript $\mathcal{R}$ denotes the randomness used in the gadget. We use the plural form of the name to denote multiple gadgets, e.g., $\mathsf{G}s$.

Gadget composition builds bigger circuits from a number of gadgets, by connecting the output wires of some gadgets to the input wires of the others. To cleanly pinpoint the "pattern" of a composition, we appeal to an acyclic graph C. That is, the resulting bigger circuit is obtained by replacing the vertices of C with the gadgets. In such a graph, the involved gadgets are called *sub-gadgets*, and the edges carry sharings. We call sub-gadgets containing input (resp., output) sharings of the composed gadget as input (resp., output) gadgets. Note that the composed gadget C' is a gadget, and thus a recursive composition of gadgets is also a gadget.

2.3 Different Types of Gadgets

As gadgets can be used as building blocks of private circuits, it is necessary to specify the types of gadgets that are required for protecting any cryptographic algorithms.

The first type is addition gadget that implements addition over finite fields. As the encoder is usually a homomorphism over addition, such a gadget can be correctly constructed by applying additions on the shares of the same index, which we will denote as *the trivial implementation of addition* and give in Gadget 1, named TrivAdd. Besides, we are also interested in a special type of function $f : \mathbb{F}_q \to \mathbb{F}_q$ such that $f(x) \oplus f(y) = f(x \oplus y) \oplus c$, where c is a constant in $\mathbb{F}_q$. This type of operations can be

1. a linear function taking with one input/output sharing $\oplus$ a constant, or
2. p^h power: $f(x) = x^{p^h}$ with h a positive integer. By Lucas' Theorem (all binomial coefficients $\binom{p^h}{i}$ for $i \in [1 : p^h - 1]$ are multiplies of p): $f(x \oplus y) = (x \oplus y)^{p^h} = x^{p^h} \oplus y^{p^h} = f(x) \oplus f(y)$.

We call this type of operation the *affine operation*, and present the trivial masked implementation in Gadget 2, named TrivAff.

Gadget 1 TrivAdd

Input: sharings $\hat{\boldsymbol{x}}, \hat{\boldsymbol{y}} \in \mathbb{F}_q^{d+1}$.
Output: sharing $\hat{\boldsymbol{z}} \in \mathbb{F}_q^{d+1}$.
The gadget ensures that: $\sum \hat{\boldsymbol{z}} = \sum \hat{\boldsymbol{x}} \oplus \sum \hat{\boldsymbol{y}}$.

```
for i := 0; i ≤ d; i++ do
    ẑ[i] := x̂[i] ⊕ ŷ[i]
end for
```

It becomes more difficult for (nonlinear) gadgets implementing nonlinear operations such as multiplication, since the encoder is usually not a homomorphism over nonlinear functions. To the best of our knowledge[5], the complexity of most multiplication gadgets in small fields is $\tilde{O}(d^2)$ and require $\tilde{O}(d^2)$ random bits. The last type is the refreshing gadget (a.k.a, the refreshing) that re-randomizes a sharing, which is usually needed to be placed between two gadgets to enable the composition. Additionally, in this paper, we

[5]It should be noted that the scheme proposed in [BBP+17] reduces either randomness usage or complexity to $O(d)$. But, despite the small instantiations for $d \leq 4$ [KR18], it requires large enough finite fields, e.g., the field size $q > d(d+1)(12d)^d$ [BBP+17, Theorem 5.4].

Gadget 2 $\mathsf{TrivAff}_{\mathsf{L},c}$

Input: sharings $\hat{\boldsymbol{x}} \in \mathbb{F}_q^{d+1}$.
Output: sharing $\hat{\boldsymbol{z}} \in \mathbb{F}_q^{d+1}$.
Parameters: function $\mathsf{L} : \mathbb{F}_q \to \mathbb{F}_q$ and constant $c \in \mathbb{F}_q$.
The gadget ensures that: $\sum \hat{\boldsymbol{z}} = \mathsf{L}\big(\sum \hat{\boldsymbol{x}}\big) \oplus c$.

1: **for** $i := 0; i \leq d; i{+}{+}$ **do**
2: $\quad \hat{\boldsymbol{z}}[i] := \mathsf{L}(\hat{\boldsymbol{x}}[i])$
3: **end for**
4: $\hat{\boldsymbol{z}}[0] := \hat{\boldsymbol{z}}[0] \oplus c$

provide a particular refreshing to transform sharings to sharings with common shares. See Section 4.1 for more details.

2.4 Maximum Distance Separable (MDS) Matrix

We recall the Maximum Distance Separable (MDS) Matrix below.

Definition 2. For an $\ell \times n$ matrix $\boldsymbol{A}$ over a finite field, let $\tilde{\boldsymbol{A}} \stackrel{\text{def}}{=} (\boldsymbol{A}; \boldsymbol{I}_n)$ be the matrix obtained by joining the $n \times n$ identity matrix $\boldsymbol{I}_n$ to $\boldsymbol{A}$. Then, $\boldsymbol{A}$ is MDS, if and only if every possible $n \times n$ submatrix of $\tilde{\boldsymbol{A}}$ is non-singular.

Then, we give a lemma showing a property related to an MDS matrix, which will be helpful in the proof of security for our proposed gadget in Section 4.

Lemma 1. *Let $\boldsymbol{A}$ be an $\ell \times n$ MDS matrix over $\mathbb{F}_q$, and let $\boldsymbol{B}$ be an $n \times n'$ uniformly distributed random matrix over $\mathbb{F}_q$. Then, any n rows of $[\boldsymbol{B}; \boldsymbol{AB}]$ are uniformly distributed.*

Proof. Let $\tilde{\boldsymbol{A}} \stackrel{\text{def}}{=} [\boldsymbol{I}; \boldsymbol{A}]$. By Definition 2, every possible $n \times n$ submatrix of $\tilde{\boldsymbol{A}}$ is non-singular. As $\boldsymbol{B}$ is uniformly distributed, any n rows of $[\boldsymbol{B}; \boldsymbol{AB}] = [\boldsymbol{I}; \boldsymbol{A}]\boldsymbol{B}$ are uniformly distributed, completing the proof. □

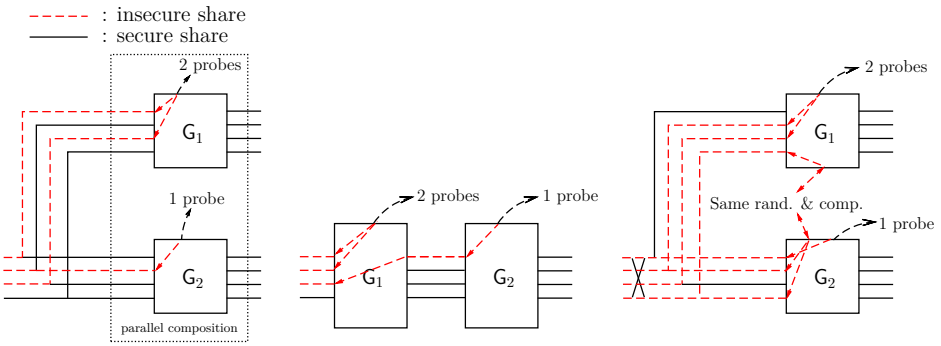
3 Composable Security Notions

Although the definition of private security nicely excludes side-channel attacks, it is not trivial to directly prove large circuits (such as the AES block cipher) to be private secure. The difficulty stems from enumerating the probes within the circuit, making the complexity of which increases exponentially with the circuit size. The natural solution is to use the composition method, so that one can focus on each individual gadget, and the global d-private security is ensured by composition. In this section, we start with the concept of 'classical' composable security notions, and discuss their limitation in presence of common shares. Then, we propose our new security notions and present how the composability works (even in the case with common shares) using the new notations.

3.1 Concept and Limitations

We recall the definition of simulatability introduced in [BBP+16]:

Definition 3 (Simulatability [BBP+16])**.** Let $\mathcal{P}$ be a set of probes of a gadget G with input shares $\mathcal{X}$. Let $\mathcal{S} \subseteq \mathcal{X}$ be a subset of input shares. A simulator is a randomized function Sim: $\mathbb{F}_q^{|\mathcal{S}|} \to \mathbb{F}_q^{|\mathcal{P}|}$. Probes $\mathcal{P}$ can be simulated with input shares $\mathcal{S}$ if and only if there exists a simulator Sim such that for any input shares $\mathcal{X}$, the distributions of $\mathsf{G}_{\mathcal{P}}(\mathcal{X})$ and $\mathsf{Sim}(\mathcal{S})$ are identical.



(a) Parallel composition (b) Serial composition (c) Probe propagation with randomness reuse

Figure 5: Examples of probe propagation.

Then, the probes of a gadget are partitioned as follows:

- Output probes: output variables.
- Internal probes: variables except for the output probes.

The concept of simulatability can significantly make security proof easier. Rather than exhaustively examining the entire private circuit, one can concentrate on analyzing every single gadget, and leave the rest to the probe propagation. For example, in Figure 5, G_1 and G_2 are two gadgets with one input/output sharing, and each gadget ensures that any t probes can be simulated with t input shares. Figure 5-(a) shows the *parallel composition* of the gadgets, where the input sharings are amalgamated. We can see that the t_1 and t_2 probes in G_1 and G_2 respectively can be simulated with $t_1 + t_2$ input shares.

Figure 5-(b) shows the *serial composition*, in which G_1's output links to G_2's input. t_2 (e.g., $t_2 = 1$) probes in G_2 can be simulated with t_2 input shares $\mathcal{S}$ of G_2, and $\mathcal{S}$ and t_1 (e.g., $t_1 = 2$) probes in G_1 can be simulated with $t_1 + t_2$ input shares of G_1, ensuring the security if $t_1 + t_2 \leq d$ (e.g., $d = 3$). In fact, it illustrates the first composable security notion called non-inference (NI) proposed by Barthe et al. [BBD+16], which we recall as follows together with a stronger variant.

Definition 4 ((Strong) Non-Inference: (S)NI [BBD+16])**.** A gadget is NI (resp., SNI), if and only if any t_{int} internal probes and t_{out} outputs probes such that $t_{int} + t_{out} \leq d$ can be simulated with at most $t_{int} + t_{out}$ (resp., t_{int}) shares of each input sharing.

Notably, it has been proven that the ISW multiplication gadget is SNI [BBD+16]. Then, we recall the composability of NI/SNI gadgets in Lemma 2. A straightforward corollary is that any composition of SNI gadgets is SNI.

Lemma 2 ([BBD+16])**.** *A composition of gadgets is NI, if every gadget is either NI or SNI and the following composition rule is fulfilled: each sharing is used at most once as input of a gadget that is not SNI. Moreover, a composition of gadgets is SNI, if it is NI and the output sharings are from SNI gadgets.*

Nevertheless, in presence of randomness and intermediate variables reuse across the gadgets, the probe propagation and the definition of composable security notions need to be paid more attention. For example, In Figure 5-(c), we consider the case that G_1 and G_2 are composed in parallel. The common randomness or intermediates across the gadgets may cause more insecure input shares: the t_1 and t_2 shares of G_1 and G_2 can be simulated with $t_1 + t_2$ shares of each of G_1 and G_2, and the amalgamation of $2 \times (t_1 + t_2)$ shares

(from input sharings of G_1 and G_2) covers all the shares of the amalgamated sharing, which is insecure.

It is non-trivial to solve such an issue. First of all, we need to consider the (composed) gadget with multiple input sharings (typically, the parallel composition of multiple gadgets), and investigate more carefully on the subset of input shares (i.e., insecure shares) to simulate the probes and subset of output shares to be simulated. Considering n sharings $\hat{\boldsymbol{x}}_{[1:\ell]}$, the naive definitions of the subsets can be as follows:

(1) At most t shares of every input sharing, i.e., $\{\hat{\boldsymbol{x}}_k[\mathcal{I}_k]\}_{k=1}^{\ell}$ with $t = \max_{k\in[1:\ell]} |\mathcal{I}_k|$. This is the type located at the input shares required to simulate the probes in NI/SNI.

(2) t shares of all input sharings, i.e., $t = \{\hat{\boldsymbol{x}}_k[\mathcal{I}_k]\}_{k=1}^{\ell}$ with $\sum_{k=1}^{\ell} |\mathcal{I}_k|$. This is the type located at the output shares as the probes that can be simulated in NI/SNI.

We can see that the positions of shares in input/output sharings can be defined differently. In this respect, to quantify a set of shares, we define the following shares-scale function.

Definition 5 (Shares-scale function)**.** A shares-scale function f takes a set of shares and returns an integer from 0 to $d+1$, such that the following properties are fulfilled.

1. Any shares set $\mathcal{S}$ contains
   - at most $f(\mathcal{S})$ shares of each sharing, and
   - at least $f(\mathcal{S})$ shares.
2. For any two shares sets $\mathcal{S}_1$ and $\mathcal{S}_2$ from the same set of sharings, $f(\mathcal{S}_1 \cup \mathcal{S}_2) \leq f(\mathcal{S}_1) + f(\mathcal{S}_2)$.

Based on Definition 5, we can rephrase NI/SNI from Definition 4 to Definition 6, which can facilitate the understanding of our new security notion presented in the next sub-section.

Definition 6 ((Strong) Non-Inference, rephrased)**.** Let G be a gadget with input and output sharings $\hat{\boldsymbol{x}}_{[1:\ell_{in}]}$ and $\hat{\boldsymbol{y}}_{[1:\ell_{out}]}$ resp.. We define two shares-scale functions f_{in} and f_{out} as follows:

(1) For any set $\mathcal{S} \stackrel{\text{def}}{=} \bigcup_{k=1}^{\ell_{in}} \hat{\boldsymbol{x}}_k[\mathcal{I}_k]$ of input shares, $f_{in}(\mathcal{S}) = \max_{k\in[1:\ell_{in}]} |\mathcal{I}_k|$. That is, $\mathcal{S}$ contains at most $f_{in}(\mathcal{S})$ shares of each input sharing.

(2) For any set $\mathcal{O} \stackrel{\text{def}}{=} \bigcup_{k=1}^{\ell_{out}} \hat{\boldsymbol{y}}_k[\mathcal{J}_k]$ of output shares, $f_{out}(\mathcal{O}) = \sum_{k=1}^{\ell_{out}} |\mathcal{J}_k|$. That is, $\mathcal{O}$ contains $f_{out}(\mathcal{O})$ output shares.

Then, G is NI (resp., SNI), if for any internal probes $\mathcal{P}_{int}$ and output probes $\mathcal{O}$ with $|\mathcal{P}_{int}| + f_{out}(\mathcal{O}) \leq d$, there exists a set $\mathcal{S}$ of input sharings with $f_{in}(\mathcal{S}) = |\mathcal{P}_{int}| + f_{out}(\mathcal{O})$ (resp., $f_{in}(\mathcal{S}) = |\mathcal{P}_{int}|$), such that $\mathcal{P}_{int}$ and $\mathcal{O}$ can be simulated with $\mathcal{S}$.

3.2 New Security Notions

This sub-section presents our new composable security notion compatible with the reuse of randomness and intermediates across gadgets. Compared with the NI/SNI defined in Definition 6, the new notion has two additional properties to support randomness reuse:

1. The shares-scale functions for input shares (to simulated probes) and output shares (to be simulated) are the same. That is, $f_{in} = f_{out}$. This property makes the security notion more compatible with the gadget with multiple input and output sharings and thus enables the trivial composition. Note that, the composability of gadgets in Multiple-Inputs / Multiple-Outputs Strong Non-Inference (MIMO) [CS20] also benefits from this property.

2. The positions of internal probes $\mathcal{P}_{int}$ determines which input shares $\mathcal{S}$ to simulate both $\mathcal{P}_{int}$ and some output probes $\mathcal{O}$ (On the contrary, for the case of NI/SNI, positions of $\mathcal{S}$ are determined by both $\mathcal{P}_{int}$ and $\mathcal{O}$). It means that the positions of $\mathcal{S}$ can be determined by only the internal probes $\mathcal{P}_{int}$ but $\mathcal{S}$ can simulate both $\mathcal{P}_{int}$ and some output probes $\mathcal{O}$, which enables separating the simulation into two stages:

   (a) Determine the positions of $\mathcal{S}$ by internal probes $\mathcal{P}_{int}$.

   (b) Simulate the internal probes $\mathcal{P}_{int}$ and some output probes $\mathcal{O}$.

   This property is the key concept allowing the deduction from parallel composition to any composition.

Then, based on the above intuition, we can give our new composable security notion in Definition 7.

Definition 7 (Randomness Reusable Non-Inference with a shares-scale function f: RNI-f)**.** Let G be a gadget with input and output sharings $\mathcal{X}$ and $\mathcal{Y}$ resp.. G is RNI-f, if and only if f is a shares-scale function and for any internal probes $\mathcal{P}_{int}$, there exists a shares set $\mathcal{S} \subset \mathcal{X}$ of input sharings with $f(\mathcal{S}) = |\mathcal{P}_{int}|$, such that $\mathcal{P}_{int}$ and any output probes $\mathcal{O} \subset \mathcal{Y}$ with $f(\mathcal{O}) + |\mathcal{P}_{int}| \leq d$ can be simulated with $\mathcal{S}$.

Note that, the definition of RNI-f is explicitly associated with a shares-scale function f. That is, for example, when we say two gadgets are both RNI-f, it means that they are RNI-f with the same shares-scale function f. As the goal of composable security notion is to prove the probing security, we give Lemma 3 to bridge RNI-f to the probing security.

Lemma 3 (RNI-f implies probing security)**.** *An RNI-f gadget is d-probing secure if any d input shares are independently distributed of the secret input.*

Proof. Let $\mathcal{P} = \mathcal{P}_{int} \cup \mathcal{O}$ be a set of probes with $\mathcal{P}_{int}$ the internal probes and $\mathcal{O}$ the output probes such that $|\mathcal{P}_{int}| + |\mathcal{O}| \leq d$. By the definition of the shares-scale function, we have $f(\mathcal{O}) \leq |\mathcal{O}|$, and thus $|\mathcal{P}_{int}| + f(\mathcal{O}) \leq |\mathcal{P}_{int}| + |\mathcal{O}| \leq d$.

By the definition of RNI-f, there exists a set of shares $\mathcal{S}$ with $f(\mathcal{S}) \leq |\mathcal{P}_{int}|$, such that $\mathcal{P}_{int}$ and $\mathcal{O}$ can be simulated with $\mathcal{S}$. Also, by the definition of the shares-scale function, $\mathcal{S}$ contains at most $f(\mathcal{S}) \leq |\mathcal{P}_{int}| \leq d$ of each input sharings.

If any d input shares are independently distributed of the secret input, then any d input shares of each input sharings should be also independently distributed of the secret input, hence $\mathcal{S}$ is independently distributed of the secret input. Then, $\mathcal{P}$ is independently distributed of the secret input, completing the proof. □

3.3 Main Theorem on Composition: From Parallel to General

In this sub-section, we present our main theorem that bridges the parallel composition to general compositions of gadgets.

We consider the composed gadget G a bipartite graph with gadgets as vertexes. That is, the sub-gadgets in G can be divided into two disjoint sets such that every connection of gadgets crosses the two sets. We show in Theorem 1 that if randomness/intermediate reuse is limited within each set and the parallel composition of gadgets in each set is RNI-f, then the composed gadget G is RNI.

Theorem 1. *Let* $\mathcal{G}_0 \stackrel{def}{=} \{\mathsf{G}_{0,1}, \ldots, \mathsf{G}_{0,\ell_0}\}$ *and* $\mathcal{G}_1 \stackrel{def}{=} \{\mathsf{G}_{1,1}, \ldots, \mathsf{G}_{1,\ell_1}\}$ *be two sets of gadgets reusing some randomness and intermediate variables. Any composition of the gadgets is RNI-*f *if the following conditions are fulfilled.*

1. *Each input sharing of* $\mathcal{G}_0$ *(resp.,* $\mathcal{G}_1$*) links to either an output sharing of* $\mathcal{G}_1$ *(resp.,* $\mathcal{G}_0$*) or an input sharing of the composed gadget.*
2. *Randomness used in* $\mathcal{G}_0$ *is independent of that in* $\mathcal{G}_1$*.*
3. *The parallel composition of the gadgets in* $\mathcal{G}_0$ *(resp.,* $\mathcal{G}_1$*) is RNI-*f*.*

Proof sketch. Let G be a composed gadget with input and output shares $\mathcal{X}$ and $\mathcal{Y}$ resp.. Let internal probes be $\mathcal{P}_{int}$, the proof is to build a simulator Sim and shares $\mathcal{S} \subset \mathcal{X}$, such that Sim can simulate $\mathcal{P}_{int}$ and any set $\mathcal{O} \subset \mathcal{Y}$ if $f(\mathcal{O}) + f(\mathcal{S}) \leq d$.

For $i \in \{1, 2\}$, let G_i be the parallel composition of the gadgets in $\mathcal{G}_i$, we have G_i is RNI-f. Let $\mathcal{P}_i$ be internal probes in G_i, i.e., $\mathcal{P}_i \stackrel{\text{def}}{=} \mathcal{P}_{int} \cap \mathcal{V}_i$ with $\mathcal{V}_i$ all internal probes of G_i. There exists a set $\mathcal{S}_i$ of input shares of G_i with $f(\mathcal{S}_i) \leq |\mathcal{P}_i|$, such that $\mathcal{P}_i$ and any output shares $\mathcal{O}_i$ of G_i can be simulated with $\mathcal{S}_i$, if $f(\mathcal{O}) + |\mathcal{P}_i| \leq d$.

We then construct the shares $\mathcal{S} = (\mathcal{S}_0 \cup \mathcal{S}_1) \cap \mathcal{X}$. And, the simulator Sim can be constructed by mingling the simulators for G_1 and G_2. This can be done because each input sharing of $\mathcal{G}_0$ (resp., $\mathcal{G}_1$) links to either an output sharing of $\mathcal{G}_1$ (resp., $\mathcal{G}_0$) or an input sharing of the composed gadget. Then, intuitively and informally, we can set $\mathcal{O}_i = \mathcal{S}_{1-i}$, and thus, by the property of RNI-f, the positions of $\mathcal{S}_i$ is only determined by the internal probes $\mathcal{P}_i$ but $\mathcal{S}_i$ can simulate both $\mathcal{P}_i$ and output shares $\mathcal{O}_i$, hence $\mathcal{P}_i$ and $\mathcal{S}_{1-i}$ can be simulated with $\mathcal{S}_i$. For strict proof, the simulator Sim should be constructed by simulating the small gadgets one by one from input to output. It requires the existence of a simulator (i.e., $\hat{\mathsf{Sim}}_i$) simulating gadgets one by one for the parallel composition, which we give in Lemma 5.

We give the detailed proof in Section 3.3.1, and show an example in Section 3.3.2. □

3.3.1 Proof of Theorem 1

Before running into the proof of Theorem 1, we first give Lemmas 4 and 5 to replenish the definition of the simulatability, where $\mathsf{G}_{\mathcal{P}}(\mathcal{S})$ returns probes $\mathcal{P}$ by feeding $\mathcal{S}$ and leaving $\bar{\mathcal{S}} \stackrel{\text{def}}{=} \mathcal{X}/\mathcal{S}$ as any values.

Lemma 4. *For a gadget* G *with input shares* $\mathcal{X}$*, if probes* $\mathcal{P}$ *can be simulated by the input shares* $\mathcal{S} \subset \mathcal{X}$*, then for any input shares* $\mathcal{X}$*, the distributions of* $\mathsf{G}_{\mathcal{P}}(\mathcal{X})$ *and* $\mathsf{G}_{\mathcal{P}}(\mathcal{S})$ *are identical. That is,* $\mathsf{G}_{\mathcal{P}}(\mathcal{X})$ *is independently distributed of* $\bar{\mathcal{S}} \stackrel{def}{=} \mathcal{X}/\mathcal{S}$*.*

Proof. Assume that $\mathsf{G}_{\mathcal{P}}(\mathcal{X})$ depends on $\bar{\mathcal{S}} \stackrel{\text{def}}{=} \mathcal{X}/\mathcal{S}$, then there does not exist any simulator Sim that can simulate the probes $\mathcal{P}$ without knowing $\bar{\mathcal{S}}$, contradicting to the existence of the simulator Sim that can simulate $\mathcal{P}$ with $\mathcal{S}$. □

Lemma 5. *Let* G *be the parallel composition of gadgets* $\mathsf{G}_1, \ldots, \mathsf{G}_\ell$*, and let* $\mathcal{P} \stackrel{def}{=} \bigcup_{k=1}^{\ell} \mathcal{P}_k$ *be the probes in* G *with* $\mathcal{P}_k$ *the probes in* G_k*. If* $\mathcal{P}$ *can be simulated by the input shares* $\mathcal{S} \stackrel{def}{=} \bigcup_{k=1}^{\ell} \mathcal{S}_k$ *of* G *with* $\mathcal{S}_k$ *a set of* G_k*'s input shares, then there exists a simulator* $\hat{\mathsf{Sim}}$ *to simulate* $\mathcal{P}$ *by evaluating* $\mathsf{G}_1, \ldots, \mathsf{G}_\ell$ *as follows:*

- *It first samples all randomness* $\boldsymbol{\mathcal{R}}$ *used in* G*.*
- *Then, for each* $k \in [1 : \ell]$*, it evaluates gadget* G_k *and returns its probes* $\mathcal{P}_k$ *with* $\mathcal{S}_k$ *and* $\boldsymbol{\mathcal{R}}$*, leaving the other input shares to be any values.*

Proof. Let $\mathcal{X}$ be all the input shares of the composed gadget, and let $\bar{\mathcal{S}} \stackrel{\text{def}}{=} \mathcal{X}/\mathcal{S}$. By the instruction of the simulator $\hat{\mathsf{Sim}}$, $\mathsf{G}_{\mathcal{P}}(\mathcal{S} \cup \bar{\mathcal{S}})$ behaves identically to $\hat{\mathsf{Sim}}(\mathcal{S})$, except for the input shares $\bar{\mathcal{S}}$. By Lemma 4, $\mathsf{G}_{\mathcal{P}}(\mathcal{X}) = \mathsf{G}_{\mathcal{P}}(\mathcal{S} \cup \bar{\mathcal{S}})$ is independently distributed of $\bar{\mathcal{S}}$. Thus, the distributions of $\mathsf{G}_{\mathcal{P}}(\mathcal{X})$ and $\hat{\mathsf{Sim}}_{\mathcal{P}}(\mathcal{S})$ are identical, completing the proof. □

Then, we give the proof of Theorem 1 as follows.

Proof. Let the composed gadget be G with input and output shares $\mathcal{X}$ and $\mathcal{Y}$ resp.. For $i \in \{0,1\}$ and $k \in [1:\ell]$, let $\mathcal{X}_{i,k}$ and $\mathcal{Y}_{i,k}$ be the input and output shares of $\mathsf{G}_{i,k}$, $\mathcal{X}_i \stackrel{\text{def}}{=} \bigcup_{k=1}^{\ell_i} \mathcal{X}_{i,k}$ and $\mathcal{Y}_i \stackrel{\text{def}}{=} \bigcup_{k=1}^{\ell_i} \mathcal{Y}_{i,k}$. We aim at proving that for any internal probes $\mathcal{P}_{int}$ with $|\mathcal{P}_{int}| \le d$, there exists a simulator Sim and a set $\mathcal{S} \subset \mathcal{X}$, such that Sim can simulate $\mathcal{P}_{int}$ and any set $\mathcal{O} \subset \mathcal{Y}$, if $f(\mathcal{O}) + |\mathcal{P}_{int}| \le d$. In other words, our goal is to build such a simulator Sim and input shares $\mathcal{S}$ for internal probes $\mathcal{P}_{int}$.

For $i \in \{0,1\}$, let the parallel composition of the gadgets in $\mathcal{G}_i$ be G_i, let $\mathcal{P}_i = \mathcal{P}_{int} \cap \mathcal{V}_i$ be the internal probes in gadgets of $\mathcal{G}_i$, where $\mathcal{V}_i$ consists of all internal variables in G_i. By Lemma 5 and the definition of RNI-f, there exist shares $\mathcal{S}_i \subset \mathcal{X}_i$, such that $\mathcal{P}_i$ and any set $\mathcal{O}_i \subset \mathcal{Y}_i$ with $f(\mathcal{O}_i) + |\mathcal{P}_i| \le d$ can be simulated with a simulator $\hat{\mathsf{Sim}}_i$ that first samples the randomness and then evaluates gadgets in $\{G_{i,1}, \ldots, G_{i,\ell_i}\}$ one by one.

We construct the shares $\mathcal{S} = (\mathcal{S}_0 \cup \mathcal{S}_1) \cap \mathcal{X}$, and we have $f(\mathcal{S}) = f((\mathcal{S}_0 \cup \mathcal{S}_1) \cap \mathcal{X}) \le f(\mathcal{S}_0 \cup \mathcal{S}_1) f(\mathcal{S}_0) \cup f(\mathcal{S}_1) \le |\mathcal{P}_{int}|$. For any gadget $\mathsf{G}_{i,k}$, let $\mathcal{S}_{i,k} \stackrel{\text{def}}{=} \mathcal{S} \cap \mathcal{X}_{i,k}$, and let $\mathcal{P}_{i,k} \stackrel{\text{def}}{=} \mathcal{P}_{int} \cap \mathcal{V}_{i,k}$ be the internal probes in $\mathsf{G}_{i,k}$, where $\mathcal{V}_{i,k}$ consists of all internal variables in $\mathsf{G}_{i,k}$. The simulator Sim can be constructed by mingling $\hat{\mathsf{Sim}}_0$ and $\hat{\mathsf{Sim}}_1$ as follows.

1. Mark all gadgets as un-simulated.
2. Sample all the randomness $\boldsymbol{R}$.
3. Find an un-simulated gadget (say, $\mathsf{G}_{i,k}$) such that at least one of the following conditions is fulfilled:
   (a) each share in $\mathcal{S}_{i,k}$ is either an input share of G or has been simulated;
   (b) $\mathcal{S}_{i,k} = \emptyset$.

   Note that, a gadget $\mathsf{G}_{i,k}$ fulfilling at least one of the above conditions always exists, since we can find such a gadget by the following steps:
   (a) Randomly select a gadget (say, $\mathsf{G}_{i,k}$).
   (b) If $\mathsf{G}_{i,k}$ fulfills one of the above conditions, then the search terminates.
   (c) If not, there exists one sharing (say, $\hat{\boldsymbol{x}}_{i,k}$) in $\mathcal{S}_{i,k}$ linked to an un-simulated gadget (say, $\mathsf{G}_{1-i,k'}$). Replace (i,k) by $(1-i,k')$, and go to step (b).

   As a circuit is a directed acyclic graph, a gadget fulfilling one of the above conditions can be found in at most $\ell_0 + \ell_1$ steps.
4. Evaluate $\mathsf{G}_{i,k}$ and calculate $\mathcal{P}_{i,k}$ and $\mathcal{O}_{i,k} \stackrel{\text{def}}{=} (\mathcal{S}_{1-i} \cap \mathcal{Y}_{i,k}) \cup (\mathcal{O} \cap \mathcal{Y}_{i,k})$ by $\boldsymbol{R}$ and $\mathcal{S}_{i,k}$, which is a subroutine of $\hat{\mathsf{Sim}}_i$. We can see that

$$
\begin{aligned}
|\bigcup_{k=1}^{\ell_i} \mathcal{P}_{i,k}| + f(\bigcup_{k=1}^{\ell_i} (\mathcal{O}_{i,k})) &\le |\bigcup_{k=1}^{\ell_i} \mathcal{P}_{i,k}| + f(\bigcup_{k=1}^{\ell_i} (\mathcal{S}_{1-i} \cap \mathcal{Y}_{i,k})) + f(\bigcup_{k=1}^{\ell_i} (\mathcal{O} \cap \mathcal{Y}_{i,k})) \\
&\le |\bigcup_{k=1}^{\ell_i} \mathcal{P}_{i,k}| + f(\mathcal{S}_{1-i}) + f(\mathcal{O}) \\
&\le |\mathcal{P}_i| + |\mathcal{P}_{1-i}| + f(\mathcal{O}) \le d \ ,
\end{aligned}
$$

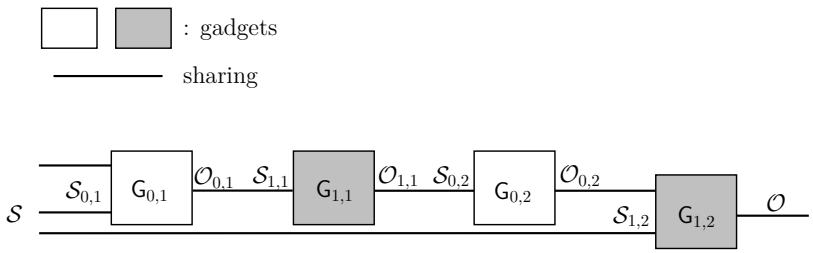


Figure 6: An example illustrating the proof of Theorem 3.3.1

where the first "$\leq$" is based on the definition of shares-scale function f. Therefore, all the probes in $\left\{\mathcal{P}_{0,k} \cup \mathcal{O}_{0,k}\right\}_{k=1}^{\ell_0}$ and $\left\{\mathcal{P}_{1,k} \cup \mathcal{O}_{1,k}\right\}_{k=1}^{\ell_1}$ can be simulated from the evaluation above. This means that the $\mathcal{P}_{i,k}$ and $\mathcal{O}_{i,k}$ can be simulated even if the other gadgets are all simulated.

5. Mark $\mathsf{G}_{i,k}$ as simulated. Repeat steps 3-5 until all the gadgets are evaluated, which means all probes are simulated.

□

3.3.2 An Example Illustrating the Proof of Theorem 1 in Section 3.3.1

We illustrate the proof of Theorem 1 by showing the RNI-f security of the composed gadget depicted in Figure 6. We aim at proving that, the composed gadget is RNI-f, if the parallel composition of $\mathsf{G}_{0,1}$ and $\mathsf{G}_{0,2}$ is RNI-f and the parallel composition of $\mathsf{G}_{1,1}$ and $\mathsf{G}_{1,2}$ is also RNI-f.

Let $\mathcal{X}$ be the input shares of composed gadget, and let $\mathcal{P}$ be a set of internal probes of the composed gadget. For $i \in \{0,1\}$ and $k \in \{1,2\}$, let $\mathcal{P}_{i,k}$ be the internal probes of $\mathsf{G}_{i,k}$. Note that, the probes to the output shares of $\mathsf{G}_{0,1}, \mathsf{G}_{0,2}, \mathsf{G}_{1,1}$ can be regarded as the internal probes of $\mathsf{G}_{1,1}, \mathsf{G}_{0,2}, \mathsf{G}_{1,2}$ resp.. By the definition of RNI-f, we can construct the positions of input shares $\mathcal{S}_{i,k}$ of each gadget based on the internal probes $\mathcal{P}_{i,k}$, such that $\mathcal{P}_{i,k}$ and any output probes $\mathcal{O}_{i,k}$ can be simulated, if $f(\mathcal{O}_{i,k}) + |\mathcal{P}_{i,k}| \leq d$. Then, we construct $\mathcal{S} \stackrel{\text{def}}{=} \mathcal{X} \cap (\mathcal{S}_{0,1} \cup \mathcal{S}_{1,2})$. For any set of output shares $\mathcal{O}$ with $f(\mathcal{O}) + |\mathcal{P}| \leq d$, the simulation can be done gadget-by-gadget as follows.

$\mathsf{G}_{0,1}$: We have $\mathcal{O}_{0,1} = \mathcal{S}_{1,1}$ and $f(\mathcal{S}_{1,1}) \leq |\mathcal{P}_{1,1}|$, then $|\mathcal{P}_{0,1}| + f(\mathcal{O}_{0,0}) \leq |\mathcal{P}_{0,1}| + |\mathcal{P}_{1,1}| \leq d$. Thus, $\mathcal{P}_{0,1} \cup \mathcal{S}_{1,1} = \mathcal{P}_{0,1} \cup \mathcal{O}_{0,1}$ can be simulated with $\mathcal{S}_{0,1}$.

$\mathsf{G}_{1,1}$: We have $\mathcal{O}_{1,1} = \mathcal{S}_{0,2}$ and $f(\mathcal{S}_{0,2}) \leq |\mathcal{P}_{0,2}|$, then $|\mathcal{P}_{1,1}| + f(\mathcal{O}_{1,1}) \leq |\mathcal{P}_{1,1}| + |\mathcal{P}_{0,2}| \leq d$. Thus, $\mathcal{P}_{1,1} \cup \mathcal{S}_{0,2} = \mathcal{P}_{1,1} \cup \mathcal{O}_{0,2}$ can be simulated with $\mathcal{S}_{1,1}$.

$\mathsf{G}_{0,2}$: We have $\mathcal{O}_{0,2} = \mathcal{S}_{1,2} \cap \mathcal{Y}_{0,2}$ and $f(\mathcal{S}_{1,2}) \leq |\mathcal{P}_{1,2}|$. By the definition of shares-scale function, we have $f(\mathcal{O}_{0,2}) \leq f(\mathcal{S}_{1,2})$ since $\mathcal{O}_{0,2} \subseteq \mathcal{S}_{1,2}$. Then, $|\mathcal{P}_{0,2}| + f(\mathcal{O}_{0,2}) + |\mathcal{P}_{0,1}| + f(\mathcal{O}_{0,1}) \leq d$. Thus, $\mathcal{P}_{0,2} \cup \mathcal{O}_{0,2} = \mathcal{P}_{0,2} \cup (\mathcal{Y}_{0,2} \cap \mathcal{S}_{1,2})$ can be simulated with $\mathcal{S}_{0,2}$, even after the simulation of $\mathsf{G}_{0,1}$, where $\mathcal{Y}_{0,2}$ is the set of output shares of $\mathsf{G}_{0,2}$. Now, $\mathcal{S}_{1,2} = (\mathcal{S}_{1,2} \cap \mathcal{Y}_{0,2}) \cup (\mathcal{S}_{1,2} \cap \mathcal{S})$ has been simulated.

$\mathsf{G}_{1,2}$: We have $|\mathcal{P}_{1,2}| + f(\mathcal{O}) + |\mathcal{P}_{1,1}| + f(\mathcal{O}_{1,1}) \leq d$. Thus, $\mathcal{P}_{1,2} \cup \mathcal{O}$ can be simulated with $\mathcal{S}_{1,2}$, even after the simulation of $\mathsf{G}_{1,1}$.

At last, by definition of shares-scale function, $f(\mathcal{S}) \leq f(\mathcal{S}_{0,1}) + f(\mathcal{S}_{1,2}) = |\mathcal{P}_{0,1}| + |\mathcal{P}_{1,2}|$. As $|\mathcal{P}_{0,1} \cap \mathcal{P}_{1,2}| = 0$ and $\mathcal{P}_{0,1} \cap \mathcal{P}_{1,2} \subseteq \mathcal{P}$, we have $f(\mathcal{S}) \leq |\mathcal{P}|$. Therefore, $\mathcal{P}$ and any output shares $\mathcal{O}$ with $f(\mathcal{O}) + |\mathcal{P}| \leq d$ can be simulated with $\mathcal{S}$, completing the proof.

3.4 Other Composition Rules

In this sub-section, for the generality and practicality of the composition, we give two more composition rules regarding the RNI-f gadgets.

Although Theorem 1 bridges the security of parallel composition to general compositions, more composition rules are still required when the parallel composition of gadgets is not RNI-f. For completeness, we provide in Lemma 6 that RNI-f implies SNI for any shares-scale function f.

Lemma 6. *If a gadget is RNI-f, it is SNI.*

Proof. By definition, RNI-f implies that any internal probes $\mathcal{P}_{int}$ and output probes $\mathcal{O}$ with $f(\mathcal{O})+|\mathcal{P}_{int}| \leq d$ can be simulated with some input shares $\mathcal{S}$, such that $f(\mathcal{S}) = |\mathcal{P}_{int}|$. By the definition of shares-scale function, we have $|\mathcal{O}| \geq d - |\mathcal{P}_{int}|$ and $\mathcal{S}$ contains at most $|\mathcal{P}_{int}|$ share of each input sharing, completing the proof. □

Then, we provide in Lemma 7 for the composition of RNI-f gadgets with independent randomness.

Lemma 7. *Any composition of RNI-f gadgets with independent randomness is RNI-f.*

Proof. By Definition 5, for any two shares' sets $\mathcal{S}_1$ and $\mathcal{S}_2$ from the some sharings, we have $f(\mathcal{S}_1 \cup \mathcal{S}_2) \leq f(\mathcal{S}_1) + f(\mathcal{S}_2)$. This gives that the parallel composition of RNI-f gadgets with independent randomness is RNI-f. Then, by Theorem 1, any composition of RNI-f gadgets with independent randomness is RNI-f. □

The above composition rules only consider RNI-f gadgets. But, obtaining RNI-f is non-trivial. For instance, we can easily verify that neither trivial addition nor affine gadgets (i.e., TrivAdd and TrivAff) presented in Section 2.3 are RNI-f. At the same time, as we will show in Section 4, the randomness and computational complexities of the RNI-f addition we can obtain is $\tilde{O}(d^2)$, which is generally more costly than trivial implementations.

Fortunately, we show in Lemma 8 that the direct composition of a trivial affine gadget and an arbitrary RNI-f gadget is RNI-f. Also note that, this composition rule only works for the trivial affine gadget (which has only one input/output sharing), and the direct composition of an RNI-f gadget and a trivial addition gadget (with two input sharings) may not be secure in RNI-f.

Lemma 8. *Any composition of an arbitrary RNI-f gadget and an arbitrary* TrivAff *is RNI-f.*

Proof. By the instruction of TrivAff, linear operations are evaluated over each share separately. It conveys that each intermediate variable of TrivAff is a function of the input or output share with the same index. That is, for an intermediate variable v corresponding to the input share with index i, we have $v = g_1(\hat{\boldsymbol{x}}[i]) = g_2(\hat{\boldsymbol{z}}[i])$, where $\hat{\boldsymbol{x}}$ and $\hat{\boldsymbol{z}}$ are input and output sharings respectively, and $g_1, g_2 : \mathbb{F}_q \to \mathbb{F}_q$. Thus, from the probing security point of view, there is no difference between an intermediate variable v and the corresponding input share $\hat{\boldsymbol{x}}[i]$ or output share $\hat{\boldsymbol{z}}[i]$. Finally, we can replace each intermediate variable in TrivAff by the corresponding input or output share in the simulation of probes, completing the proof. □ □

3.5 Relations of RNI-f, SNI, NI, PINI and Probing Security

In Figure 7, we illustrate the relations of our new notion RNI-f to the well-known SNI, NI, PINI and probing security. While the composable security notions SNI, NI, PINI and RNI-f imply probing security if any d input shares are independent of the secret, they support the composition of small gadgets. SNI is stronger than NI but requires

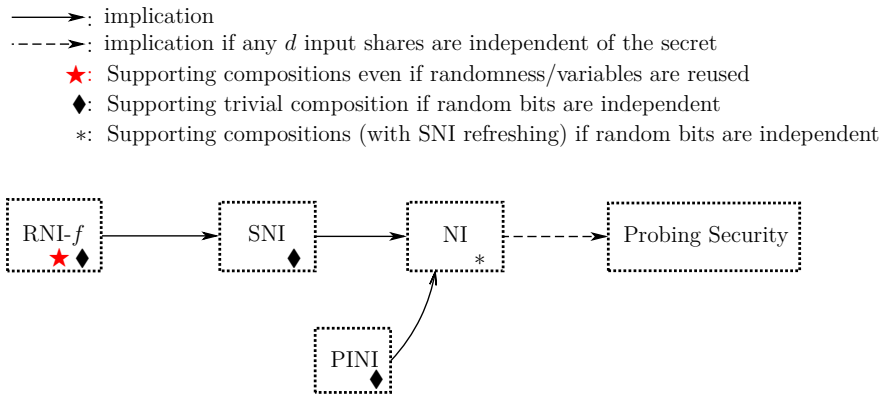


Figure 7: Relations of different security notions

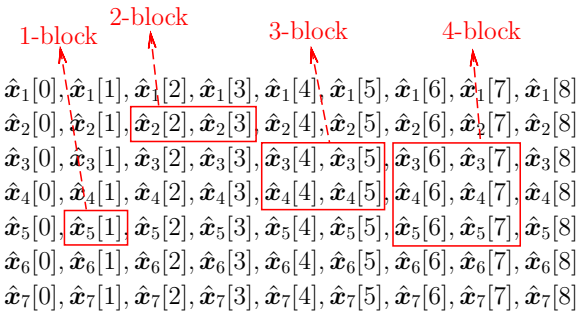


Figure 8: An example to illustrate the blocks in 7 sharings with $d = 8$. Each row contains shares of a distinct sharing.

more randomness for a gadget to achieve. SNI gadgets support trivial composition, which conveys that any composition of SNI gadget is SNI. PINI is another security notion proposed by Cassiers et al. [CS19] supporting trivial composition. Compared with SNI, PINI requires less randomness for a gadget to achieve. SNI, NI and PINI suffice the use of independent randomness. That is, the random bits for different gadgets should be independently distributed, which prevents the reuse of randomness cross gadgets.

By Lemma 6, our proposed RNI-f is stronger than SNI, which conveys that it also supports trivial composition. More importantly, we have provided in Theorem 1 a composition rule that allows the randomness/variables reuse across different RNI-f gadgets. It should be noted that, the definitions of the security notions do not imply any randomness requirements, but all known implementations do.

3.6 Block-scale Function

In this sub-section, we introduce a specific shares-scale function called block-scale function, which lies in between the two shares-scale functions (i.e., f_{in} and f_{out} in Definition 6) of NI/SNI, and most importantly, complies with our new scheme presented in Section 4. The block-scale function is based on the definition of block that we give below.

Definition 8 (Block)**.** A block of sharings $\hat{\boldsymbol{x}}_{[1:\ell]}$ is a shares set $\hat{\boldsymbol{x}}_{\mathcal{K}}[\mathcal{I}]$, where $\mathcal{K} \subseteq [1:\ell]$ and $\mathcal{I} \subseteq [0:d]$. Moreover, $\hat{\boldsymbol{x}}_{\mathcal{K}}[\mathcal{I}]$ is a t-block if and only if $t = |\mathcal{K}| + |\mathcal{I}| - 1$.

A block can be regarded as a "rectangle" containing shares. We exemplify different blocks in Figure 8. Obviously, by definition, a 1-block is equivalent to a share. For a block

$\mathcal{S}$ and any integer $n \geq 1$, a set of nonempty blocks $\{\mathcal{S}_j\}_{j=1}^n$ is a *block-covering* of $\mathcal{S}$ if and only if $\mathcal{S} = \bigcup_{j=1}^n \mathcal{S}_j$. Then, Lemma 9 shows a property of block-covering.

Lemma 9 (Property of block-covering)**.** *Let $\mathcal{S}$ be a t-block, and let $\{\mathcal{S}_j\}_{j=1}^n$ be a block-covering of $\mathcal{S}$, where $\mathcal{S}_j$ is a t_j-block for $j \in [1:n]$. Then, we have $\sum_{j=1}^n t_j \geq t$.*

Proof. We only consider the case when $\bigcap_{j=1}^n \mathcal{S}_j = \emptyset$ and $n \geq 2$. It is because that, if $\bigcap_{j=1}^n \mathcal{S}_j \neq \emptyset$, there exists t'_j-block $\mathcal{S}'_j \subseteq \mathcal{S}_j$ for $j \in [1:n]$ such that $\bigcap_{j=1}^n \mathcal{S}'_j = \emptyset$, and thus $\sum_{j=1}^n t'_j \leq \sum_{j=1}^n t_j$, which conveys that we can consider $\{\mathcal{S}'_j\}_{j=1}^n$ instead of $\{\mathcal{S}_j\}_{j=1}^n$. And, if $n = 1$, then $\mathcal{S} = \mathcal{S}_1$ and thus $\sum_{j=1}^n t_j = t$.

Let $\mathcal{S} \stackrel{\text{def}}{=} \hat{\boldsymbol{x}}_{\mathcal{K}}[\mathcal{I}]$ with $\mathcal{K} \subseteq [1:\ell]$ and $\mathcal{I} \subseteq [0:d]$. We define the neighbor shares $\mathcal{N}_S$ of $\mathcal{S}$ by

$$\mathcal{N}_{\mathcal{S}} \stackrel{\text{def}}{=} \hat{\boldsymbol{x}}_{\max(\mathcal{K})}[\mathcal{I}] \cup \hat{\boldsymbol{x}}_{\min(\mathcal{K})}[\mathcal{I}] \cup \hat{\boldsymbol{x}}_{\mathcal{K}}[\max(\mathcal{I})] \cup \hat{\boldsymbol{x}}_{\mathcal{K}}[\min(\mathcal{I})],$$

where $\max(\cdot)$ and $\min(\cdot)$ return the maximum and minimum index resp.. In other words, if we regard $\mathcal{S}$ as a rectangle containing shares, the neighbor shares are positioned at the sides of the rectangle. We then call $\hat{\boldsymbol{x}}_{\max(\mathcal{K})}[\mathcal{I}]$, $\hat{\boldsymbol{x}}_{\min(\mathcal{K})}[\mathcal{I}]$ $\hat{\boldsymbol{x}}_{\mathcal{K}}[\max(\mathcal{I})]$ and $\hat{\boldsymbol{x}}_{\mathcal{K}}[\min(\mathcal{I})]$ sides of $\mathcal{S}$, and call $\hat{\boldsymbol{x}}_{\max(\mathcal{K})}[\max(\mathcal{I})]$,$\hat{\boldsymbol{x}}_{\max(\mathcal{K})}[\min(\mathcal{I})]$,$\hat{\boldsymbol{x}}_{\min(\mathcal{K})}[\max(\mathcal{I})]$ and $\hat{\boldsymbol{x}}_{\min(\mathcal{K})}[\min(\mathcal{I})]$ vertices of $\mathcal{S}$. By the definition of the block, we have $t = |\mathcal{I}| + |\mathcal{K}| = \frac{|\mathcal{N}_{\mathcal{S}}|}{2} + 1$. We also define the internal shares of $\mathcal{S}$ by $\mathcal{T}_{\mathcal{S}} \stackrel{\text{def}}{=} \mathcal{S}/\mathcal{N}_{\mathcal{S}}$.

For any $j \in [1:n]$, we define $\mathcal{N}'_{\mathcal{S}_j,\mathcal{S}} \stackrel{\text{def}}{=} \mathcal{N}_{\mathcal{S}_j} \cap \mathcal{T}_{\mathcal{S}}$, which is the intersection of $\mathcal{S}_j$'s neighbor shares and $\mathcal{S}$'s internal shares. That is, $\mathcal{N}'_{\mathcal{S}_j,\mathcal{S}}$ contains $\mathcal{S}_j$'s sides but contains no $\mathcal{S}$'s sides. As $\mathcal{S}_j \subset \mathcal{S}$, by geometric shape of a rectangle, $\mathcal{S}_j$ has at least one side that is not a side of $\mathcal{S}$, and thus we have $|\mathcal{N}'_{\mathcal{S}_j,\mathcal{S}}| > 0$ for any $j \in [1:n]$. As every share in $\mathcal{N}'_{\mathcal{S}_j,\mathcal{S}}$ is included in $\mathcal{S}_j$ and another block of $\mathcal{S}_1, \ldots, \mathcal{S}_n$, and every share in $\mathcal{N}_{\mathcal{S}}$ is included in $\mathcal{S}$ and one block of $\mathcal{S}_1, \ldots, \mathcal{S}_n$, we have

$$\sum_{j=1}^n (2t_j - 2) = \sum_{j=1}^n |\mathcal{N}_{\mathcal{S}_j}| = |\mathcal{N}_{\mathcal{S}}| + 2\sum_{j=1}^n |\mathcal{N}'_{\mathcal{S}_j,\mathcal{S}}| = 2t - 2 + 2\sum_{j=1}^n |\mathcal{N}'_{\mathcal{S}_j,\mathcal{S}}|.$$

Thus, we have $\sum_{j=1}^n t_j = t + n - 1 + \sum_{j=1}^n |\mathcal{N}'_{\mathcal{S}_j,\mathcal{S}}|$.

We then prove that there exist at most 2 blocks in $\mathcal{S}_1, \ldots, \mathcal{S}_n$ such that neighbor shares of each one involves only one internal share of $\mathcal{S}$, i.e., there exist at most two indices j in $[1:n]$ satisfying $|\mathcal{N}'_{\mathcal{S}_j,\mathcal{S}}| = 1$. Let $\mathcal{S}_j \stackrel{\text{def}}{=} \hat{\boldsymbol{x}}_{\mathcal{K}_j}[\mathcal{I}_j]$ for $j \in [1:n]$. Without loss of generality, suppose $|\mathcal{N}'_{\mathcal{S}_j,\mathcal{S}}| = 1$, then 3 sides of $\mathcal{S}_j$ constitute a subset of 3 different sides of $\mathcal{S}$. It conveys that 2 vertices of $\mathcal{S}$ are included in both $\mathcal{S}_j$ and $\mathcal{S}$. Hence, if there exists another block, say $\mathcal{S}_{j'}$, such that $|\mathcal{N}_{\mathcal{S}_{j'}} \cap \mathcal{T}_{\mathcal{S}}| = 1$, then another 2 vertices of $\mathcal{S}$ are included in both $\mathcal{S}_{j'}$ and $\mathcal{S}$. As $\mathcal{S}$ only has 4 vertices, there exist at most two indices j in $[1:n]$ such that $|\mathcal{N}'_{\mathcal{S}_j,\mathcal{S}}| = 1$, which conveys that $\sum_{j=1}^n |\mathcal{N}'_{\mathcal{S}_j,\mathcal{S}}| \geq 2 + 2(n-2)$.

Finally, we have

$$\sum_{j=1}^n t_j = t + n - 1 + \sum_{j=1}^n |\mathcal{N}'_{\mathcal{S}_j,\mathcal{S}}| \geq t + n - 1 + 2 + 2(n-2) = t + 2n - 3.$$

As we consider $n > 1$, $\sum_{j=1}^n t_j \geq t + 1 > t$. □

Then, we give the definition of the block-scale function as follows.

Definition 9 (The block-scale function)**.** The block-scale function f_b is a function that takes a shares set and returns a positive integer, such that one of the following conditions is fulfilled:

- If $\mathcal{S}$ is a t-block, then $t = f_b(\mathcal{S})$.

- If $\mathcal{S}$ is not a block, then $f_b(\mathcal{S}) = \min_{\{\mathcal{S}_j\}_{j=1}^n \in \boldsymbol{\mathcal{S}}} \big(\sum_{j=1}^n f_b(\mathcal{S}_j) \big)$, where $\boldsymbol{\mathcal{S}}$ is the set made up of all block-coverings of $\mathcal{S}$.

The definition defines a deterministic function f_b by minimizing the possible outputs. That is, when $\mathcal{S}$ is not a block, it returns the least value over all the block-coverings of $\mathcal{S}$; and when $\mathcal{S}$ is a t-block, then by Lemma 9, $f_b(\mathcal{S}) = t = \min_{\{\mathcal{S}_j\}_{j=1}^n \in \boldsymbol{\mathcal{S}}} \big(\sum_{j=1}^n f_b(\mathcal{S}_j) \big)$, where $\boldsymbol{\mathcal{S}}$ is the set made up of all block-coverings of $\mathcal{S}$ and the latter "=" holds by assigning $n = 1$. Moreover, the definition is a recursive one that naturally supports the union of multiple sets of shares, which we give in Lemma 10. Note that, Lemma 10 considers the union of multiple *sets* of shares, which is stronger than the union of a block-covering that are *blocks*.

Lemma 10 (The union of multiple sets of shares)**.** *Let f_b be a block-scale function, and let $\mathcal{S}_1, \ldots, \mathcal{S}_n$ be shares sets of the same sharings, we have $f_b(\bigcup_{j=1}^n \mathcal{S}_j) \le \sum_{j=1}^n f_b(\mathcal{S}_j)$.*

Proof. Let $\mathcal{S} \stackrel{\text{def}}{=} \bigcup_{j=1}^n \mathcal{S}_j$, If $\mathcal{S}$ is not a block, then by definition,

$$f_b(\mathcal{S}) = \min_{n'=1,2,\ldots} \Big(\min_{\mathcal{S}=\bigcup_{j=1}^{n'} \mathcal{S}'_j} \big(\sum_{j=1}^{n'} f_b(\mathcal{S}'_j) \big) \Big) \le \sum_{j=1}^{n} f_b(\mathcal{S}_j).$$

We then consider the case when $\mathcal{S}$ is a t-block. Assume $f_b(\mathcal{S}) > \sum_{j=1}^n f_b(\mathcal{S}_j)$, then there exist blocks $\mathcal{S}'_1, \ldots, \mathcal{S}'_{n'}$ such that $\mathcal{S} = \bigcup_{j=1}^n \mathcal{S}_j = \bigcup_{j=1}^{n'} \mathcal{S}'_j$ and $\sum_{j=1}^{n'} f_b(\mathcal{S}'_j) = \sum_{j=1}^n f_b(\mathcal{S}_j)$, and thus we have $f_b(\mathcal{S}) > \sum_{j=1}^{n'} f_b(\mathcal{S}'_j)$. Without loss of generality, suppose $\mathcal{S}'_j$ is a t'_j-block for $j \in [1 : n']$, we then have $t > \sum_{j=1}^{n'} t'_j$, contradicting to Lemma 9. □

Lemma 11 shows that the block-scale function is a specific shares-scale function.

Lemma 11. *The block-scale function f_b is a shares-scale function.*

Proof. Let f_b be the block-scale function. Let $\mathcal{S}_1$ and $\mathcal{S}_2$ be any two sets of shares from the same set of sharings, by Lemma 10, we have $f_b(\mathcal{S}_1 \cup \mathcal{S}_2) \le f_b(\mathcal{S}_1) + f_b(\mathcal{S}_2)$.

Then, let $\mathcal{S}$ be a set of shares, the following two properties are fulfilled.

- By Lemma 10, we have $f_b(\mathcal{S}) \le \sum_{j=1}^{|\mathcal{S}|} \big(f_b(p_j)\big) = |\mathcal{S}|$ with p_j the share of $\mathcal{S}$ for $j \in [1 : |\mathcal{S}|]$.
- By definition of the block-scale function, there exist a block-covering $\{\mathcal{S}_1, \ldots, \mathcal{S}_n\}$ of $\mathcal{S}$ such that $f_b(\mathcal{S}) = \sum_{j=1}^n f_b(\mathcal{S}_j)$. Let $\mathcal{S}_j \stackrel{\text{def}}{=} \hat{\boldsymbol{x}}_{\mathcal{K}_j}[\mathcal{I}_j]$ for $j \in [1 : n]$, by Lemma 10, we have $f_b(\mathcal{S}) = \sum_{j=1}^n f_b(\mathcal{S}_j) = \sum_{j=1}^n (|\mathcal{K}_j| + |\mathcal{I}_j| - 1) \ge \sum_{j=1}^n |\mathcal{I}_j|$. Thus, $\mathcal{S}$ contains at most $f(\mathcal{S})$ shares of each sharing.

Therefore, the block-scale function is a shares-scale function. □

We give a lemma showing a property of block-scale function, which will be helpful in the security proof of our proposed gadget in Section 4.

Lemma 12. *Let f_b be the block-scale function. For any shares $\mathcal{S} = \hat{\boldsymbol{x}}_{\mathcal{K}}[0] \cup \hat{\boldsymbol{x}}_{[1:\ell]}[\mathcal{I}]$ of sharings $\hat{\boldsymbol{x}}_{[1:\ell]}$ with $\mathcal{K} \subseteq [1 : \ell]$ and $\mathcal{I} \subseteq [1 : d]$, we have $f_b(\mathcal{S}) \ge |\mathcal{K}| + |\mathcal{I}|$.*

Proof. By the recursive definition of block-scale function, there exist blocks $\mathcal{S}_1, \ldots, \mathcal{S}_n$ of sharings $\hat{\boldsymbol{x}}_{[1:\ell]}$ such that $f_b(\mathcal{S}) = \sum_{j=1}^n f_b(\mathcal{S}_j)$. The sets $\mathcal{K}$ and $\mathcal{I}$ can be constructed from $\mathcal{S}_1, \ldots, \mathcal{S}_n$ as follows.

- Initiate $\mathcal{K}$ and $\mathcal{I}$ to be empty.

- For each block $\mathcal{S}_j = \hat{\boldsymbol{x}}_{\mathcal{K}_j}[\mathcal{I}_j]$ in $\mathcal{S}_1, \ldots, \mathcal{S}_n$, we put indices in $\mathcal{I}_j/\{0\}$ into $\mathcal{I}$. Then, if $0 \in \mathcal{I}_j$, we put indices in $\mathcal{K}_j$ into $\mathcal{K}$. Now, we have $\hat{\boldsymbol{x}}_{\mathcal{K}_j}[\mathcal{I}_j] \subset \hat{\boldsymbol{x}}_{\mathcal{K}_j}[0] \cup \hat{\boldsymbol{x}}_{[1:\ell]}[\mathcal{I}_k]$ and $|\mathcal{K}_j| + |\mathcal{I}_j/\{0\}| \leq f_b(\mathcal{S}_j)$.

Finally, we have $f_b(\mathcal{S}) = \sum_{j=1}^{n} f_b(\mathcal{S}_j) \geq \sum_{j=1}^{n}(|\mathcal{K}_j| + |\mathcal{I}_j/\{0\}|) \geq |\mathcal{K}| + |\mathcal{I}|$. □

At last, we give a lemma showing the equivalence between t shares and a shares set of sharings with common shares. This lemma will also be helpful in the security proof of our proposed gadget in Section 4.

Lemma 13. *Let $\hat{\boldsymbol{x}}_{[1:\ell]}$ be a set of sharings with common shares, i.e., $\hat{\boldsymbol{x}}_k[1:d] = \hat{\boldsymbol{x}}_{k'}[1:d] = \hat{\boldsymbol{x}}[1:d]$ for any $k, k' \in [0:\ell]$. Then, for any shares $\mathcal{S}$ of $\hat{\boldsymbol{x}}_{[1:\ell]}$, we have $f_b(\mathcal{S}) = |\mathcal{S}|$.*

Proof. We first prove $f_b(\mathcal{S}) \leq |\mathcal{S}|$ for any set $\mathcal{S}$ of shares. Let $\mathcal{S} \stackrel{\text{def}}{=} \{p_1, \ldots, p_t\}$. As one share is a 1-block, we have $f_b(\mathcal{S}) = f_b(\bigcup_{k=1}^{t}\{p_k\}) \leq \sum_{k=1}^{t} f_b(\{p_k\}) = |\mathcal{S}|$.

We then prove $f_b(\mathcal{S}) \geq |\mathcal{S}|$ for any set $\mathcal{S}$ of shares. The shares can be represented as $\mathcal{S} = \hat{\boldsymbol{x}}_{\mathcal{K}}[0] \cup \hat{\boldsymbol{x}}_{[1:\ell]}[\mathcal{I}]$ with $\mathcal{K} \subseteq [1:\ell]$ and $\mathcal{I} \subseteq [1:d]$. Obviously, we have $|\mathcal{K}| + |\mathcal{I}| \geq |\mathcal{S}|$. By Lemma 12, we have $f_b(\mathcal{S}) \geq |\mathcal{K}| + |\mathcal{I}| \geq |\mathcal{S}|$. □

4 New Masking Scheme

4.1 Constructions of New Gadgets

In this sub-section, we present our new refreshing, multiplication and addition gadgets as the building blocks of the masking scheme with common shares.

Gadget 3 is a refreshing gadget (named as RNIRefresh). A part of the output shares $\hat{\boldsymbol{z}}[1:d]$ is determined only by the random matrix $\boldsymbol{R}$, and thus the refreshing always returns the same shares $\hat{\boldsymbol{z}}[1:d]$ for any input shares and parameter $\boldsymbol{\alpha}$ as long as random matrix $\boldsymbol{R}$ is the same. Hence, RNIRefresh provides a transformation from a set of sharings to sharings with common shares, by refreshing the sharings using the same randomness $\boldsymbol{R}$ and different parameters.

Moreover, we highlight by underlining in red color the parts that are independent of the input sharing $\hat{\boldsymbol{x}}$. Considering that, in some cases, a part of the input $\hat{\boldsymbol{x}}[1:d]$ can be precomputed by the randomness of the former gadget (e.g., two RNIRefresh*es* are linked serially), we also highlight by underwaving in blue color the part that is independent of $\hat{\boldsymbol{x}}[0]$. Note that the highlighted part enables the precomputation-based design paradigm that we will discuss in detail in Section 4.3. We give the correctness and security proof of parallel composition in Lemma 14.

Gadget 3 $\mathsf{RNIRefresh}_{\alpha}^{\boldsymbol{R}}$

Input: sharings $\hat{\boldsymbol{x}} \in \mathbb{F}_q^{d+1}$.
Output: sharing $\hat{\boldsymbol{z}} \in \mathbb{F}_q^{d+1}$.
Parameter: $\boldsymbol{\alpha} \in \mathbb{F}_q^{d}$.
Randomness: matrix $\boldsymbol{R} \in \mathbb{F}_q^{d \times d}$.

The gadget ensures that: $\langle \hat{\boldsymbol{z}}, \boldsymbol{a} \rangle = \sum \hat{\boldsymbol{x}}$ with $\boldsymbol{a} \stackrel{\text{def}}{=} [1, \boldsymbol{\alpha}]$.

1: $\hat{\boldsymbol{z}}[1:d] := \sum \boldsymbol{R}^{\mathrm{T}}$
2: $\boldsymbol{r} := -\boldsymbol{\alpha}\boldsymbol{R}$
3: $t := \sum(\hat{\boldsymbol{x}}[1:d] \oplus \boldsymbol{r})$
4: $\hat{\boldsymbol{z}}[0] := \hat{\boldsymbol{x}}[0] \oplus t$

Lemma 14. *Let f_b be the block-scale function. Let* G *be a parallel composition of ℓ* RNIRefresh*es using parameters $\boldsymbol{\alpha}_1, \ldots, \boldsymbol{\alpha}_\ell$ respectively with the same random matrix $\boldsymbol{R}$. Then, we have:*

- ***Correctness***. *For any* $k \in [1:\ell]$, $\langle \hat{\boldsymbol{z}}_k, [1, \boldsymbol{\alpha}_k] \rangle = \sum \hat{\boldsymbol{x}}_k$, *and* $\hat{\boldsymbol{z}}_k[1\!:\!d]$ *is determined only by randomness.*
- ***Security***. G *is RNI-*f_b *if the matrix* $[\boldsymbol{\alpha}_1; \ldots; \boldsymbol{\alpha}_\ell]$ *is MDS.*

Proof of correctness property in Lemma 14. For any $k \in [1:\ell]$, we have:

$$\begin{aligned}
\hat{\boldsymbol{z}}_k[0] \oplus \langle \hat{\boldsymbol{z}}_k[1\!:\!d], \boldsymbol{\alpha}_k \rangle &= \hat{\boldsymbol{z}}_k[0] \oplus \Big(\sum \boldsymbol{R}^{\mathrm{T}} \Big) \boldsymbol{\alpha}_k^{\mathrm{T}} \\
&= \hat{\boldsymbol{z}}_k[0] \oplus \sum (\boldsymbol{\alpha}_k \boldsymbol{R}) \\
&= \hat{\boldsymbol{x}}_k[0] \oplus \sum \big(\hat{\boldsymbol{x}}_k[1\!:\!d] \oplus (-\boldsymbol{\alpha}_k \boldsymbol{R}) \big) \oplus \sum (\boldsymbol{\alpha}_k \boldsymbol{R}) \\
&= \hat{\boldsymbol{x}}_k[0] \oplus \sum \hat{\boldsymbol{x}}_k[1\!:\!d] \\
&= x_k \ .
\end{aligned}$$

And, as $\hat{\boldsymbol{z}}_k[1\!:\!d] = \sum \boldsymbol{R}^{\mathrm{T}}$, $\hat{\boldsymbol{z}}_k[1\!:\!d]$ is only determined by the randomness. □

Proof of security property in Lemma 14. We use subscript to differentiate the parameters and variables in different gadgets, e.g., let $\boldsymbol{\alpha}_k$ and $\hat{\boldsymbol{x}}_k$ be the parameter and input sharing of the k-th gadget. Particularly, as $\hat{\boldsymbol{z}}_k[1:d]$ is identical for any values k, the output shares of the parallel composed gadget can be represented as $\hat{\boldsymbol{z}}_{[1:\ell]}[0] \cup \hat{\boldsymbol{z}}[1:d]$. We define a matrix $\boldsymbol{A} \stackrel{\text{def}}{=} [\boldsymbol{\alpha}_1; \ldots; \boldsymbol{\alpha}_\ell]$. The probes are partitioned into different subsets based on the types of variables:

- Internal probes $\mathcal{P}_{int}$:
  - The probes in the input shares: $\mathcal{P}_{input}$.
  - The probes in the random variables: $\mathcal{P}_{rand}$, which can be regarded as the variables in the calculation of $\boldsymbol{r}_k = -\boldsymbol{\alpha}_k \boldsymbol{R}$ or $\sum \boldsymbol{R}^{\mathrm{T}}$. Thus, for each probe p in $\mathcal{P}_{rand}$, there exists a function $g : \mathbb{F}_q^d \to \mathbb{F}_q$ and an index $i \in [1\!:\!d]$ (or, $j \in [1\!:\!d]$), such that $p = g(\boldsymbol{R}[i,1], \ldots, \boldsymbol{R}[i,d])$ (or, $p = g(\boldsymbol{R}[1,j], \ldots, \boldsymbol{R}[d,j])$).
  - The probes in the computation of $\sum \boldsymbol{t}_k$ with $\boldsymbol{t}_k \stackrel{\text{def}}{=} \hat{\boldsymbol{x}}_k[1\!:\!d] \oplus \boldsymbol{r}_k$: $\mathcal{P}_{sum}$.

  We have $|\mathcal{P}_{input}| + |\mathcal{P}_{rand}| + |\mathcal{P}_{sum}| \le |\mathcal{P}_{int}|$.
- Output probes $\mathcal{O}$ (By Lemma 13, $|\mathcal{O}| = f_b(\mathcal{O})$):
  - The probes in $\hat{\boldsymbol{z}}_{[1:\ell]}[0]$: $\mathcal{O}_1$.
  - The probes in $\hat{\boldsymbol{z}}[1:d]$: $\mathcal{O}_2$.

  We have $|\mathcal{O}_1| + |\mathcal{O}_2| + |\mathcal{P}_{input}| + |\mathcal{P}_{rand}| + |\mathcal{P}_{sum}| = f_b(\mathcal{O}) + |\mathcal{P}_{int}| \le d$.

We build a set $\mathcal{S}$ and temporary indices' sets $\mathcal{I}_{rand}$ and $\mathcal{J}_{rand}$ and run a simulator that proceeds by the following steps.

1. Initiate sets $\mathcal{I}_{rand}$, $\mathcal{J}_{rand}$ and $\mathcal{S}$ to be empty.
2. Put the probes in $\mathcal{P}_{input}$ into $\mathcal{S}$, and they can be simulated with $\mathcal{S}$. This step puts at most $|\mathcal{P}_{input}|$ shares into $\mathcal{S}$. Now, $f_b(\mathcal{S}) \le |\mathcal{P}_{input}|$.
3. The probes in $\mathcal{P}_{rand}$ or $\mathcal{O}_2$ can be simulated by sampling from uniform distribution. And, for each probe in the calculation in $\mathcal{P}_{rand}$, say $p = g(\boldsymbol{R}[i,1], \ldots, \boldsymbol{R}[i,d])$ (or, $p = g(\boldsymbol{R}[1,j], \ldots, \boldsymbol{R}[d,j])$), we put the corresponding index i to the set $\mathcal{I}_{rand}$ (or, j to the set $\mathcal{J}_{rand}$), which is useful in the latter process of simulation. In this step, there are at most $|\mathcal{P}_{rand}| + |\mathcal{O}_2|$ indices put into $\mathcal{I}_{rand}$ and at most $|\mathcal{P}_{rand}|$ indices put into $\mathcal{J}_{rand}$. That is, now, $|\mathcal{I}_{rand}| \le |\mathcal{P}_{rand}| + |\mathcal{O}_2|$ and $|\mathcal{J}_{rand}| \le |\mathcal{P}_{rand}|$.

4. For each probe in $\mathcal{P}_{sum}$, there exists a set $\mathcal{J}' \subseteq [1:d]$, such that the probe can be represented as $p = \sum \boldsymbol{t}_k[\mathcal{J}']$. Then, $\mathcal{J}'$ can be divided into two parts $\mathcal{J}'_1$ and $\mathcal{J}'_2$ such that $\mathcal{J}'_1 \subseteq \mathcal{J}_{rand}$ and $\mathcal{J}'_2 \subseteq [1:d]/\mathcal{J}_{rand}$. Then, the probe can be represented as $p = \sum \boldsymbol{t}_k[\mathcal{J}'_1] \oplus \sum \boldsymbol{t}_k[\mathcal{J}'_2]$, we have:
   - As $\sum \boldsymbol{t}_k[\mathcal{J}'_1]$ can be simulated with $\hat{\boldsymbol{x}}_k[\mathcal{J}_{rand}]$, put $\hat{\boldsymbol{x}}_k[\mathcal{J}_{rand}]$ into $\mathcal{S}$. Additionally, put $\hat{\boldsymbol{x}}_k[0]$ into $\mathcal{S}$, which is useful in the latter simulation. We have $f_b(\hat{\boldsymbol{x}}_k[\mathcal{J}_{rand}] \cup \hat{\boldsymbol{x}}_k[0]) \leq |\mathcal{J}_{rand}| + 1$.
   - We then consider $\sum \boldsymbol{t}_k[\mathcal{J}'_2] = \sum \left(\hat{\boldsymbol{x}}_k[\mathcal{J}'_2] \oplus \boldsymbol{\alpha}_k \boldsymbol{R}[,\mathcal{J}'_2]\right)$. By Lemma 1, any $|\bar{\mathcal{I}}_{rand}|$ rows of $(\boldsymbol{AR}[\bar{\mathcal{I}}_{rand},])[,\mathcal{J}'_2]$ are uniformly distributed with $\bar{\mathcal{I}}_{rand} \stackrel{\text{def}}{=} [1:d]/\mathcal{I}_{rand}$, and thus any $|\bar{\mathcal{I}}_{rand}|$ rows of $(\boldsymbol{AR})[,\mathcal{J}'_2]$ are uniformly distributed (corresponding to $(\boldsymbol{\alpha}_k \boldsymbol{R})[,\mathcal{J}'_2]$ of $|\bar{\mathcal{I}}_{rand}|$ gadgets). Clearly, as $|\mathcal{P}_{sum}| \leq d - |\mathcal{P}_{rand}| - |\mathcal{O}_2| = |\bar{\mathcal{I}}_{rand}|$. $\sum \boldsymbol{t}_k[\mathcal{J}'_2]$ can be simulated from uniform distribution without knowing any input shares.

   Then, all the probes in $\mathcal{P}_{sum}$ can be simulated with $\mathcal{S}$, and now:

$$f_b(\mathcal{S}) \leq |\mathcal{P}_{input}| + |\mathcal{J}_{rand}| + |\mathcal{P}_{sum}| \leq |\mathcal{P}_{input}| + |\mathcal{P}_{rand}| + |\mathcal{P}_{sum}| \leq |\mathcal{P}_{int}|.$$

5. For each probe in $\mathcal{O}_1$, there exists a set $\mathcal{J}' \subseteq [1:d]$, and the probe can be represented as $p = \hat{\boldsymbol{x}}_k[0] \oplus \sum \boldsymbol{t}_k[\mathcal{J}']$, then $\mathcal{J}'$ can be partitioned into two parts $\mathcal{J}'_1$ and $\mathcal{J}'_2$ such that $\mathcal{J}'_1 \subseteq \mathcal{J}_{rand}$ and $\mathcal{J}'_2 \subseteq [0:d]/\mathcal{J}_{rand}$. Then, the probe can be represented as $p = \hat{\boldsymbol{x}}_k[0] \oplus \sum(\hat{\boldsymbol{x}}_k[\mathcal{J}'_1] \oplus \boldsymbol{r}_k[\mathcal{J}'_1]) \oplus \sum(\hat{\boldsymbol{x}}_k[\mathcal{J}'_2] \oplus \boldsymbol{r}_k[\mathcal{J}'_2])$. As $|\mathcal{J}_{rand}| = |\mathcal{P}_{rand}| \leq d$, $\mathcal{J}'_2 \neq \emptyset$. We separate the analysis into following two cases.
   - If there exists no probe in $\mathcal{P}_{sum} \cap \mathcal{V}_k$ with $\mathcal{V}_k$ the set of all the internal variables in G_k, then $\boldsymbol{r}_k[\mathcal{J}'_2]$ should not appears in internal probes. Then, the probe can be simulated by sampling from uniform distribution.
   - If there exists at least one probe in $\mathcal{P}_{sum} \cap \mathcal{V}_k$, the simulator has already put $\hat{\boldsymbol{x}}_k[\mathcal{J}'_1 \cup \{0\}]$ into $\mathcal{S}$, and thus $\hat{\boldsymbol{x}}_k[0] \oplus \sum(\hat{\boldsymbol{x}}_k[\mathcal{J}'_1] \oplus \boldsymbol{r}_k[\mathcal{J}'_1])$ can be simulated with $\mathcal{S}$. For $\hat{\boldsymbol{x}}_k[\mathcal{J}'_2]$, as $\sum \boldsymbol{t}_k[\mathcal{J}'_2] = \sum \left(\hat{\boldsymbol{x}}_k[\mathcal{J}'_2] \oplus \boldsymbol{\alpha}_k \boldsymbol{R}[,\mathcal{J}'_2]\right)$, by Lemma 1, any $|\bar{\mathcal{I}}_{rand}|$ rows of $(\boldsymbol{AR}[\bar{\mathcal{I}}_{rand},])[,\mathcal{J}'_2]$ are uniformly distributed with $\bar{\mathcal{I}}_{rand} \stackrel{\text{def}}{=} [1:d]/\mathcal{I}_{rand}$, and thus any $|\bar{\mathcal{I}}_{rand}|$ rows of $(\boldsymbol{AR})[,\mathcal{J}'_2]$ are uniformly distributed (corresponding to $(\boldsymbol{\alpha}_k \boldsymbol{R})[,\mathcal{J}'_2]$ of $|\bar{\mathcal{I}}_{rand}|$ gadgets). Clearly, as $|\mathcal{P}_{sum}| \leq d - |\mathcal{P}_{rand}| - |\mathcal{O}_2| = |\bar{\mathcal{I}}_{rand}|$. $\sum \boldsymbol{t}_k[\mathcal{J}'_2]$ can be simulated from uniform distribution without knowing any input shares.

Now, all the probes are simulated with shares $\mathcal{S}$ such that $f_b(\mathcal{S}) \leq |\mathcal{P}_{sum}| + |\mathcal{P}_{rand}| + |\mathcal{P}_{input}|$, and positions of $\mathcal{S}$ are determined by internal probes. Hence, for any internal probes $\mathcal{P}_{int}$, there exists a simulator Sim shares $\mathcal{S}$ of input sharings with $f_b(\mathcal{S}) = |\mathcal{P}_{int}|$, such that for any shares $\mathcal{O}$ of output sharings with $|\mathcal{P}_{int}| + f_b(\mathcal{O}) \leq d$, the distributions of $\mathsf{G}_{\mathcal{P}}(\{\hat{\boldsymbol{x}}_i\}_{i=1}^{\ell})$ and $\mathsf{Sim}(\mathcal{S})$ are identical, where $\mathcal{P}$ is the set of all probes. □

We give multiplication gadget with common shares in Gadget 4, named SubMul. It complies with the strategy of product-then-compress of the ISW multiplication. We also highlight parts that are independent of the input sharings $\hat{\boldsymbol{x}}$ and $\hat{\boldsymbol{y}}$ (by underlining in red color) and parts that are independent of $\hat{\boldsymbol{x}}[0]$ and $\hat{\boldsymbol{y}}[0]$ (by underwaving in blue color). We give the security of parallel composition and correctness in Lemma 15. Notably, the security part of Lemma 15 says that, informally speaking, the parallel composition of SubMul*s* reusing randomness is somewhat secure in the probing model, if inputs of different gadgets have shares in common.

Lemma 15. *Let* G *be a parallel composition of* ℓ *gadgets* SubMul *using the same random matrix* $\boldsymbol{R}$ *with inputs sharings* $\hat{\boldsymbol{x}}_{[1:\ell]}$ *and* $\hat{\boldsymbol{y}}_{[1:\ell]}$ *satisfying* $\hat{\boldsymbol{x}}_k[1:d] = \hat{\boldsymbol{x}}_{k'}[1:d]$ *and* $\hat{\boldsymbol{y}}_k[1:d] = \hat{\boldsymbol{y}}_{k'}[1:d]$ *for any* $k, k' \in [1:\ell]$. *Then we have:*

Gadget 4 $\mathsf{SubMul}_{\alpha}^{\boldsymbol{R}}$

Input: sharings $\hat{\boldsymbol{x}} \in \mathbb{F}_q^{d+1}$, $\hat{\boldsymbol{y}} \in \mathbb{F}_q^{d+1}$.
Output: sharing $\hat{\boldsymbol{z}} \in \mathbb{F}_q^{d+1}$.
Parameter: $\boldsymbol{\alpha} \in \mathbb{F}_q^{d}$.
Randomness: matrix $\boldsymbol{R} \in \mathbb{F}_q^{d\times d}$.

The gadget ensures that: $\langle \hat{\boldsymbol{z}}, \boldsymbol{a}\rangle = \langle \hat{\boldsymbol{x}}, \boldsymbol{a}\rangle\langle \hat{\boldsymbol{y}}, \boldsymbol{a}\rangle$ with $\boldsymbol{a} \stackrel{\text{def}}{=} [1, \boldsymbol{\alpha}]$.

1: $\hat{\boldsymbol{z}}[1:d] := -\boldsymbol{\alpha}\boldsymbol{R}^{\mathrm{T}}$
2: $e := \hat{\boldsymbol{x}}[0]\hat{\boldsymbol{y}}[0]$, $\boldsymbol{v} := \hat{\boldsymbol{x}}[0]\hat{\boldsymbol{y}}[1:d]$, $\boldsymbol{w} := \hat{\boldsymbol{x}}[1:d]\hat{\boldsymbol{y}}[0]$
$\boldsymbol{T} := \hat{\boldsymbol{x}}[1:d]^{\mathrm{T}}\hat{\boldsymbol{y}}[1:d]$
▷ Calculate the outer product: $\begin{bmatrix} e, & \boldsymbol{v} \\ \boldsymbol{w}^{\mathrm{T}}, & \boldsymbol{T} \end{bmatrix} := \hat{\boldsymbol{x}}^{\mathrm{T}}\hat{\boldsymbol{y}}$
3: $\boldsymbol{r} := \boldsymbol{\alpha}(\boldsymbol{T} \oplus \boldsymbol{R})$
4: $\boldsymbol{t} := \big(\boldsymbol{w} \oplus (\boldsymbol{v} \oplus \boldsymbol{r})\big)$
5: $\boldsymbol{z}[0] := e \oplus \langle \boldsymbol{t}, \boldsymbol{\alpha}\rangle$

- ***Correctness***. *For any* $k \in [1:\ell]$, $\big\langle \hat{\boldsymbol{z}}_k, [1, \boldsymbol{\alpha}_k]\big\rangle = \big\langle \hat{\boldsymbol{x}}_k, [1, \boldsymbol{\alpha}_k]\big\rangle\big\langle \hat{\boldsymbol{y}}_k, [1, \boldsymbol{\alpha}_k]\big\rangle$, *and* $\hat{\boldsymbol{z}}_k[1:d]$ *is determined only by the randomness.*
- ***Security***. *Any internal probes* $\mathcal{P}_{int}$ *and output probes* $\mathcal{O}$ *with* $|\mathcal{P}_{int}| + f_b(\mathcal{O}) \le d$ *can be simulated with some shares* $\mathcal{S}_x$ *of* $\{\hat{\boldsymbol{x}}_k\}_{k=1}^{\ell}$ *and some shares* $\mathcal{S}_y$ *of* $\{\hat{\boldsymbol{y}}_k\}_{k=1}^{\ell}$ *such that* $f_b(\mathcal{S}_x) \le |\mathcal{P}_{int}| + f_b(\mathcal{O})$ *and* $f_b(\mathcal{S}_y) \le |\mathcal{P}_{int}| + f_b(\mathcal{O})$ *, if the matrix* $[\boldsymbol{\alpha}_1; \ldots; \boldsymbol{\alpha}_\ell]$ *is MDS.*

Proof of the correctness property in Lemma 15. For any $k \in [1:\ell]$, we have

$$\begin{aligned}
\hat{\boldsymbol{z}}_k[0] \oplus \ldots \oplus \hat{\boldsymbol{z}}_k[d] &= \hat{\boldsymbol{z}}_k[0] \oplus \sum\big((-\boldsymbol{\alpha}_k\boldsymbol{R}^{\mathrm{T}}) \odot \boldsymbol{\alpha}_k\big) \\
&= e \oplus (\boldsymbol{v}_k \oplus \boldsymbol{w}_k \oplus \boldsymbol{r}_k)\boldsymbol{\alpha}^{\mathrm{T}} \oplus (-\boldsymbol{\alpha}_k\boldsymbol{R}^{\mathrm{T}}\boldsymbol{\alpha}_k^{\mathrm{T}}) \\
&= e \oplus (\boldsymbol{v}_k \oplus \boldsymbol{w}_k \oplus \boldsymbol{\alpha}(\hat{\boldsymbol{x}}[1:d]^{\mathrm{T}}\hat{\boldsymbol{y}}[1:d] \oplus \boldsymbol{R}^{\mathrm{T}}))\boldsymbol{\alpha}^{\mathrm{T}} \oplus (-\boldsymbol{\alpha}_k\boldsymbol{R}^{\mathrm{T}}\boldsymbol{\alpha}_k^{\mathrm{T}}) \\
&= e \oplus \boldsymbol{v}\boldsymbol{\alpha}_k^{\mathrm{T}} \oplus \boldsymbol{w}\boldsymbol{\alpha}_k^{\mathrm{T}} \oplus \boldsymbol{\alpha}_k\hat{\boldsymbol{x}}[1:d]^{\mathrm{T}}\hat{\boldsymbol{y}}[1:d]\boldsymbol{\alpha}_k^{\mathrm{T}} \oplus \boldsymbol{\alpha}_k\boldsymbol{R}^{\mathrm{T}}\boldsymbol{\alpha}_k^{\mathrm{T}} \oplus (-\boldsymbol{\alpha}_k\boldsymbol{R}^{\mathrm{T}}\boldsymbol{\alpha}_k^{\mathrm{T}}) \\
&= \hat{\boldsymbol{x}}[0]\hat{\boldsymbol{y}}[0] \oplus \hat{\boldsymbol{x}}[0]\hat{\boldsymbol{y}}[1:d]\boldsymbol{\alpha}_k^{\mathrm{T}} \oplus \hat{\boldsymbol{y}}[0]\hat{\boldsymbol{x}}[1:d]\boldsymbol{\alpha}_k^{\mathrm{T}} \oplus \boldsymbol{\alpha}_k\hat{\boldsymbol{x}}[1:d]^{\mathrm{T}}\hat{\boldsymbol{y}}[1:d]\boldsymbol{\alpha}_k^{\mathrm{T}} \\
&= (\hat{\boldsymbol{x}}[0] \oplus \hat{\boldsymbol{x}}[1:d]\boldsymbol{\alpha}_k^{\mathrm{T}})(\hat{\boldsymbol{y}}[0] \oplus \hat{\boldsymbol{y}}[1:d]\boldsymbol{\alpha}_k^{\mathrm{T}}) \\
&= xy \ .
\end{aligned}$$

And, as $\hat{\boldsymbol{z}}_k[1:d] = \boldsymbol{\alpha}_k\boldsymbol{R}^{\mathrm{T}}$, $\hat{\boldsymbol{z}}_k[1:d]$ is only determined by the randomness. □

Proof of the security property in Lemma 15. We use subscript to differentiate the parameter and variables of different gadgets, e.g., let $\boldsymbol{\alpha}_k$ and $\hat{\boldsymbol{x}}_k$ be the parameter and input sharing of the k-th gadget. Particularly, as $\hat{\boldsymbol{x}}_k[1:d]$ (resp., $\hat{\boldsymbol{y}}_k[1:d]$) is identical for any values k, the output shares of the parallel composed gadget can be represented as $\hat{\boldsymbol{x}}_{[1:\ell]}[0] \cup \hat{\boldsymbol{x}}[1:d]$ (resp., $\hat{\boldsymbol{y}}_{[1:\ell]}[0] \cup \hat{\boldsymbol{y}}[1:d]$). We also define the matrix $\boldsymbol{A} \stackrel{\text{def}}{=} [\boldsymbol{\alpha}_1; \ldots; \boldsymbol{\alpha}_\ell]$. Let $\boldsymbol{T}_{\boldsymbol{R}} \stackrel{\text{def}}{=} \boldsymbol{T} \oplus \boldsymbol{R}$. The probes are divided into different subsets based on the types of variables:

- Internal probes $\mathcal{P}_{int}$:
  - The probes in the input shares and tensor product: $\mathcal{P}_{input}$.
  - The probes in the random variables: $\mathcal{P}_{rand}$, which can be regarded as the variables in the calculation $\boldsymbol{r}_k = -\boldsymbol{\alpha}_k\boldsymbol{R}^{\mathrm{T}}$. Thus, for each probe p in $\mathcal{P}_{rand}$, there exists a function $g : \mathbb{F}_q^d \to \mathbb{F}_q$ and an index $i \in [1:d]$, such that $p = g(\boldsymbol{R}[i,1], \ldots, \boldsymbol{R}[i,d])$.

– The probes in the computation of $\boldsymbol{r}_k = \boldsymbol{\alpha}_k(\boldsymbol{T} \oplus \boldsymbol{R})$: $\mathcal{P}_\times$. For each probe p in $\mathcal{P}_\times$, there exists a function $g : \mathbb{F}_q^d \to \mathbb{F}_q$ and an index $j \in [1:d]$, such that $p = g(\boldsymbol{T}_{\boldsymbol{R}}[1,j], \ldots, \boldsymbol{T}_{\boldsymbol{R}}[d,j])$.
– The probes in the computation of $e_k \oplus \langle \hat{\boldsymbol{t}}_k, \boldsymbol{\alpha}_k \rangle = e_k \oplus \langle (\boldsymbol{w} \oplus (\boldsymbol{v} \oplus \boldsymbol{r})), \boldsymbol{\alpha}_k \rangle$: $\mathcal{P}_{sum}$.

We have $|\mathcal{P}_{input}| + |\mathcal{P}_{rand}| + |\mathcal{P}_\times| + |\mathcal{P}_{sum}| = |\mathcal{P}_{int}|$.

- Output probes $\mathcal{O}$:
  – The probes in $\hat{\boldsymbol{z}}_{[1:\ell]}[0]$: $\mathcal{O}_1$.
  – The probes in $\hat{\boldsymbol{z}}_{[1:\ell]}[1:d]$: $\mathcal{O}_2$. Let $\mathcal{I}_O$ be the set of shares' indices (in a sharing) of the probes in $\mathcal{O}_2$. That is, suppose $\mathcal{O}_2 = \{\hat{\boldsymbol{z}}_k[\mathcal{I}_k]\}_{k=1}^{\ell}$, then $\mathcal{I}_O = \bigcup_{k=1}^{\ell} \mathcal{I}_k$.

  By Lemma 12, we have $|\mathcal{O}_1| + |\mathcal{I}_O| \le f_b(\mathcal{O})$. Thus, $|\mathcal{O}_1| + |\mathcal{I}_O| + |\mathcal{P}_{input}| + |\mathcal{P}_{rand}| + |\mathcal{P}_\times| + |\mathcal{P}_{sum}| = f_b(\mathcal{O}) + |\mathcal{P}_{int}| \le d$

We build sets $\mathcal{S}_x$, $\mathcal{S}_y$ and temporary indices's sets $\mathcal{I}_{rand}$ and $\mathcal{J}_{rand}$ and run a simulator as following steps.

1. Initiate sets $\mathcal{S}_x$, $\mathcal{S}_y$, $\mathcal{I}_{rand}$ and $\mathcal{J}_{rand}$ to be empty.
2. For each probe in $\mathcal{P}_{input}$, say $\hat{\boldsymbol{x}}_k[i]$, $\hat{\boldsymbol{y}}_k[j]$ or $\hat{\boldsymbol{x}}_k[i]\hat{\boldsymbol{y}}_k[j]$ for $k \in [1:\ell]$ and $i, j \in [0:d]$, put $\hat{\boldsymbol{x}}_k[i]$ into $\mathcal{S}_y$ and $\hat{\boldsymbol{y}}_k[j]$ into $\mathcal{S}_x$. Then, p is a function of $\{\hat{\boldsymbol{x}}_k[\mathcal{I}_x]\}_{k\in\mathcal{K}_x}$ and $\{\hat{\boldsymbol{y}}_k[\mathcal{I}_y]\}_{k\in\mathcal{K}_y}$, and thus can be simulated. After all the probes in $\mathcal{P}_{input}$ are simulated, we have $|\mathcal{S}_x| \le |\mathcal{P}_{input}|$ and $|\mathcal{S}_y| \le |\mathcal{P}_{input}|$.
3. The probes in $\mathcal{P}_{rand}$ or $\mathcal{O}_2$ can be simulated by sampling uniform distribution. Additionally, for each probe in calculation of $-\boldsymbol{\alpha}_k\boldsymbol{R}^{\mathrm{T}}$, say $p = g(\boldsymbol{R}[i,1], \ldots, \boldsymbol{R}[i,d])$, put the corresponding index i to the sets $\mathcal{I}_{rand}$, which is useful in the latter process of simulation. In this step, there are at most $|\mathcal{P}_{rand}| + |\mathcal{I}_O|$ indices put into $\mathcal{I}_{rand}$. That is, now $|\mathcal{I}_{rand}| \le |\mathcal{P}_{rand}| + |\mathcal{I}_O|$.
4. For each probe in $\mathcal{P}_\times$, say, $p = g(\boldsymbol{T}_{\boldsymbol{R}}[1,j], \ldots, \boldsymbol{T}_{\boldsymbol{R}}[d,j])$, there exists a set $\mathcal{I}' \subseteq [1:d]$, and the probe can be represented as $p = g(\boldsymbol{T}_{\boldsymbol{R}}[\mathcal{I}', j])$, then $\mathcal{I}'$ can be divided into two parts: $\mathcal{I}'_1 \subseteq \mathcal{I}_{rand}$ and $\mathcal{I}'_2 \subseteq [1:d]/\mathcal{I}_{rand}$. Hence, the probe can be represented as $p = g_1(\boldsymbol{T}_{\boldsymbol{R}}[\mathcal{I}'_1, j]) \oplus g_2(\boldsymbol{T}_{\boldsymbol{R}}[\mathcal{I}'_2, j])$, and the simulator puts j into $\mathcal{J}_{rand}$, put $\hat{\boldsymbol{y}}[j]$ into $\mathcal{S}_y$, and separate the rest of analysis into following two parts.
   - For $\boldsymbol{T}_{\boldsymbol{R}}[\mathcal{I}'_1, j]$, as it can be simulated with $\hat{\boldsymbol{x}}[\mathcal{I}_{rand}]$ and $\hat{\boldsymbol{y}}[j]$, the simulator puts $\hat{\boldsymbol{x}}[\mathcal{I}_{rand}]$ into $\mathcal{S}_x$ and $\hat{\boldsymbol{y}}[j]$ has already in $\mathcal{S}_y$, then $\boldsymbol{T}_{\boldsymbol{R}}[\mathcal{I}'_1, j]$ can be simulated with $\mathcal{S}_x$ and $\mathcal{S}_y$.
   - For $\boldsymbol{T}_{\boldsymbol{R}}[\mathcal{I}'_2, j] = \boldsymbol{T}[\mathcal{I}'_2, j] \oplus \boldsymbol{R}[\mathcal{I}'_2, j]$, as variables in $\boldsymbol{R}[\mathcal{I}'_2, j]$ do not appear in the previous probes, $\boldsymbol{T}_{\boldsymbol{R}}[\mathcal{I}'_2, j]$ can be simulated by sampling from uniform distribution without knowing any input shares.

   Then, all the probes in $\mathcal{P}_\times$ can be simulated with $\mathcal{S}_x$ and $\mathcal{S}_y$. Now, we have

$$|\mathcal{S}_x| \le |\mathcal{P}_{input}| + |\mathcal{I}_{rand}| \le |\mathcal{P}_{input}| + |\mathcal{P}_{rand}| + |\mathcal{I}_O|,$$
$$|\mathcal{S}_y| \le |\mathcal{P}_{input}| + |\mathcal{P}_\times|,\ |\mathcal{J}_{rand}| \le |\mathcal{P}_\times|, \text{ and } \hat{\boldsymbol{y}}[\mathcal{J}_{rand}] \subseteq \mathcal{S}_y\ .$$

5. For each probe in $\mathcal{P}_{sum}$ or $\mathcal{O}_1$, there exists a set $\mathcal{J}' \subseteq [1:d]$, and the probe can be represented as $p = \beta e_k \oplus \langle \boldsymbol{t}_k[\mathcal{J}'], \boldsymbol{\alpha}_k[\mathcal{J}'] \rangle = g(e_k, \boldsymbol{t}_k[\mathcal{J}'])$ with $\beta \in \{0,1\}$. $\mathcal{J}'$ can be divided into two parts: $\mathcal{J}'_1 \subseteq \mathcal{J}_{rand}$ and $\mathcal{J}'_2 \subseteq [1:d]/\mathcal{J}_{rand}$. Hence, the probe can be represented as $p = \beta e_k \oplus g_1(\boldsymbol{t}_k[\mathcal{J}'_1]) \oplus g_2(\boldsymbol{t}_k[\mathcal{J}'_2])$. The simulator puts $\hat{\boldsymbol{x}}_k[0]$ into $\mathcal{S}_x$ and $\hat{\boldsymbol{y}}_k[0]$ into $\mathcal{S}_y$ (which increases the sizes of both $\mathcal{S}_x$ and $\mathcal{S}_y$ by $|\mathcal{P}_{sum}| + |\mathcal{O}_1|$), and we separate the rest of analysis into following three parts.

- For βe_k, as the simulator has already put $\hat{\boldsymbol{x}}_k[0]$ into $\mathcal{S}_x$ and $\hat{\boldsymbol{y}}_k[0]$ into $\mathcal{S}_y$, e_k can be simulated with $\mathcal{S}_x$ and $\mathcal{S}_y$.
- For $\boldsymbol{t}_k[\mathcal{J}_1']$, as it can be simulated with $\hat{\boldsymbol{x}}[\mathcal{J}_{rand}]$, $\hat{\boldsymbol{y}}[\mathcal{J}_1']$, $\hat{\boldsymbol{x}}_k[0]$ and $\hat{\boldsymbol{y}}_k[0]$, the simulator puts $\hat{\boldsymbol{x}}[\mathcal{J}_{rand}]$ into $\mathcal{S}_x$ (which increases the sizes of both $\mathcal{S}_x$ by $|\mathcal{J}_{rand}| \leq |\mathcal{P}_\times|$). We also have $\hat{\boldsymbol{y}}[\mathcal{J}_1'] \subseteq \mathcal{S}_y$ from step 4, and the simulator has already put $\hat{\boldsymbol{x}}_k[0]$ into $\mathcal{S}_x$ and $\hat{\boldsymbol{y}}_k[0]$ into $\mathcal{S}_y$. Then, $\boldsymbol{t}_k[\mathcal{J}_1']$ can be simulated with $\mathcal{S}_x$ and $\mathcal{S}_y$.
- For $\boldsymbol{t}_k[\mathcal{J}_2']$, as we can rewrite it as $\boldsymbol{t}_k[\mathcal{J}_2'] = \boldsymbol{s} \oplus \boldsymbol{\alpha}_k \boldsymbol{R}[,\mathcal{J}_2']$ with $\boldsymbol{s}$ a function of $\hat{\boldsymbol{x}}[1:d]$, $\hat{\boldsymbol{y}}[1:d]$, $\hat{\boldsymbol{x}}_k[0]$ and $\hat{\boldsymbol{y}}_k[0]$, by Lemma 1, any $|\bar{\mathcal{I}}_{rand}|$ rows of $(\boldsymbol{AR}[\bar{\mathcal{I}}_{rand},])[,\mathcal{J}_2']$ are uniformly distributed with $\bar{\mathcal{I}}_{rand} \stackrel{\text{def}}{=} [1:d]/\mathcal{I}_{rand}$, and thus any $|\bar{\mathcal{I}}_{rand}|$ rows of $(\boldsymbol{AR})[,\mathcal{J}_2']$ are uniformly distributed (corresponding to $(\boldsymbol{\alpha}_k \boldsymbol{R})[,\mathcal{I}_2']$ of $|\bar{\mathcal{I}}_{rand}|$ gadgets). Clearly, as $|\mathcal{P}_{sum}| \leq d - |\mathcal{P}_{rand}| = |\bar{\mathcal{I}}_{rand}|$, the simulator can sample $\boldsymbol{t}_k[\mathcal{J}_2']$ from uniform distribution, and $\boldsymbol{t}_k[\mathcal{J}_2']$ can be simulated without knowing any input shares.

Then, all the probes in $\mathcal{P}_{sum}$ can be simulated with $\mathcal{S}_x$ and $\mathcal{S}_y$. Now, we have

$$\begin{aligned}
|\mathcal{S}_x| &\leq |\mathcal{P}_{input}| + |\mathcal{P}_{rand}| + |\mathcal{I}_O| + |\mathcal{P}_{sum}| + |\mathcal{O}_1| + |\mathcal{J}_{rand}| \\
&\leq |\mathcal{P}_{input}| + |\mathcal{P}_{rand}| + |\mathcal{I}_O| + |\mathcal{P}_{sum}| + |\mathcal{O}_1| + |\mathcal{P}_\times| \\
&\leq |\mathcal{P}_{int}| + |\mathcal{I}_O| + |\mathcal{O}_1| \leq |\mathcal{P}_{int}| + f(\mathcal{O}), \text{ and} \\
|\mathcal{S}_y| &\leq |\mathcal{P}_{input}| + |\mathcal{P}_\times| + |\mathcal{P}_{sum}| + |\mathcal{O}_1| \leq |\mathcal{P}_{int}| + |\mathcal{I}_O| + |\mathcal{O}_1| \\
&\leq |\mathcal{P}_{int}| + f(\mathcal{O}) \ .
\end{aligned}$$

Now, all the probes are simulated with $\mathcal{S}_x \cup \mathcal{S}_y$. By Lemma 13, $f_b(\mathcal{S}_x) = |\mathcal{S}_x|$ and $f_b(\mathcal{S}_y) = |\mathcal{S}_y|$. Therefore, for any internal probes $\mathcal{P}_{int}$ and output probes $\mathcal{O}$, there exists a simulator Sim and input shares $\mathcal{S}$ with $f_b(\mathcal{S}_x) \leq |\mathcal{P}_{int}| + f_b(\mathcal{O})$ and $f_b(\mathcal{S}_y) \leq |\mathcal{P}_{int}| + f_b(\mathcal{O})$, such that the distributions of $\mathsf{G}_{\mathcal{P}}\big(\{\hat{\boldsymbol{x}}_i\}_{i=1}^{\ell} \cup \{\hat{\boldsymbol{y}}_i\}_{i=1}^{\ell}\big)$ and $\mathsf{Sim}\,(\mathcal{S}_x \cup \mathcal{S}_y)$ are identical, where $\mathcal{P}$ is the set of all probes. □

We can build our RNI-f_b multiplication with common shares by linking an $\mathsf{RNIRefresh}$ to each input sharing of SubMul, and give the construction in Gadget 5, named Mul. Similarly, we can build an RNI-f_b addition gadget by linking an $\mathsf{RNIRefresh}$ to each input/output sharing of $\mathsf{TrivAdd}$, and give the construction in Gadget 6, named Add. The correctness of Mul and Add can be guaranteed by the constructions of SubMul, $\mathsf{TrivLin}$ and $\mathsf{RNIRefresh}$. In Lemma 16, we show that the parallel composition of multiple Mul*s* and Add*s* is RNI-f_b, if the concatenation of their parameters (i.e., vectors $\boldsymbol{\alpha}$) is an MDS matrix.

Gadget 5 $\mathsf{Mul}_{\boldsymbol{\alpha}}^{\boldsymbol{R}_x,\boldsymbol{R}_y,\boldsymbol{R}}$

Input: sharings $\hat{\boldsymbol{x}} \in \mathbb{F}_q^{d+1}$.
Output: sharing $\hat{\boldsymbol{z}} \in \mathbb{F}_q^{d+1}$.
Parameter: $\boldsymbol{\alpha} \in \mathbb{F}_q^d$.
Randomness: matrices $\boldsymbol{R}_x, \boldsymbol{R}_y, \boldsymbol{R} \in \mathbb{F}_q^{d \times d}$.
The gadget ensures that: $\sum \hat{\boldsymbol{z}} = (\sum \hat{\boldsymbol{x}})(\sum \hat{\boldsymbol{y}})$.

1: $\hat{\boldsymbol{x}}' := \mathsf{RNIRefresh}_{\boldsymbol{\alpha}}^{\boldsymbol{R}_x}(\hat{\boldsymbol{x}})$
2: $\hat{\boldsymbol{y}}' := \mathsf{RNIRefresh}_{\boldsymbol{\alpha}}^{\boldsymbol{R}_y}(\hat{\boldsymbol{y}})$
3: $\hat{\boldsymbol{z}} := \mathsf{SubMul}_{\boldsymbol{\alpha}}^{\boldsymbol{R}}(\hat{\boldsymbol{x}}', \hat{\boldsymbol{y}}') \odot [1, \boldsymbol{\alpha}]$

Gadget 6 $\mathsf{Add}_{\boldsymbol{\alpha}}^{\boldsymbol{R}_x,\boldsymbol{R}_y}$

Input: sharings $\hat{\boldsymbol{x}}, \hat{\boldsymbol{y}} \in \mathbb{F}_q^{d+1}$.
Output: sharing $\hat{\boldsymbol{z}} \in \mathbb{F}_q^{d+1}$.
Parameter: $\boldsymbol{\alpha} \in \mathbb{F}_q^d$.
Randomness: matrices $\boldsymbol{R}_x, \boldsymbol{R}_y \in \mathbb{F}_q^{d\times d}$.
The gadget ensures that: $\sum \hat{\boldsymbol{z}} = \sum \hat{\boldsymbol{x}} \oplus \sum \hat{\boldsymbol{y}}$.

1: $\hat{\boldsymbol{x}}' := \mathsf{RNIRefresh}_{\boldsymbol{\alpha}}^{\boldsymbol{R}_x}(\hat{\boldsymbol{x}}) \odot [1, \boldsymbol{\alpha}]$
2: $\hat{\boldsymbol{y}}' := \mathsf{RNIRefresh}_{\boldsymbol{\alpha}}^{\boldsymbol{R}_y}(\hat{\boldsymbol{y}}) \odot [1, \boldsymbol{\alpha}]$
3: $\hat{\boldsymbol{z}} := \mathsf{TrivAdd}(\hat{\boldsymbol{x}}', \hat{\boldsymbol{y}}')$

Lemma 16. *Let f_b be the block-scale function, and let ℓ_1, ℓ_2 be positive integers. Let $\mathcal{G}$ be a set consisting of ℓ_1 Muls and ℓ_2 Adds using the same randomness, i.e.,*

$$\mathcal{G} = \{\mathsf{Mul}_{\boldsymbol{\alpha}_1}^{\boldsymbol{R}_x,\boldsymbol{R}_y,\boldsymbol{R}_z}, \ldots, \mathsf{Mul}_{\boldsymbol{\alpha}_{\ell_1}}^{\boldsymbol{R}_x,\boldsymbol{R}_y,\boldsymbol{R}_z}, \mathsf{Add}_{\boldsymbol{\beta}_1}^{\boldsymbol{R}_x,\boldsymbol{R}_y}, \ldots, \mathsf{Add}_{\boldsymbol{\beta}_{\ell_2}}^{\boldsymbol{R}_x,\boldsymbol{R}_y}\}.$$

The parallel composition of gadgets in $\mathcal{G}$ is RNI-f_b, if the matrix $[\boldsymbol{\alpha}_1; \ldots; \boldsymbol{\alpha}_{\ell_1}; \boldsymbol{\beta}_1; \ldots; \boldsymbol{\beta}_{\ell_2}]$ is MDS.

Proof. The composed gadget can be partitioned into two sub-gadgets:

- The parallel composition of $\mathsf{RNIRefresh}es$ at the beginning of $\mathsf{Mul}s$ and $\mathsf{Lin}s$, denoted as G_1.
- The parallel composition of $\mathsf{SubMul}s$ and $\mathsf{TrivAdd}s$, denoted as G_2.

Let $\ell \stackrel{\text{def}}{=} \ell_1 + \ell_2$, and the sharings between G_1 and G_2 be $\hat{\boldsymbol{x}}_{[1:\ell]}$ and $\hat{\boldsymbol{y}}_{[1:\ell]}$ respectively. According to the separation, we have the following two conclusions:

- For G_1, by Lemma 14, the parallel compositions of $\{\mathsf{RNIRefresh}_{\boldsymbol{\alpha}_1}^{\boldsymbol{R}_x}, \ldots, \mathsf{RNIRefresh}_{\boldsymbol{\alpha}_\ell}^{\boldsymbol{R}_x}\}$ and $\{\mathsf{RNIRefresh}_{\boldsymbol{\alpha}_1}^{\boldsymbol{R}_y}, \ldots, \mathsf{RNIRefresh}_{\boldsymbol{\alpha}_\ell}^{\boldsymbol{R}_y}\}$ are both RNI-f_b.
- For G_2, by Lemma 15 and the instruction of $\mathsf{TrivAdd}$, any output shares $\mathcal{O}$ and t_{int} internal probes can be simulated with shares $\mathcal{S}_y$ of $\hat{\boldsymbol{x}}_{[1:\ell]}$ and shares $\mathcal{S}_y$ of $\hat{\boldsymbol{y}}_{[1:\ell]}$ that have been refreshed by G_1, such that $f_b(\mathcal{S}_x) \le t_{int} + f_b(\mathcal{O})$ and $f_b(\mathcal{S}_y) \le t_{int} + f_b(\mathcal{O})$.

At last, Lemma 16 can be directly achieved by combining the above two conclusions. □ □

Finally, to adopt Theorem 2 enabling any compositions of our new gadgets, we link the output of each Mul and Add with an $\mathsf{RNIRefresh}$, resulting in RMul and RAdd respectively. We show the constructions in Gadgets 7 and 8, and give in Theorem 2 the security of any composition of them, which can be directly deduced by Lemmas 14 and 16, and Theorem 1.

Gadget 7 $\mathsf{RMul}_{\alpha}^{\boldsymbol{R}_x,\boldsymbol{R}_y,\boldsymbol{R}_z,\boldsymbol{R}}$

Input: sharings $\hat{\boldsymbol{x}} \in \mathbb{F}_q^{d+1}$.
Output: sharing $\hat{\boldsymbol{z}} \in \mathbb{F}_q^{d+1}$.
Parameter: $\boldsymbol{\alpha} \in \mathbb{F}_q^d$.
Randomness: matrices $\boldsymbol{R}_x, \boldsymbol{R}_y, \boldsymbol{R}_z, \boldsymbol{R} \in \mathbb{F}_q^{d\times d}$.
The gadget ensures that: $\sum \hat{\boldsymbol{z}} = (\sum \hat{\boldsymbol{x}})(\sum \hat{\boldsymbol{y}})$.

1: $\hat{\boldsymbol{z}}' := \mathsf{Mul}_{\boldsymbol{\alpha}}^{\boldsymbol{R}_x,\boldsymbol{R}_y,\boldsymbol{R}}(\hat{\boldsymbol{x}}, \hat{\boldsymbol{y}})$
2: $\hat{\boldsymbol{z}} := \mathsf{RNIRefresh}_{\boldsymbol{\alpha}}^{\boldsymbol{R}_z}(\hat{\boldsymbol{z}}') \odot [1, \boldsymbol{\alpha}]$

Gadget 8 $\mathsf{RAdd}_{\alpha}^{\boldsymbol{R}_x,\boldsymbol{R}_y,\boldsymbol{R}_z}$

Input: sharings $\hat{\boldsymbol{x}}, \hat{\boldsymbol{y}} \in \mathbb{F}_q^{d+1}$.
Output: sharing $\hat{\boldsymbol{z}} \in \mathbb{F}_q^{d+1}$.
Parameter: $\boldsymbol{\alpha} \in \mathbb{F}_q^d$.
Randomness: matrices $\boldsymbol{R}_x, \boldsymbol{R}_y, \boldsymbol{R}_z \in \mathbb{F}_q^{d\times d}$.
The gadget ensures that: $\sum \hat{\boldsymbol{z}} = \sum \hat{\boldsymbol{x}} \oplus \sum \hat{\boldsymbol{y}}$.
1: $\hat{\boldsymbol{z}}' := \mathsf{Add}_{\boldsymbol{\alpha}}^{\boldsymbol{R}_x,\boldsymbol{R}_y}(\hat{\boldsymbol{x}}, \hat{\boldsymbol{y}})$
2: $\hat{\boldsymbol{z}} := \mathsf{RNIRefresh}_{\boldsymbol{\alpha}}^{\boldsymbol{R}_z}(\hat{\boldsymbol{z}}') \odot [1, \boldsymbol{\alpha}]$

Theorem 2. *Let f_b be the block-scale function. Let ℓ_1, ℓ_2 be positive integers. Let $\mathcal{G}$ be a set consisting of ℓ_1 RMuls and ℓ_2 RAdds using the same randomness, i.e.,*

$$\mathcal{G} = \{\mathsf{RMul}_{\boldsymbol{\alpha}_1}^{\mathcal{R}_1}, \ldots, \mathsf{RMul}_{\boldsymbol{\alpha}_{\ell_1}}^{\mathcal{R}_1}, \mathsf{RAdd}_{\boldsymbol{\beta}_1}^{\mathcal{R}_2}, \ldots, \mathsf{RAdd}_{\boldsymbol{\beta}_{\ell_2}}^{\mathcal{R}_2}\},$$

where $\mathcal{R}_1 = \{\boldsymbol{R}_x, \boldsymbol{R}_y, \boldsymbol{R}_z, \boldsymbol{R}\}$ and $\mathcal{R}_2 = \{\boldsymbol{R}_x, \boldsymbol{R}_y, \boldsymbol{R}_z\}$. Any composition of gadgets in $\mathcal{G}$ is RNI-f_b, if the matrix $[\boldsymbol{\alpha}_1; \ldots; \boldsymbol{\alpha}_{\ell_1}; \boldsymbol{\beta}_1; \ldots; \boldsymbol{\beta}_{\ell_2}]$ is MDS.

Note that the addition gadget RAdd additionally requires 3 calls of $\mathsf{RNIRefresh}$, which is not as efficient as expected by many previous schemes that only use the trivial implementation. But we emphasize that, in presence of the resue of randomness and intermediate variables, the calls of $\mathsf{RNIRefresh}$ might be unavoidable. Similar results can be found in [CGZ20] as well. Fortunately, by Lemma 8, the composition of $\mathsf{TrivAff}$ and any RNI-f_b gadget is a new RNI-f_b gadget. Therefore, in Corollary 1, we show the security of the composition of $\mathsf{RMul}s$, $\mathsf{RAdd}s$ and $\mathsf{TrivAdd}s$.

Corollary 1. *Let ℓ_1, ℓ_2 and ℓ_3 be positive integers. Let $\mathcal{G}$ be a set consisting of ℓ_1 RMuls, ℓ_2 RAdds and ℓ_3 $\mathsf{TrivAff}$s using the same randomness, i.e.,*

$$\mathcal{G} = \left\{\mathsf{RMul}_{\boldsymbol{\alpha}_1}^{\mathcal{R}_1}, \ldots, \mathsf{RMul}_{\boldsymbol{\alpha}_{\ell_1}}^{\mathcal{R}_1}, \mathsf{RAdd}_{\boldsymbol{\beta}_1}^{\mathcal{R}_2}, \ldots, \mathsf{RAdd}_{\boldsymbol{\beta}_{\ell_2}}^{\mathcal{R}_2}, \mathsf{TrivAff}_1, \ldots, \mathsf{TrivAff}_{\ell_3}\right\},$$

where $\mathcal{R}_1 = \{\boldsymbol{R}_x, \boldsymbol{R}_y, \boldsymbol{R}_z, \boldsymbol{R}\}$ and $\mathcal{R}_2 = \{\boldsymbol{R}_x, \boldsymbol{R}_y, \boldsymbol{R}_z\}$. Any composition of the gadgets in $\mathcal{G}$ is RNI-f_b, if the matrix $[\boldsymbol{\alpha}_1; \ldots; \boldsymbol{\alpha}_{\ell_1}; \boldsymbol{\beta}_1; \ldots; \boldsymbol{\beta}_{\ell_2}]$ is MDS.

4.2 How to Protect Any Circuit Using New Gadgets?

By the construction of Reed-Solomon code [RS60], an $\ell \times d$ MDS matrix over $\mathbb{F}_q$ always exists as long as $\ell + d \le q$. Thus, for any circuit over $\mathbb{F}_q$ consisting of ℓ_1 multiplications, ℓ_2 additions and an arbitrary number of affine operations such that $\ell_1 + \ell_2 \le q - d$, we can adopt Corollary 1 and transform the operations to $\mathsf{RMul}s$, $\mathsf{RAdd}s$ and $\mathsf{TrivAff}s$ respectively. That is, we can build a masked implementation for any function with intermediate variables in $\mathbb{F}_q$ containing ℓ_1 multiplications (over $\mathbb{F}_q$), ℓ_2 additions (over $\mathbb{F}_q$) and an arbitrary number of affine operations (defined in Section 2.3) such that $\ell_1 + \ell_2 + d \le q$. Meanwhile, it becomes a bit complicated when the total number of additions and multiplications is greater than $q - d$, where Corollary 1 cannot be adopted directly. To cope with such an issue, we present two strategies as follows.

Strategy 1. We can partition a circuit over $\mathbb{F}_q$ into several sub-circuits such that the total number of additions and multiplications of each sub-circuit is smaller than $q - d$. Then, we can protect each sub-circuit by transforming the operations to $\mathsf{RAdd}s$, $\mathsf{RMul}s$ and $\mathsf{TrivAff}s$, in which the randomness is reused across the gadgets in the same sub-circuit. By Corollary 1, any composition of gadgets in the same sub-circuits is RNI-f. Then, we use independent randomness for the gadgets of different sub-circuits, and thus Lemma 7 can be adopted. That is, for any function with intermediate variables in $\mathbb{F}_q$ containing

ℓ_1 multiplications, ℓ_2 additions and an arbitrary number of affine operations (given in Section 2.3) such that $\ell_1 + \ell_2 + d > q$, one can partition the circuit into $\lceil \frac{d+\ell_1+\ell_2}{q} \rceil$ sub-circuits and adopt Corollary 1 and Lemma 7. The complexity of multiplication or addition operation over $\mathbb{F}_q$ are $O(\log^2 q)$ and $O(\log q)$ resp.. This gives Corollary 2, where the gadgets corresponding to the same sub-circuit use the same randomness.

Corollary 2. *Any function with intermediate variables in $\mathbb{F}_q$ containing ℓ_1 multiplications, ℓ_2 additions and an arbitrary number of affine operations can be transformed to a d-private circuit with computational complexity $O\big(\lceil \frac{d+\ell}{q} \rceil \ell d^2 \log^2 q\big)$ and using $O\big(\lceil \frac{d+\ell}{q} \rceil d^2\big)$ random bits, where $\ell \stackrel{def}{=} \ell_1 + \ell_2$.*

Strategy 2. We consider the more generalized case of any Boolean circuit of size ℓ (containing ℓ AND and XOR gates)[6]. We can map each variable in $\mathbb{F}_2$ to a field element in $\mathbb{F}_{2^m}$ with $m = \lceil \log(\ell + d) \rceil$, such that $0 \in \mathbb{F}_2$ is mapped to $0 \in \mathbb{F}_{2^m}$ and $1 \in \mathbb{F}_2$ is mapped to an arbitrary nonzero element in $\mathbb{F}_{2^m}$. We also have $2^m \geq \ell + d$. Then, we can replace the AND and XOR gates with $\mathsf{RMul}s$ and $\mathsf{RAdd}s$ with the same randomness, requiring $4d^2 \log(\ell + d)$ random bits and ℓ gadgets, where each gadget contains $O(d^2 \log^2(\ell + d))$ operations over $\mathbb{F}_{2^m}$, and the complexity of multiplication and addition operation over $\mathbb{F}_{2^m}$ are $O(m^2)$ and $O(m)$ resp.. This gives Theorem 3.

Theorem 3. *Any Boolean circuit of size ℓ can be transformed to a d-private circuit with computational complexity $O\big(\ell d^2 \log^4(\ell + d)\big) = \tilde{O}(\ell d^2)$ and using $O\big(d^2 \log(\ell + d)\big) = \tilde{O}(d^2)$ random bits.*

4.3 Towards Precomputation-based Design Paradigm

In this sub-section, for the sake of brevity, we consider the composition of a set $\mathcal{G}$ of gadgets consisting of $\mathsf{TrivAff}s$, $\mathsf{RMul}s$ and $\mathsf{RAdd}s$ over $\mathbb{F}_q$ using the same randomness, i.e.,

$$\mathcal{G} = \big\{\mathsf{RMul}^{\mathcal{R}_1}_{\boldsymbol{\alpha}_1}, \ldots, \mathsf{RMul}^{\mathcal{R}_1}_{\boldsymbol{\alpha}_{\ell_1}}, \mathsf{RAdd}^{\mathcal{R}_2}_{2,\boldsymbol{\beta}_1}, \ldots, \mathsf{RAdd}^{\mathcal{R}_2}_{2,\boldsymbol{\beta}_{\ell_2}}, \mathsf{TrivAff}_1, \ldots, \mathsf{TrivAff}_{\ell_3}\big\},$$

where $\mathcal{R}_1 = \{\boldsymbol{R}_x, \boldsymbol{R}_y, \boldsymbol{R}_z, \boldsymbol{R}\}$, $\mathcal{R}_2 = \{\boldsymbol{R}_x, \boldsymbol{R}_y, \boldsymbol{R}_z\}$. By Corollary 1, any composition of gadgets in $\mathcal{G}$ is RNI-f_b with f_b the block-scale function if the matrix $[\boldsymbol{\alpha}_1; \ldots; \boldsymbol{\alpha}_{\ell_1}; \boldsymbol{\beta}_1; \ldots; \boldsymbol{\beta}_{\ell_2}]$ is MDS. In the following, we first discuss the precomputation-based design paradigm when $\ell_1 + \ell_2 + d \leq q$ and $[\boldsymbol{\alpha}_1; \ldots; \boldsymbol{\alpha}_{\ell_1}; \boldsymbol{\beta}_1; \ldots; \boldsymbol{\beta}_{\ell_2}]$ being MDS. The masked computation of the cryptographic algorithm can be divided into the following two phases.

Precomputation phase. It precomputes the intermediate variables that are only determined by the randomness, such as those highlighted in the instructions of $\mathsf{RNIRefresh}$ and SubMul (described in Gadgets 3 and 4). This phase can be performed before input (e.g., the plaintext or ciphertext for encryption or decryption resp.) reaches the cryptographic device. Obviously, the computational complexity of the precomputation phase is $O(\ell d^2 \log^2 q)$ with $\ell \stackrel{\text{def}}{=} \ell_1 + \ell_2$. Meanwhile, for practicality, we have to reduce the RAM space required to store the precomputed intermediates as much as possible.

Online phase. It computes the output from input sharings. Thanks to the intermediates precomputed in the precomputation phase, the online phase can be quite efficient.

The composition is made up of $\mathsf{RMul}s$, $\mathsf{RAdd}s$ and $\mathsf{TrivAff}s$, and each input sharing of these gadgets is either the input sharing of the composed gadget or linked to those gadgets in $\mathcal{G}$. By Lemmas 14 and 15, the last d shares of output sharings of $\mathsf{RNIRefresh}es$ and $\mathsf{SubMul}s$ are determined only by the randomness and can be precomputed. To simplify the discussion, we assume that the shares with indices $[1 : d]$ of input sharings of the composed

[6]It should be noted that, any circuit over $\mathbb{F}_q$ of size ℓ can be represented as a Boolean circuit of size $O(\lceil \log_2 q \rceil^2 \ell)$.

gadget can be precomputed. For each TrivAdd or TrivAff, the output $\hat{\boldsymbol{z}}[1:d]$ is determined by shares with indices in $[1:d]$ of input sharing. Therefore, shares with indices in $[1:d]$ of input sharings of RMul$s$, RAdd$s$ or TrivAff$s$ can be precomputed. In the following, we discuss the ingredients of RNIRefresh, SubMul and TrivAff separately.

- RNIRefresh: By the construction of RNIRefresh, we can precompute and store intermediate t and outputs $\hat{\boldsymbol{z}}[1:d]$ (and we don't need to store the intermediate $\boldsymbol{r}$). Then, the online phase of RNIRefresh is only at line 4, which is extremely efficient and runs in $O(\log q)$.

  We calculate the RAM space for storing the precomputed variables for all RNIRefreshes in $\mathcal{G}$ as follows. First, there are three random matrices (i.e., $\boldsymbol{R}_x$, $\boldsymbol{R}_y$ and $\boldsymbol{R}_z$) used in RNIRefreshes. As the values of $\hat{\boldsymbol{z}}[1:d]$ are determined by only the random matrices, we need to store outputs $\hat{\boldsymbol{z}}[1:d]$, requiring $3d$ variables in $\mathbb{F}_q$. Second, there are three RNIRefreshes in each RMul or RAdd, and thus we need to store 3ℓ variables in $\mathbb{F}_q$ for intermediate t. In total, the RAM space required to store precomputed variables is $(3d + 3\ell)\log q$ bits.

- TrivAff: A TrivAff can be either an output sub-gadget (in the composed gadget) or only linked to inputs of RNIRefreshes, and we discuss the two cases separately as follows.

  - If it is an output sub-gadget, we can precompute and store the output shares $\hat{\boldsymbol{z}}[1:d]$ in RAM. Then, the online phase of this gadget runs in $O(\log^2 q)$.
  - If it is only linked to inputs of RNIRefresh, we can precompute the output shares $\hat{\boldsymbol{z}}[1:d]$ but do not need to store it. It is because that $\hat{\boldsymbol{z}}[1:d]$ only affects the intermediates t in RNIRefreshes that links to the output of this gadget, which can be precomputed as well. Then, the online phase of this gadget runs in $O(\log^2 q)$ as well.

  Therefore, the RAM space for precomputed variables is $d\ell' \log q$ bits with ℓ' the number of TrivAff$s$ that are the output sub-gadget (in the composed gadget).

- SubMul: By the instruction of SubMul, we can precompute and store the vector $\boldsymbol{r}$ in RAM. Then, the online phase of SubMul is quite efficient and runs in $O(\log^2 q)$. The space for storing the precomputed variables in $\mathbb{F}_q$ for all SubMul$s$ in $\mathcal{G}$ is $d\ell_1$.

To summarize, the composed gadget runs in complexity $O(\ell d^2 \log^2 q)$ in the precomputation phase and stores $3d + 3\ell + d\ell_1 + d\ell' = O(\ell d \log q)$ bits in RAM. And, it runs with complexity $O(d\ell_1 + \ell') = O(\ell d \log^2 q)$ in the online phase, where ℓ' is the number of output TrivAff$s$.

For the more generalized case of Boolean circuits of size ℓ, we can adopt strategy 2 in Section 4.2 and replace the AND and XOR gates by RMul$s$ and RAdd$s$ respectively, using the same randomness. It provides d order probing security. The transformed circuit contains ℓ gadgets in $\{\text{RMul}s, \text{RAdd}s\}$ over $\mathbb{F}_{2^m}$ and requires $\leq 4d^2 \log(\ell + d)$ random bits, where $m = \lceil \log(\ell + d) \rceil$. Therefore, in the precomputation phase, the transformed circuit runs in time $\tilde{O}(\ell d^2)$ and stores $\tilde{O}(\ell d)$ random bits in RAM. In the online phase, it runs with complexity $\tilde{O}(d\ell)$.

4.4 A Discussion on the Concrete Security

While our scheme is provable secure in the probing model (which can be regarded as the first necessary step to rule out a number of security issues such as the composability), we discuss the concrete security of our scheme against practical side-channel attacks in this sub-section.

The concrete security can be achieved by adopting the general reduction from probing model to noisy model. The reduction suffices that the noise of the leakage linearly increases with the circuit size. Intuitively, an adversary can collect the (noisy) leakage of all intermediates, and the larger the circuit is, the more leakage will be. For the case of randomness/variables reuse, accessing the same value in the offline phase may cause multiple leaks, increasing the possibility of horizontal attacks & decreasing practical security with the low noise level. For the protection of cipher such as the AES, masking schemes secure in the probing model theoretically suffice a sufficient large level of noise. However, we emphasize that such an issue stems from the nature of probing security.

If we only consider the case without randomness reuse, the random probing model (requiring more shares and randomness) [AIS18, BCP+20] might be a good alternative security notion to theoretically overcome the multiple leakages issue. It was stated in [BCP+20]: "this notion advantageously captures the horizontal attacks which exploit the repeated manipulations of variables throughout the implementation." On the contrary, the security of the probing model requires sufficient noise that should grow linearly with the order of probing security [BCPZ16]. However, for the case with randomness reuse, our intuition is that the multiple leakages always exists, making masking with common shares secure in the random probing model almost impossible. In this respect, our scheme can be regarded as the investigation on how efficient the masking in the probing model can be when the noise is sufficient.

5 Application to AES-128

5.1 Implementation Approach

We show an application of our masking to the AES-128 block cipher that is performed on 16 variables in $\mathbb{F}_{2^8}$. In every round of AES-128, four types of transformations are performed: AddRoundKey, SubBytes, ShiftRows and MixColumns (see [DR02] for more details). ShiftRows and MixColumns are linear transformations over $\mathbb{F}_{2^8}^{16}$ that can be represented by a number of XOR operations, and a round key is XORed to the state in AddRoundKey. In the SubBytes transformation, a nonlinear function $\mathbb{F}_{2^8} \to \mathbb{F}_{2^8}$ called S-box is computed over each of the 16 variables of the state. The S-box is a composition of an inverse in $\mathbb{F}_{2^8}$ and an affine operation over $\mathbb{F}_2^8$. Note that, each RMul is a composition of Mul and $\mathsf{RNIRefresh}$, which are both RNI-f_b and use different random matrices (i.e., $\{\boldsymbol{R}_x, \boldsymbol{R}_y, \boldsymbol{R}\}$ and $\boldsymbol{R}_z$ for Mul and $\mathsf{RNIRefresh}$ resp.). This satisfies that every connection of gadgets ($\mathsf{Mul}s$ and $\mathsf{RNIRefresh}es$) crosses the two sets of gadgets using different random bits, and thus we can use Theorem 1 to prove that the masked S-box is RNI-f_b. Yet, a simpler proof is to use Theorem 2 (that directly considers RMul and RAdd).

We firstly focus on the field inverse. In [RP10], Rivain et al. proposed to represent the inverse by a power function $x \to x^{254}$, which can be further decomposed into a quite efficient chain of several multiplications and squaring functions, and then can be performed by applying our multiplication and trivial affine gadgets. We present our masked implementation in Figure 9, which is a composition of $\mathsf{TrivAff}s$ and $\mathsf{RMul}s$. Moreover, to apply Theorem 2, the concatenation of $\boldsymbol{\alpha}$ vectors of all gadgets should be an MDS matrix.

Similarly, the masked Mixcolumns is a linear transformation and can be implemented using $\mathsf{RAdd}s$ and $\mathsf{TrivAff}s$. The Mixcolumns of one round includes $3 \times 16 = 48$ $\mathsf{RAdd}s$. The ShiftRows transformation simply interchanges the sharings, and we do not need any gadgets to implement it. The AddRoundKey can be implemented directly by $\mathsf{RAdd}s$. Therefore the composition merely contains RMul and RAdd, and thus we can also use Theorem 2 to achieve to prove the RNI-f_b of the composed gadget.

In the following, we build an MDS matrix $\boldsymbol{A} \in \mathbb{F}_{2^8}^{\ell \times d}$, each row of which is used as the parameter vector in a distinct RMul or RAdd. We construct a matrix $\boldsymbol{A}' \in \mathbb{F}_{2^8}^{(\ell+d) \times d}$ that

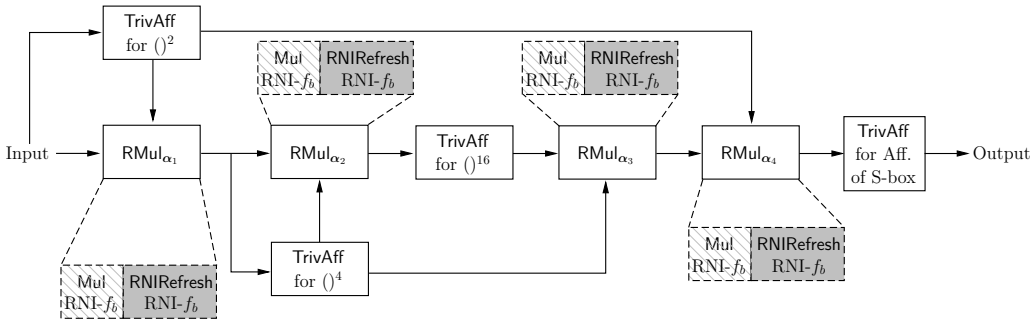


Figure 9: Masked implementation of AES S-box.

Table 1: The randomness requirement of AES implementation

AddRoundKey for 11 rounds	a fresh set of $\{\boldsymbol{R}_x, \boldsymbol{R}_y, \boldsymbol{R}_z, \boldsymbol{R}\}$
SubBytes and MixColumns for 2 rounds	a fresh set of $\{\boldsymbol{R}_x, \boldsymbol{R}_y, \boldsymbol{R}_z, \boldsymbol{R}\}$
Total randomness for full round	$(5+1)$ fresh sets of $\{\boldsymbol{R}_x, \boldsymbol{R}_y, \boldsymbol{R}_z, \boldsymbol{R}\}$

is the transpose of Vandermonde matrix over $\mathbb{F}_{2^8}$:

$$\boldsymbol{A}' = \begin{bmatrix} 1 & 1 & \dots & 1 \\ a_1 & a_2 & \dots & a_{\ell+d} \\ a_1^2 & a_2^2 & \dots & a_{\ell+d}^2 \\ \vdots & \vdots & \ddots & \vdots \\ a_1^{d-1} & a_2^{d-1} & \dots & a_{\ell+d}^{d-1} \end{bmatrix},$$

where $a_1 \dots a_{\ell+d}$ are distinct values in $\mathbb{F}_{2^8}$ (therefore, the construction is limited to $\ell + d \leq 2^8$). Since $\boldsymbol{A}'$ is the generating matrix of a Reed-Solomon code of length $\ell + d$ and dimension d, and by the property of Reed-Solomon code [RS60], a code with $\boldsymbol{A}'$ as the generating matrix is an MDS code, and thus its minimal distance is d. Then, an MDS matrix $\boldsymbol{A} \in \mathbb{F}_{2^8}^{\ell \times d}$ can be built by:

$$\boldsymbol{A} = \boldsymbol{A}'\Big[\big((d+1):(\ell+d)\big), :\Big] \times \boldsymbol{A}'^{-1}\big[(1:d), :\big] \ .$$

We hereafter discuss the randomness usage in the implementation. By our construction of the MDS matrix, there always exists an $\ell \times d$ MDS matrix over $\mathbb{F}_q$ as long as $\ell + d \leq q$. The AES cipher is performed over the field $\mathbb{F}_{2^8}$, making $q = 2^8$. For practicality, we consider the security order $d \leq 32$, making a set of random matrices $\{\boldsymbol{R}_x, \boldsymbol{R}_y, \boldsymbol{R}_z, \boldsymbol{R}\} \in \{\mathbb{F}_q^{d\times d}, \mathbb{F}_q^{d\times d}, \mathbb{F}_q^{d\times d}, \mathbb{F}_q^{d\times d}\}$ to be used for at most $256 - 32 = 224$ gadgets in $\{\mathsf{RMul}, \mathsf{RAdd}\}$. One round of AES involves $16 \times 4 + 48 = 112$ gadgets in $\{\mathsf{RMul}, \mathsf{RAdd}\}$ for SubBytes and MixColumns and 16 $\mathsf{RAdd}s$ for AddRoundKey.

Obviously, a full round masked AES requires multiple sets of random matrices $\{\boldsymbol{R}_x, \boldsymbol{R}_y, \boldsymbol{R}_z, \boldsymbol{R}\} \in \{\mathbb{F}_q^{d\times d}, \mathbb{F}_q^{d\times d}, \mathbb{F}_q^{d\times d}, \mathbb{F}_q^{d\times d}\}$, and we adopt strategy 1 in Section 4.2. As show in Table 1 Our implementation generates one set of random matrices for $\mathsf{RAdd}s$ of AddRoundKey in 11 rounds (since $16 \times 11 + d \leq 240$), and generates one set of random matrices for the 112×2 gadgets in $\{\mathsf{RMul}, \mathsf{RAdd}\}$ SubBytes and MixColumns in two rounds (since $112 \times 2 = 224$). We can easily prove the RNI-f_b of the implementation by Lemma 7 and Corollary 1. In total, it requires $(5+1) \times 4 \times d^2 = 24d^2$ bytes of randomness.

It should be noted that, the choice of the MDS matrix may impact the performance of the implementation, depending on the cost of the multiplication between the MDS

matrix and any vectors. This opens the possibility of the performance optimization of our masking by finding an MDS matrix with more efficient multiplication. But, we do not investigate it in this paper, and leave this topic as an interesting future work.

5.2 Implementation Results

In the rest of this section, we showcase the advantage of our scheme in the precomputation-based design paradigm, where the performance of the online phase plays an important role and should be taken into more consideration. We consider the security orders $d \in [4:16]$ and the ARM Cortex M architecture. For the consistency with the state-of-the-art results, the randomness in our implementations can be obtained from a True Random Number Generators (TRNGs) outputing 32 bits of random in 6000 cycles, which is recommended in [CGZ20]. For the comparison with the state-of-the-art implementations, we consider the results reported in [GR17, BGR18, CRZ18] as the benchmarks for Rivain and Prouff's S-box decomposition (shorted as R.-P.) method, Bitsliced (shorted as BS) method and look-up table (shorted as LUT) method of masking implementations.

Our implementation follows the strategy described in Section 4.3, and requires $3840 + 655d$ bytes of RAM to store the precomputed intermediates. Besides, the parameters (i.e., $\boldsymbol{\alpha}$ vectors) used in $\{\mathsf{RAdd}, \mathsf{RMul}\}$ should be stored in RAM as well, requiring $240d$ bytes. In total, our implementation requires $3840 + 895d$ bytes of RAM space for the precomputation.

As reported in [BGR18], the AES S-boxes for one round with the tight private circuits run in $290.5(d+1)^2 + 910.5(d+1) + 872$ cycles and requires $256d(d+1)$ random bits[7]. However, the authors of [BGR18] only reported the running cycles for S-boxes, we give a quite conservative estimation by only considering the running time of S-boxes. According to the timing divergence between S-boxes and the full AES reported in [GR17], the running time of the full AES should increase approximately by 30%, 20%, and 5% for security orders 4, 8 and 16 respectively.

To achieve SNI security, the scheme in [CRZ18] suffices to refresh the shares $2^k d^2/2$ bits for a single S-box with $k = 8$. In total, it requires $80 \times 2^k d^2$ bits for all the S-boxes of the full AES-128.

To comply with the precomputation design paradigm and accelerate the online phase, the masking approaches in [GR17, BGR18, CRZ18] can at best generate all the random bits and store in RAM, resulting in a large RAM requirement (that exhibits a quadratic growth in the security order). And, the online phase runs the masked operations with the pre-generated random bits. Indeed, one can also generate the randomness one-the-fly, but the cycles of online computation will largely grow by those of randomness generating.

The performance results are summarized in Table 2[8]. First and foremost, our implementation gains a significant speed-up in the online phase, and the RAM requirement (that exhibits a linear growth to the security order) is much smaller, making the scheme more practical in many embedded platforms. Our implementation also saves significantly for the generation of random bits. For instance, it saves clock cycles by a factor of 9.6 when $d = 4$. Indeed, our implementation includes a relatively heavy precomputation phase. It mainly stems from the multiplication over $\mathrm{GF}(2^8)$, which is not directly supported in microprocessors and can only rely on precomputed tables. This issue has been pointed out in [WGS+20] as well (and can be significantly mitigated in hardware implementation, where the field multiplication can be optimized on bit-level). Thus, the performance of the offline phase can be much more efficient for an instruction set extension supporting the multiplication over finite field with characteristic two.

[7] The result reported in [BGR18] was a function of the number of share t (since the number of reported bytes of s-boxes is $16t(t-1)$, indicating that $t \geq 2$), and we have $d + 1 = t$ in our case.

[8] The unprotected implementation is taken from https://github.com/kokke/tiny-AES-c.

It should be noted that the implementation is only provided for illustrative purposes (showing the practical relevance of the new masking). The implementation was done carefully as far as possible to avoid known implementation (but it is laborious to found out all possibilities of the transitional leakage[9] and thus we only check some obvious ones) issues caused by hardware architectural features and make the independent leakage assumption to be fulfilled [BDF+17, DFS15, CPR07]. For instance, to mitigate the issue caused by the nonlinear leakage of bits in a register (with Barrel shifter) reported recently in [GMPO20], our implementation always loads at most one share into a register. That is, we never pack multiple shares into one register. It indeed slows down the timings of our implementation, but we chose to be rigorous as far as possible for a fair comparison. Nevertheless, the number of cycles provided in this paper is still not quite "accurate" (as well as the state-of-the-art results we compared) for a particular platform or micro-architecture. Obtaining a secure implementation would require (at least) carefully examining the assembly code (mainly to find out the transitional leakage caused by the memory buffers), and perform a comprehensive security evaluation (following the suggestions in e.g., [WO19a, WO19b, BS20, MPW21]). In next sub-section, rather than a comprehensive security evaluation, we only perform a T-test evaluation to show the practical security order of masking, and report that the properties of our masking may benefit the security of implementation in practice.

5.3 Practical Evaluation of the Masked AES

To validate the security order of the masked AES in practice, we ran our implementation on a ChipWhisperer STM32F303 UFO target board, and acquired the power traces of the AES round function using Picoscope 5244D at 125 MS/s. Before performing a fixed vs. random Welch's T-test, we re-sample the trace by conducting a simple moving average pre-processing to decrease the noise. As the variables in the pre-computation phase are all independent of the secure input, we only consider the power consumption of the online phase. For each case we show in the rest of this section, the number of traces for fixed (resp., random) input is 5 0000.

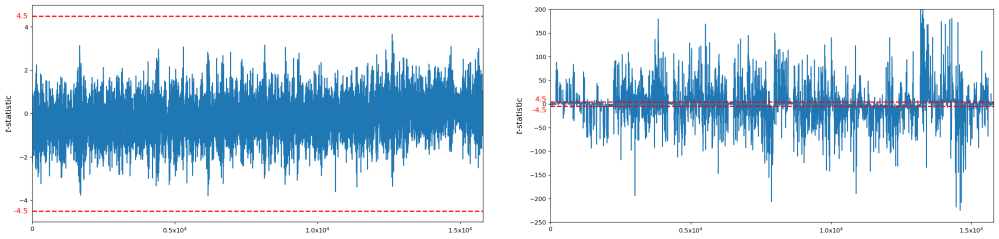
Figures 10(a) and 11(a) depicts the T-test results for security orders $d = 1$ and $d = 2$, where parameters $\boldsymbol{\alpha}$ are distinct nonzero values in $\mathbb{F}_{2^8}$. For comparison, we also provide in Figures 10(b) and 11(b) the results for the implementation when the randomness source is turned off. We can see that our implementation does not have first-order leakage. This (good) result for the case of $d = 1$ is a bit surprising, since, as stated in the previous sub-section, we do not attempt to eliminate all the transitional leakage that may damage the independent leakage assumption. We contribute this advantage to the relatively more complex algebraic structure (thanks to the values that are not ones of parameters for different gadgets) than the Boolean masking.

To confirm the intuition that using different parameters for different gadgets can overcome some lapses (such as transitional leakage) in implementation, we perform a fixed vs. random Welch's T-test with all parameters $[\boldsymbol{\alpha}_1; \boldsymbol{\alpha}_2; \ldots]$ are ones. Note that, when security order is $d = 1$, the parameter $\boldsymbol{\alpha}$ of each gadget is in $\mathbb{F}_q$, and thus $[\boldsymbol{\alpha}_1; \boldsymbol{\alpha}_2; \ldots]$ is MDS if and only if the elements are nonzero values. Hence, the theoretical security order ($d = 1$) holds for both $[\boldsymbol{\alpha}_1; \boldsymbol{\alpha}_2; \ldots]$ being ones (shown in Figure 12) and $[\boldsymbol{\alpha}_1; \boldsymbol{\alpha}_2; \ldots]$ being distinct nonzero values (shown in Figure 10(a)). However, when $[\boldsymbol{\alpha}_1; \boldsymbol{\alpha}_2; \ldots]$ are ones, the masking becomes Boolean masking, and the algebraic structure is less complex than the case using distinct nonzero values of $[\boldsymbol{\alpha}_1; \boldsymbol{\alpha}_2; \ldots]$. The T-test result in Figure 12 shows the existence of multiple points where the security order does not hold, due to the transitional leakage. It confirms that our masking scheme using parameters that are not ones is more

[9] The Hamming distance between the values before and after a change of a register is a typical transitional leakage the depends on two values.

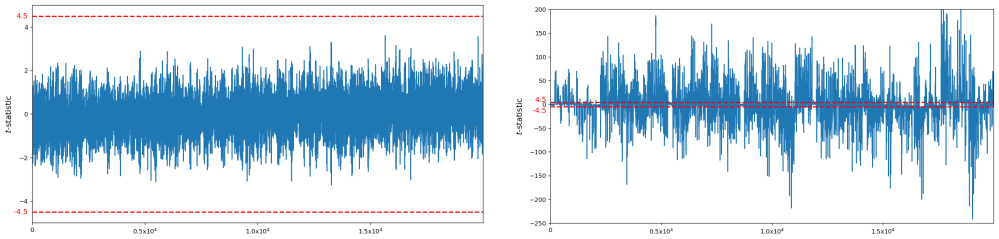
Table 2: Summary of masked AES implementations in the precomputation-based design paradigm (in kilos of clock cycles for timings).

	KCycles for precomp.		RAM for precomp.	KCycles/penalty factor for online	KCycles for total
	Rand. gen.	Cal.			
Unprotected	-	-	-	9.33 / 1	9.33
[BGR18] BS method $d = 2$	2 880	-	1.92 KB for all rand.	62 / 6.65	2 942
[CRZ18] LUT method $d = 2$	15 360	-	10.24 KB for all rand.	435 / 46.62	15 795
Our work $d = 2$	144	705	5.63 KB	**60** / 6.43	**909**
[BGR18] BS method $d = 4$	9 600	-	6.4 KB for all rand.	128 / 13.72	9 728
[CRZ18] LUT method $d = 4$	61 440	-	40.9 KB for all rand.	1 345 / 144.16	62 785
Our work $d = 4$	576	1 362	7.42 KB	**85** / 9.11	**2 023**
[BGR18] BS method $d = 8$	34 560	-	23.04 KB for all rand.	330 / 35.36	34 890
[CRZ18] LUT method $d = 8$	245 760	-	164 KB for all rand.	unreported	unreported
Our work $d = 8$	2 304	3 662	11 KB	**137** / 14.68	**6 103**
[BGR18] BS method $d = 16$	130 560	-	87.04 KB for all rand.	1 017 / 109	131 577
[CRZ18] LUT method $d = 16$	983 040	-	655 KB for all rand.	unreported	unreported
Our work $d = 16$	9 216	12 230	18.2 KB	**239** / 25.62	**21 685**



(a) Random number generator is on (b) Random number generator is off

Figure 10: Security order $d = 1$, parameters are distinct nonzero values.



(a) Random number generator is on (b) Random number generator is off

Figure 11: Security order $d = 2$.

robust to some lapses in implementation. As the matrix consisting of parameters requires to be MDS, there are many choices for parameters for an arbitrary security order. It will be interesting to investigate which parameters provide the best robustness, which we leave as promising future work.

Last but not least, we make the source codes of our AES round function available on https://github.com/wjwangcrypto/MaskingWithCommomShares.

6 Conclusion

In this paper, we continue the long line of works seeking to reduce the overhead of masking. In summary, the main contribution of this paper is two folds. First, we propose a new scheme with common shares and a new composable notion for the probing security proof. They contribute to pushing the limit of the complexity of masking schemes in the probing model reusing randomness / variables. Second, we present a new and practical design paradigm for masking where most of the intermediate variables can be precomputed only

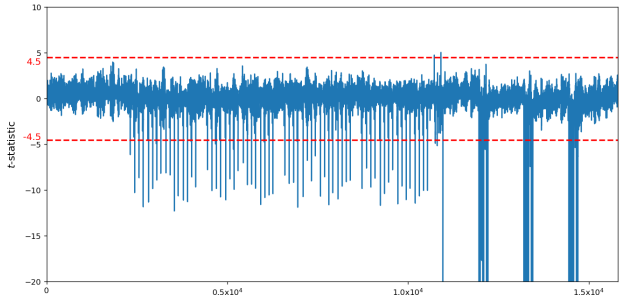


Figure 12: Security order $d = 1$, parameters are all 1s, random number generator is on.

by randomness, and the online computation (that takes inputs) can be quite efficient. The separation of precomputation and online computation can significantly lift masking schemes in many scenarios such as the challenge-response authentication protocols. Our scheme perfectly fits the new paradigm. For the AES on ARM Cortex M architecture, the results show the practical relevance of the new masking. A promising future work is to adapt the formal automated tools such as maskVerif [BBC+19] and SILVER [KSM20] to verify and optimize (by considering physical defaults such as the transitional leakage and Glitch) the implementation of our proposed gadget. Another promising future work should be applications to post-quantum cryptography. As our scheme is generic to any circuit over a finite field, we believe it is potentially interesting to provide a efficient masking scheme for post-quantum cryptographic algorithms.

Acknowledgments

The authors would like to thank the reviewers for their helpful comments and suggestions. This work was supported by the National Key Research and Development Program of China (Nos. 2021YFA1000600, 2020YFA0309705 and 2018YFA0704701), the Program of Qilu Young Scholars (Grant Nos. 61580089963177 and 61580082063088) of Shandong University, the Program of Taishan Young Scholars of the Shandong Province, the National Natural Science Foundation of China (Grant Nos. 62002202, 62002204, 62125204 and 61872236), the Shandong Nature Science Foundation of China (Grant No. ZR2020MF053), and the Major Program of Guangdong Basic and Applied Research (Grant No. 2019B030302008). Yu Yu also acknowledges the support from the XPLORER PRIZE.

References

[AIS18] Prabhanjan Ananth, Yuval Ishai, and Amit Sahai. Private circuits: A modular approach. In Hovav Shacham and Alexandra Boldyreva, editors, *Advances in Cryptology - CRYPTO 2018 - 38th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 19-23, 2018, Proceedings, Part III*, volume 10993 of *Lecture Notes in Computer Science*, pages 427–455. Springer, 2018.

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

[BBD+16] Gilles Barthe, Sonia Belaïd, François Dupressoir, Pierre-Alain Fouque, Benjamin Grégoire, Pierre-Yves Strub, and Rébecca Zucchini. Strong non-interference and type-directed higher-order masking. In *ACM CCS '16*, pages 116–129, 2016.

[BBP+16] Sonia Belaïd, Fabrice Benhamouda, Alain Passelègue, Emmanuel Prouff, Adrian Thillard, and Damien Vergnaud. Randomness complexity of private circuits for multiplication. In *Advances in Cryptology - EUROCRYPT 2016 - 35th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Vienna, Austria, May 8-12, 2016, Proceedings, Part II*, pages 616–648, 2016.

[BBP+17] Sonia Belaïd, Fabrice Benhamouda, Alain Passelègue, Emmanuel Prouff, Adrian Thillard, and Damien Vergnaud. Private multiplication over finite fields. In *CRYPTO 2017 (3)*, pages 397–426, 2017.

[BCP+20] Sonia Belaïd, Jean-Sébastien Coron, Emmanuel Prouff, Matthieu Rivain, and Abdul Rahman Taleb. Random probing security: Verification, composition, expansion and new constructions. In Daniele Micciancio and Thomas Ristenpart, editors, *Advances in Cryptology - CRYPTO 2020 - 40th Annual International Cryptology Conference, CRYPTO 2020, Santa Barbara, CA, USA, August 17-21, 2020, Proceedings, Part I*, volume 12170 of *Lecture Notes in Computer Science*, pages 339–368. Springer, 2020.

[BCPZ16] Alberto Battistello, Jean-Sébastien Coron, Emmanuel Prouff, and Rina Zeitoun. Horizontal side-channel attacks and countermeasures on the ISW masking scheme. In *CHES 2016*, pages 23–39, 2016.

[BDF+17] Gilles Barthe, François Dupressoir, Sebastian Faust, Benjamin Grégoire, François-Xavier Standaert, and Pierre-Yves Strub. Parallel implementations of masking schemes and the bounded moment leakage model. In *EUROCRYPT 2017 (1)*, pages 535–566, 2017.

[BFG15] Josep Balasch, Sebastian Faust, and Benedikt Gierlichs. Inner product masking revisited. In *Advances in Cryptology - EUROCRYPT 2015 - 34th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Sofia, Bulgaria, April 26-30, 2015, Proceedings, Part I*, pages 486–510, 2015.

[BGR18] Sonia Belaïd, Dahmun Goudarzi, and Matthieu Rivain. Tight private circuits: Achieving probing security with the least refreshing. In *ASIACRYPT 2018(2)*, pages 343–372, 2018.

[BS20] Olivier Bronchain and François-Xavier Standaert. Side-channel countermeasures' dissection and the limits of closed source security evaluations. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2020(2):1–25, 2020.

[CGPZ16] Jean-Sébastien Coron, Aurélien Greuet, Emmanuel Prouff, and Rina Zeitoun. Faster evaluation of sboxes via common shares. In Benedikt Gierlichs and Axel Y. Poschmann, editors, *Cryptographic Hardware and Embedded Systems - CHES 2016 - 18th International Conference, Santa Barbara, CA, USA, August 17-19, 2016, Proceedings*, volume 9813 of *Lecture Notes in Computer Science*, pages 498–514. Springer, 2016.

[CGZ20] Jean-Sébastien Coron, Aurélien Greuet, and Rina Zeitoun. Side-channel masking with pseudo-random generator. In Anne Canteaut and Yuval Ishai, editors, *Advances in Cryptology - EUROCRYPT 2020 - 39th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Zagreb, Croatia, May 10-14, 2020, Proceedings, Part III*, volume 12107 of *Lecture Notes in Computer Science*, pages 342–375. Springer, 2020.

[CJRR99] Suresh Chari, Charanjit S. Jutla, Josyula R. Rao, and Pankaj Rohatgi. Towards sound approaches to counteract power-analysis attacks. In crypto99ed, editor, *crypto99name*, volume crypto99vol of *mylncs*, pages 398–412, cryptoaddr, crypto99month 1999. cryptopub.

[Cor14] Jean-Sébastien Coron. Higher order masking of look-up tables. In Phong Q. Nguyen and Elisabeth Oswald, editors, *Advances in Cryptology - EUROCRYPT 2014 - 33rd Annual International Conference on the Theory and*

Applications of Cryptographic Techniques, Copenhagen, Denmark, May 11-15, 2014. Proceedings, volume 8441 of *Lecture Notes in Computer Science*, pages 441–458. Springer, 2014.

[CPR07] Jean-Sébastien Coron, Emmanuel Prouff, and Matthieu Rivain. Side channel cryptanalysis of a higher order masking scheme. In Pascal Paillier and Ingrid Verbauwhede, editors, *Cryptographic Hardware and Embedded Systems - CHES 2007, 9th International Workshop, Vienna, Austria, September 10-13, 2007, Proceedings*, volume 4727 of *Lecture Notes in Computer Science*, pages 28–44. Springer, 2007.

[CRZ18] Jean-Sébastien Coron, Franck Rondepierre, and Rina Zeitoun. High order masking of look-up tables with common shares. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2018(1):40–72, 2018.

[CS19] Gaetan Cassiers and François-Xavier Standaert. Towards globally optimized masking: From low randomness to low noise rate or probe isolating multiplications with reduced randomness and security against horizontal attacks. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2019(2):162–198, 2019.

[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.

[DDF14] Alexandre Duc, Stefan Dziembowski, and Sebastian Faust. Unifying leakage models: From probing attacks to noisy leakage. In *EUROCRYPT 2014*, pages 423–440, 2014.

[DFS15] Alexandre Duc, Sebastian Faust, and François-Xavier Standaert. Making masking security proofs concrete - or how to evaluate the security of any leaking device. In *Advances in Cryptology - EUROCRYPT 2015 - 34th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Sofia, Bulgaria, April 26-30, 2015, Proceedings, Part I*, pages 401–429, 2015.

[DR02] Joan Daemen and Vincent Rijmen. *The Design of Rijndael: AES - The Advanced Encryption Standard*. Information Security and Cryptography. Springer, 2002.

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

[FPS17] Sebastian Faust, Clara Paglialonga, and Tobias Schneider. Amortizing randomness complexity in private circuits. In *ASIACRYPT 2017 (1)*, pages 781–810, 2017.

[GMK16] Hannes Groß, Stefan Mangard, and Thomas Korak. Domain-oriented masking: Compact masked hardware implementations with arbitrary protection order. In *ACM TIS@CCS 2016*, page 3, 2016.

[GMPO20] Si Gao, Ben Marshall, Dan Page, and Elisabeth Oswald. Share-slicing: Friend or foe? *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2020(1):152–174, 2020.

[GR17] Dahmun Goudarzi and Matthieu Rivain. How fast can higher-order masking be in software? In *EUROCRYPT 2017(1)*, pages 567–597, 2017.

[GS18] Vincent Grosso and François-Xavier Standaert. Masking proofs are tight and how to exploit it in security evaluations. In *EUROCRYPT 2018(2)*, pages 385–412, 2018.

[IKL+13] Yuval Ishai, Eyal Kushilevitz, Xin Li, Rafail Ostrovsky, Manoj Prabhakaran, Amit Sahai, and David Zuckerman. Robust pseudorandom generators. In *ICALP 2013(1)*, pages 576–588, 2013.

[ISW03] Yuval Ishai, Amit Sahai, and David Wagner. Private circuits: Securing hardware against probing attacks. In *CRYPTO 2003*, pages 463–481, 2003.

[KR18] Pierre Karpman and Daniel S. Roche. New instantiations of the CRYPTO 2017 masking schemes. In *ASIACRYPT 2018(2)*, pages 285–314, 2018.

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

[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 *EUROCRYPT 2011*, pages 69–88, 2011.

[MPW21] Ben Marshall, Dan Page, and James Webb. MIRACLE: micro-architectural leakage evaluation. *IACR Cryptol. ePrint Arch.*, page 261, 2021.

[PR13] Emmanuel Prouff and Matthieu Rivain. Masking against side-channel attacks: A formal security proof. In *EUROCRYPT 2013*, pages 142–159, 2013.

[RBN+15] Oscar Reparaz, Begül Bilgin, Svetla Nikova, Benedikt Gierlichs, and Ingrid Verbauwhede. Consolidating masking schemes. In *Advances in Cryptology - CRYPTO 2015 - 35th Annual Cryptology Conference, Santa Barbara, CA, USA, August 16-20, 2015, Proceedings, Part I*, pages 764–783, 2015.

[RP10] Matthieu Rivain and Emmanuel Prouff. Provably secure higher-order masking of AES. In *CHES 2010*, pages 413–427, 2010.

[RS60] Irving S Reed and Gustave Solomon. Polynomial codes over certain finite fields. *Journal of the society for industrial and applied mathematics*, 8(2):300–304, 1960.

[WGS+20] Weijia Wang, Chun Guo, François-Xavier Standaert, Yu Yu, and Gaëtan Cassiers. Packed multiplication: How to amortize the cost of side-channel masking? In Shiho Moriai and Huaxiong Wang, editors, *Advances in Cryptology - ASIACRYPT 2020 - 26th International Conference on the Theory and Application of Cryptology and Information Security, Daejeon, South Korea, December 7-11, 2020, Proceedings, Part I*, volume 12491 of *Lecture Notes in Computer Science*, pages 851–880. Springer, 2020.

[WMCS20] Weijia Wang, Pierrick Méaux, Gaëtan Cassiers, and François-Xavier Standaert. Efficient and private computations with code-based masking. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2020(2):128–171, 2020.

[WO19a] Carolyn Whitnall and Elisabeth Oswald. A cautionary note regarding the usage of leakage detection tests in security evaluation. *IACR Cryptol. ePrint Arch.*, 2019:703, 2019.

[WO19b] Carolyn Whitnall and Elisabeth Oswald. A critical analysis of ISO 17825 ('testing methods for the mitigation of non-invasive attack classes against cryptographic modules'). In Steven D. Galbraith and Shiho Moriai, editors, *Advances in Cryptology - ASIACRYPT 2019 - 25th International Conference on the Theory and Application of Cryptology and Information Security, Kobe, Japan, December 8-12, 2019, Proceedings, Part III*, volume 11923 of *Lecture Notes in Computer Science*, pages 256–284. Springer, 2019.

Appendix

A ISW Multiplication Gadget

The ISW multiplication gadget, introduced in [ISW03], takes two sharings $(\hat{\boldsymbol{x}}, \hat{\boldsymbol{y}}) \in (\mathbb{F}_q^{d+1}, \mathbb{F}_q^{d+1})$ as inputs and computes the sharing $\hat{\boldsymbol{z}} \in \mathbb{F}_q^{d+1}$ as follows, such that $\sum \hat{\boldsymbol{z}} = \sum \hat{\boldsymbol{x}} \sum \hat{\boldsymbol{y}}$:

1. for every $0 \leq i < j \leq d$, pick uniformly at random a value $r_{i,j}$ over $\mathbb{F}_2$;
2. for every $0 \leq i < j \leq d$, compute $r_{j,i} \leftarrow (r_{i,j} + \hat{\boldsymbol{x}}[i]\hat{\boldsymbol{y}}[j]) \oplus \hat{\boldsymbol{x}}[j]\hat{\boldsymbol{y}}[i]$;
3. for every $0 \leq i \leq d$, compute $\hat{\boldsymbol{z}}[i] \leftarrow \hat{\boldsymbol{x}}[i]\hat{\boldsymbol{y}}[i] \oplus \sum_{j \neq i} r_{i,j}$.