Packed Multiplication: How to Amortize the Cost of Side-channel Masking?

Weijia Wang[1,2,3], Chun Guo[1,2,3], François-Xavier Standaert[4], Yu Yu[5,6] and Gaëtan Cassiers[4]

[1] School of Cyber Science and Technology, Shandong University, Qingdao 266237, China,
[2] Key Laboratory of Cryptologic Technology and Information Security of Ministry of Education, Shandong University, Qingdao, 266237, China
wjwang@sdu.edu.cn,chun.guo@sdu.edu.cn,
[3] State Key Laboratory of Information Security (Institute of Information Engineering, Chinese Academy of Sciences, Beijing 100093)
[4] Institute of Information and Communication Technologies, Electronics and Applied Mathematics (ICTEAM), UCLouvain, B-1348 Louvain-la-Neuve, Belgium
francois-xavier.standaert@uclouvain.be,gaetan.cassiers@uclouvain.be
[5] Department of Computer Science and Engineering, Shanghai Jiao Tong University, Shanghai, 200240
yuyu@yuyu.hk
[6] Shanghai Qizhi Institute, Shanghai, 200232

Abstract. Higher-order masking countermeasures provide strong provable security against side-channel attacks at the cost of incurring significant overheads, which largely hinders its applicability. Previous works towards remedying cost mostly concentrated on "local" calculations, i.e., optimizing the cost of computation units such as a single AND gate or a field multiplication. This paper explores a complementary "global" approach, i.e., considering multiple operations in the masked domain as a batch and reducing randomness and computational cost via amortization. In particular, we focus on the amortization of ℓ parallel field multiplications for appropriate integer $\ell > 1$, and design a kit named *packed multiplication* for implementing such a batch. For $\ell + d \leq 2^m$, when ℓ parallel multiplications over $\mathbb{F}_{2^m}$ with d-th order probing security are implemented, packed multiplication consumes $d^2 + 2\ell d + \ell$ bilinear multiplications and $2d^2 + d(d+1)/2$ random field variables, outperforming the state-of-the-art results with $O(\ell d^2)$ multiplications and $\ell \lfloor d^2/4 \rfloor + \ell d$ randomness. To prove d-probing security for packed multiplications, we introduce some weaker security notions for multiple-inputs-multiple-outputs gadgets and use them as intermediate steps, which may be of independent interest. As parallel field multiplications exist almost everywhere in symmetric cryptography, lifting optimizations from "local" to "global" substantially enlarges the space of improvements. To demonstrate, we showcase the method on the AES Subbytes step, GCM and TET (a popular disk encryption). Notably, when $d = 8$, our implementation of AES Subbytes in ARM Cortex M architecture achieves a gain of up to 33% in total speeds and saves up to 68% random bits than the state-of-the-art bitsliced implementation reported at ASIACRYPT 2018.

1 Introduction

Side-channel attacks that exploit leakage emitting from devices pose an important threat for cryptographic implementations. Masking [14, 26] is one of the most investigated protection techniques. The core idea is to randomly split each secret-dependent variable into a vector of $d+1$ shares called *sharing*, and then implements the cryptographic algorithm over sharings instead of the raw secrets. This ensures that the initial secret cannot be rebuilt from any less than d intermediate variables in the implementation, which is called *d-private security* (a.k.a. *d-probing security*).[7]

To have secure functionalities over sharings, a masking scheme, or a *private circuit*, firstly constructs *gadgets* for various elementary calculations over sharings, and then compose the gadgets to reach the desired functionality. Obviously, to improve efficiency, it is crucial to have better gadgets (particularly for multiplications). This has motivated plenty of works concentrating on e.g., reducing the randomness complexity [5, 6, 28, 12], and securing processing dependent inputs [20, 12].

Recently proposed masking schemes are typically accomplished by *formal proofs* of the aforementioned *d-private security* notion. To establish this notion, the naive method is to show that the possible tuples of intermediate variables are all independent of the secret by enumeration. Though, such an enumeration becomes intricate as the size of 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 the notions of d-Non-Inference (NI) and d-Strong Non-Inference (SNI) [2]. Thanks to those security notions, a composition of gadgets with some refreshing added in-between, can be proved to be globally d-private secure.

Besides the above foundational advances, the past two decades have also witnessed the rapid efficiency improvement of masking schemes. Despite these, higher-order masking with many shares remains of limited use due to the overhead, especially in the resource-constraint environment [27, 19]. *It is still compelling and challenging to decrease the complexity of masking schemes.*

Local versus Global efficiency optimization. As discussed before, the community has devoted to designing better gadgets [5, 6, 28, 12] due to their fundamental influences on the high-level circuits. In fact, to our knowledge, modulo a few exceptions that will be discussed later, most of the prior works only concentrated on "local" optimizations, i.e., on reducing the complexity of individual elementary calculation such as an S-box or even a single AND gate. This "local" approach considerably simplifies the situation and enables pushing the limits of gadgets. At the same time, by the aforementioned composition framework,

[7] While the leakages of all the $d+1$ shares enable reconstruction of information theoretically, the intrinsic noise in the leakages renders secret recovery infeasible in practice [14, 17, 21, 33].

this naturally results in high-level circuits with better performance and provable security.

On the other hand, note that cryptographic algorithms typically consist of executing a basic function for many times in parallel. For example, the AES (more generally, virtually all the block ciphers except for the so-called ARX designs) evaluates an S-box for 16 times within each round. And, at a higher level, many modes of operations are explicitly designed to support running several primitives in parallel. For instance, the Counter (CTR) mode encrypts several blocks in parallel, and the Galois/Counter Mode (GCM) combines the CTR mode with a structure consisting of several field multiplications in parallel.

Facing this situation, this paper takes a complementary "global" view, considers multiple such parallel functions as a batch, and seeks for optimizations within such batches. This switch enables many possibilities of improvements that used to be excluded in the classical "local" optimizations. In particular, the presence of multiple calculations naturally motivates using the *amortization* technique, which aims at *reducing the averaged complexity for the masking of several operations*.

While the idea of "global" optimization via amortizing appears natural, the technique of security proof is quite non-trivial. Particularly, due to amortization, various operations in the same batch now share randomness or intermediate variables, and thus cannot be analyzed independently. To cope with this difficulty, in our security analysis, we will treat parallel operations in the same batch as a whole, and consider the corresponding *gadgets with multiple input and output sharings (shorted as MIMO gadgets in the rest of the paper)*. This shift of viewpoint clearly excludes NI/SNI as the security goal. Informally speaking, any composition of d-NI and d-SNI gadgets is still d-SNI if each sharing is used at most once as input of any d-NI gadget and the input sharings of a gadget come from different gadgets. Designing secure circuits under this condition may requires many refresh gadgets, which are expensive. Therefore, new security notions for MIMO gadgets are required.

1.1 Our Contributions

We investigate global optimizations within batches of several field multiplications.[8] The concrete technique is to amortize the randomness and computational costs of several parallel masked multiplications. As a result, we propose a new construction named *packed multiplication*, which computes ℓ masked multiplications in parallel for any integer $\ell \geq 1$. Then, in order to prove security for our scheme, we introduce a new set of security notions for MIMO gadgets. We finally demonstrate potential applications and showcase the packed multiplication method on AES, Galois/Counter Mode (GCM) [30], and a popular disk encryption scheme TET (which is short for linear-Transformation; ECB; linear-Transformation) [23]. We details these contributions below.

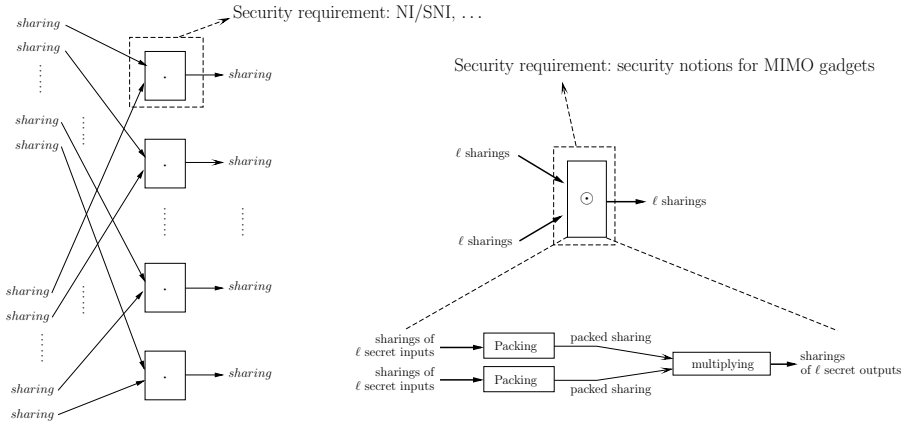
[8] Note that the AND gate can be viewed as the field multiplication in $\mathbb{F}_2$.

Packed Multiplication. To maximize the efficiency of linear gadgets, this paper concentrates on Boolean sharings (a.k.a. additive sharings) over the finite field $\mathbb{F}_q$ of characteristic 2, meaning that the XOR of the shares equals the initial secret. In this setting, a packed multiplication scheme takes two vectors of ℓ Boolean sharings as inputs, which encode the 2ℓ inputs of the ℓ field multiplications, and gives ℓ Boolean sharings as outputs encoding the ℓ multiplication results, as depicted in Figure 1 (right). Packed multiplication proceeds in two steps. First, each input vector is (re)encoded as a "packed" sharing using a randomized linear code. When the field size $q \geq \ell+d$, each resulted "packed" sharing consists of only $\ell+d$ shares in total, meaning that the size of data is compressed from $\ell(d+1)$ to $\ell+d$. Second, a multiplication over the packed sharings is calculated, resulting in Boolean sharings (the number of result shares is $\ell(d+1)$). This step can be seen as a batch of ℓ masked local multiplications *sharing* some randomness and intermediate results. Besides, our scheme is compatible even when the field size $q \leq \ell+d$, at the cost of raising the number of shares, say n, to $n > d+1$ with security order d, as long as the linear codes of length $\ell+n-1$ with dual distance $d+1$ exist.

In contrast, following the classical "local" approach, the two input vectors are viewed as ℓ pairs of sharings, and each of the ℓ pairs is processed independently, as shown in Figure 1 (left). As mentioned before, such independence simplifies security analysis at the expense of limiting optimizations to local. For a more complete comparison, we consider the setting of masking ℓ parallel multiplications, and list the complexities of packed multiplication and some other popular schemes in Table 1, where the complexity of our scheme is typical estimated when the field size $q \geq \ell + d$. In the comparison, we regard the number of bilinear multiplications (i.e., of general multiplications of two non-constant variables in the finite field) and the number of random elements as the metrics for computational [6] and randomness complexities respectively.

Towards Provable Security. Packed multiplication schemes produce MIMO gadgets. For their provable security, Cassiers el al. introduced a stronger variant of SNI named Multiple-Inputs / Multiple-Outputs Strong Non-Inference (MIMO-SNI) [13]. They also introduced Probe Isolating Non-Interference (PINI) [13] notion that enables the building of more efficient gadgets. Unfortunately, both MIMO-SNI and PINI are too strong and could not be achieved by ours. To rescue, we identify a set of intermediate composable security notions for MIMO gadgets that interpolates between the stronger MIMO-SNI and the weaker (S)NI. In addition, ours are orthogonal to PINI. We refer to Figure 2 for an illustration. With the new notions, our gadgets can be securely composed with each other, either by satisfying our specialized composition theorem, or through direct proof in the probe propagation framework introduced in [5, 11].

Applications. As parall multiplications exist almost everywhere in symmetric cryptography, our packed multiplication has potentially broad applicability and deep impact. To demonstrate, we showcase the method on the *AES Subbytes step* and *the polynomial-evaluation hash.*



(a) ℓ masked multiplications with isolating approach

(b) ℓ masked multiplications with our packed approach

Fig. 1. Packed multiplication in general (right) and the comparison with classical isolating approach (left).

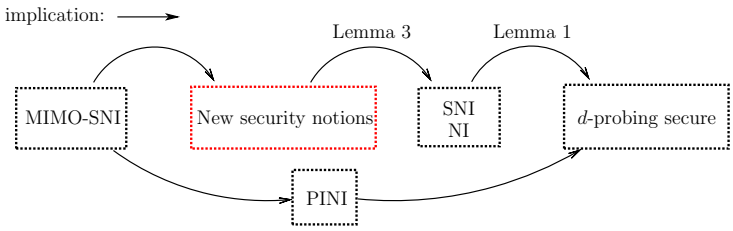


Fig. 2. Relations among different security notions.

Table 1. Complexities of ℓ parallel multiplications with security order d

		Computational complexity [1]	Randomness complexity [2]
Our Scheme	**Packing**	$\mathbf{0}$	$\boldsymbol{d^2}$
	Multiplying	$\boldsymbol{d^2 + 2\ell d + \ell}$	$\boldsymbol{d(d+1)/2}$
	Total	$\boldsymbol{d^2 + 2\ell d + \ell}$	$\boldsymbol{2d^2 + d(d+1)/2}$
Tight private circuits [26]		$\ell d^2 + 2\ell d + \ell$	$\ell d(d+1)/2$
Masking with reduced randomness [5]		$\ell d^2 + 2\ell d + \ell$	$\ell\big(d + \lfloor d^2/4 \rfloor\big)$
Multiplication over finite fields [*]	[6, Algorithm 4]	$2\ell d + \ell$	$\ell\big(2d^2 + d(d+1)/2\big)$
	[6, Algorithm 5]	$\ell d^2 + 2\ell d + \ell$	ℓd
Code-based masking [38]		$d^2 + 2\ell d + \ell^2$	$2d(d+\ell)$

[*] Despite the small instantiations for $d \leq 4$ [28], it requires large enough finite fields, e.g., the field size $q > d(d+1)(12d)^d$ [6, Theorem 5.4].

The AES Subbytes step consists of parallel S-boxes evaluations. Based on the ARM Cortex M architecture, we implement 16 AES S-boxes by applying the packed multiplication and report the performance results. Notably, when the security order is of $d = 8$, our implementation achieves a gain of up to 33% in speeds and saves up to 68% random bits compared with the state-of-the-art bitsliced implementation [9].

The polynomial-evaluation hash involves a structure of several multiplications in parallel, and thus our packed multiplication is well suited. This benefits the SCA resilience for two scenarios: GCM and TET.

1.2 Related works

Previous amortization. As mentioned before, global view and amortization were only considered in very few early works. Roughly, they fall into three concrete approaches, i.e., randomness re-use, masking with robust Pseudorandom Generator (PRG) and the code-based masking. The former two approaches aim at amortization of randomness rather than reducing computational cost, while the last addresses both.

Randomness re-use. This approach aims at re-using random bits in different gadgets. Faust et al. [18] introduce a security model allowing multiple gadgets to securely re-use randomness, and proposed threshold implementation-based gadgets in their model. This method provides a quite efficient scheme for small values of security order.

Masking with robust PRGs. Ishai et al. proposed to expand the randomness using the so-called robust PRG [25] in the private circuits, where the number of True Random Number Generator (TRNG) calls for seeds is independent of the circuit size. A recent work of Coron et al. describes a quite practical construction in this direction [15], where the number of random bits is only $\tilde{O}(t^2)$ for security against t probes. This strategy can be regarded as a certain form of amortization (of TRNG calls), but it is a bit of orthogonal to ours. In contrast, we consider the amortization of both randomness and computational costs.

Code-based masking. It was recently shown by Wang et al. [38] that the general 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. Admittedly, the packed multiplication proposed in this paper shares some ideas with the code-based masking. But we give a practical and much more efficient scheme. Notably, our scheme generally works with Boolean sharings, which enables more efficient masked linear operations. In contrast, the code-based masking proposed in [38] was a generic scheme, whose further specification and optimization were left as an open problem. We give a complexity comparison in Table 1 to highlight the improvement of our scheme.

Polynomial masking with packed secret sharing technique [22] can be regarded as a variant of the code-based masking, and its multiplications are performed based on the MPC protocol of Damgård et al. [16]. This scheme however

requires a heavy random generation process that becomes an efficiency bottleneck.

Security notions for MIMO gadgets. Cassiers el al. introduced a stronger variant of SNI named Multiple-Inputs / Multiple-Outputs Strong Non-Inference (MIMO-SNI) [13].[9] Though, MIMO-SNI gadget comes at a higher complexity compared to the SNI ones. They also introduced the Probe Isolating Non-Interference (PINI) [13] notion that enables the building of more efficient gadgets. Informally speaking, a composition of multiple gadgets is d-PINI (resp., d-MIMO-SNI) if every gadget is d-PINI (resp., d-MIMO-SNI). In addition, d-PINI (resp., d-MIMO-SNI) alone implies the d-private security. Unfortunately, these two notions are both too strong for our new multiplication gadget. For this reason, we will propose in Section 3 a set of new security notions that bridge our new gadgets to the probing security.

1.3 Organization

In the remainder of this paper, we present notations and necessary notions in Section 2. We then introduce our new security notions in Section 3. We propose the packed multiplication in Section 4, and in Section 5 propose a construction of the linear operations that complies with the new security notions. Section 7 illustrates the applications.

2 Preliminary

2.1 Notations

In the following, we denote by $\mathbb{F}_q$ a characteristic 2 finite field, where $q = 2^m$ for any $m \geq 1$, and denote field elements by lower-case letters. We use $\oplus$ to denote plus over the finite field. For simplicity, we use $\sum$ for the summation over any fields or rings. For a natural number n we denote by $[n]$ the set of integers from 1 to n both included. Let calligraphies (e.g., $\mathcal{I}$) be sets, and $|\mathcal{I}|$ denote the length of the set $\mathcal{I}$. Let bold lower cases (e.g., $\mathbf{x}$) be the vectors over $\mathbb{F}_q^{|\mathbf{x}|}$, where $|\mathbf{x}|$ denotes the length of the vector, $\mathbf{x}[i]$ denotes the i^{th} element of vector $\mathbf{x}$, and $\mathbf{x}[i:j]$ denotes the vector made up of i^{th} to j^{th} elements of vector $\mathbf{x}$. Unless otherwise noted, we assume the vectors are row vectors in this paper, and the column vectors are denoted as $\mathbf{x}^{\mathrm{T}}$.

Let bold capital letters (e.g., $\mathbf{A}$) be the matrices in $\mathbb{F}_q^{r\times c}$ (or $r \times c$ matrix), for row and column counts being r and c respectively. $\mathbf{A}[i,j]$ denotes the element of $\mathbf{A}$ at i^{th} row and j^{th} column, $\mathbf{A}[i,]$ (resp., $\mathbf{A}[,i]$) denotes the i^{th} row (resp., column) of matrix $\mathbf{A}$, and $\mathbf{A}[i:j,]$ denotes the matrix made up of i^{th} to j^{th} rows of $\mathbf{A}$. Let $\mathbf{A}^{\mathrm{T}}$ denote the transpose of the matrix $\mathbf{A}$. For a $r \times c$ matrix $\mathbf{A}$ and

[9] The notion of MIMO gadgets shall be distinguished from MIMO-SNI: the former are gadgets with multiple input and output sharings, while the latter is a security model for MIMO gadgets.

a set $\mathcal{I} \subseteq [r]$ (resp., $\mathcal{J} \subseteq [c]$), $\mathbf{A}[\mathcal{I},]$ (resp., $\mathbf{A}[, \mathcal{J}]$) denotes the submatrix of $\mathbf{A}$ made up of the rows (resp., columns) indexed by $\mathcal{I}$ (resp., $\mathcal{J}$). For matrices $\mathbf{A}$ and $\mathbf{B}$, we denote their product as $\mathbf{A} \times \mathbf{B}$, or in short $\mathbf{AB}$ in non-ambiguous cases. Specifically, we use $\mathbf{O}^{r \times c}$ to denote the zero matrix in $\mathbb{F}_q^{r \times c}$ and $\mathbf{I}^n$ the identity matrix in $\mathbb{F}_q^{n \times n}$; to ease understanding, when there is no ambiguity, their superscripts will be omitted. For two matrices $\mathbf{A}$ and $\mathbf{B}$, $[\mathbf{A}, \mathbf{B}]$ is the concatenation of $\mathbf{A}$ and $\mathbf{B}$ by columns, and $[\mathbf{A}; \mathbf{B}]$ is the concatenation of $\mathbf{A}$ and $\mathbf{B}$ by rows. A set of n variables can be represented as $\{x_i\}_{i=1}^n \stackrel{\text{def}}{=} \{x_1, \ldots, x_n\}$, and this representation can be adopted for a set of vectors or matrices.

2.2 Private Circuits

We view a circuit C as a directed acyclic graph with gates and wires being vertices and edges respectively. We assume that 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. A random gate is a gate that puts a random variable in its output wire. Variables carried in the wires of a circuit C are called intermediate variables of C. A probe to a circuit is an intermediate variable whose value is assumed to be revealed to the adversary. For a circuit C with input $\mathbf{x} \in \mathbb{F}_q^\ell$, $\mathsf{C}(\mathbf{x})$ produces the output $\mathbf{y} \in \mathbb{F}_q^{\ell'}$ that we denote $\mathsf{C}(\mathbf{x}) \stackrel{\text{def}}{=} \mathbf{y}$. And for a set $\mathcal{P}$ of probes, $\mathsf{C}_{\mathcal{P}}(\mathbf{x})$ returns the values of the probes by feeding $\mathbf{x}$ as the input of C. We call a set (or vector) of variables (say, $\mathbf{x}$) over $\mathbb{F}_q$ independent of the other vector of variables $\mathbf{y}$ if $\Pr(\mathbf{x} = \alpha \mid \mathbf{y} = \beta) = \Pr(\mathbf{x} = \alpha)$ for any value α of $\mathbf{x}$ and any value β of $\mathbf{y}$, where the probability is taken over the random coins used to generate these vectors.

We begin by recalling the notion of sharings, the basis of masking. We also provide our new notion of *packed sharing*. It should be noted that, for the notion of sharing, we let the number of shares be n (rather than $d+1$) for compatibility (of any field sizes) reason. As mentioned in the introduction, our scheme is also compatible with a small field size (i.e., $q < \ell + d$ with ℓ parallel multiplications), at the cost of raising the number of shares in a sharing to $n > d+1$ with security order d.

Definition 1 (Sharing and packed sharing) *For a variable $x \in \mathbb{F}_q$, we say $\hat{\mathbf{x}} \in \mathbb{F}_q^n$ is a sharing of x if there exists an encoder $\mathsf{Enc} : (\mathbb{F}_q, \mathbb{F}_q^{n-1}) \to \mathbb{F}_q^n$, a decoder $\mathsf{Dec} : \mathbb{F}_q^n \to \mathbb{F}_q$ and $\mathbf{r} \in \mathbb{F}_q^{n-1}$ such that $\hat{\mathbf{x}} = \mathsf{Enc}(x, \mathbf{r})$ and $x = \mathsf{Dec}(\hat{\mathbf{x}})$. Particularly, a sharing $\hat{\mathbf{x}}$ of x is called a Boolean sharing, if $x = \mathsf{Dec}(\hat{\mathbf{x}}) = \sum_{i=1}^n \hat{\mathbf{x}}[i]$.*

For $\ell > 1$ and a vector of variables $\mathbf{x} \in \mathbb{F}_q^\ell$, we say $(\tilde{\mathbf{x}}, \hat{\mathbf{u}}) \in (\mathbb{F}_q^\ell, \mathbb{F}_q^{n-1})$ is a packed sharing of $\mathbf{x}$ if there exists an encoder $\mathsf{Enc} : (\mathbb{F}_q^\ell, \mathbb{F}_q^{n-1}) \to (\mathbb{F}_q^\ell, \mathbb{F}_q^{n-1})$, a decoder $\mathsf{Dec} : (\mathbb{F}_q^\ell, \mathbb{F}_q^{n-1}) \to \mathbb{F}_q^\ell$ and $\mathbf{r} \in \mathbb{F}_q^{n-1}$ such that $(\tilde{\mathbf{x}}, \hat{\mathbf{u}}) = \mathsf{Enc}(\mathbf{x}, \mathbf{r})$ and $\mathbf{x} = \mathsf{Dec}(\tilde{\mathbf{x}}, \hat{\mathbf{u}})$. Moreover, an element of a packed sharing $(\tilde{\mathbf{x}}, \hat{\mathbf{u}})$ is an element of either $\tilde{\mathbf{x}}$ or $\hat{\mathbf{u}}$.

Elements of a sharing or packed sharing are called shares.

In the rest of the paper, unless explicitly stated, all sharings are Boolean sharings. We next recall the notion of private circuit compiler [26] as follows.

Definition 2 (Private circuit compiler [26]) *A private circuit compiler for a circuit* C *with input in* $\mathbb{F}_q^{\ell}$ *and output in* $\mathbb{F}_q^{\ell'}$ *is defined by a triple* $(\mathsf{I}, \mathsf{T}, \mathsf{O})$ *where*

- $\mathsf{I} : \mathbb{F}_q \to \mathbb{F}_q^n$*, is an encoder that randomly maps each input* $x \in \mathbb{F}_q$ *to a sharing.*
- T *is a circuit transformation whose input is circuit* C*, and output is a randomized circuit* C' *with* ℓ *sharings as the input, and* ℓ' *sharings as the output.*
- $\mathsf{O} : \mathbb{F}_q^n \to \mathbb{F}_q$ *is a decoder that maps each output sharing* $\hat{\mathbf{z}} \in \mathbb{F}_q^n$ *to the corresponding output* $z \in \mathbb{F}_q$*.*

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* $\mathbf{x} \in \mathbb{F}_q^{\ell}$*,* $\Pr\Big(\mathsf{O}^{\circ}\big(\mathsf{C}'(\mathsf{I}^{\circ}(\mathbf{x}))\big) = \mathsf{C}\big(\mathbf{x}\big)\Big) = 1$*, where* I° *(resp.,* O°*) is a canonical encoder (resp., decoder) that encodes (resp., decodes) each element of input secrets* $\mathbf{x}$ *(resp., each sharing of output sharings) by repeatedly calling* I *(resp.,* O*).*
- *Privacy: for any input* $\mathbf{x} \in \mathbb{F}_q^{\ell}$ *and any set of probes* $\mathcal{P}$ *such that* $|\mathcal{P}| \leq d$*,* $\mathsf{C}'_{\mathcal{P}}\big(\mathsf{I}(\mathbf{x})\big)$ *are independent of the input* $\mathbf{x}$*, where* d *is called the security order.*

We consider the circuit transformation T realized by the composition of gadgets. An *gadget* is a randomized circuit whose inputs (resp., outputs) are either sharings or packed sharings. We say that a gadget G implements a function $f : \mathbb{F}_q^{\ell} \to \mathbb{F}_q^{\ell'}$, if and only if $\mathsf{O}^{\circ}\Big(\mathsf{G}\big(\mathsf{I}^{\circ}(\mathbf{x})\big)\Big) = f(\mathbf{x})$ for any $\mathbf{x} \in \mathbb{F}_q^{\ell}$, where I° (resp., O°) encodes (resp., decodes) each input (resp., output). 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. I.e., the resulted 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 or packed sharings. An *MIMO gadget* is a gadget with multiple input sharings or output sharings. Note that the composed gadget C' is a gadget, and thus a recursive composition of gadgets is also a gadget.

2.3 Composable security notions

While the notion of d-private security nicely protects against side-channel attacks, it is not trivial to prove that large circuits such the AES fulfill it. The difficulty stems from enumerating the probes within the circuit, 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, while the global d-private security is ensured by composition. Barthe et al. introduced first composable security notions [2] for (small) gadgets that are sufficient to result in provable probing security.

Simulatability. We first recall the definition of simulatability introduced in [5]:

Definition 3 (Simulatability [5]) *Let $\mathcal{P} = \{p_1, ..., p_d\}$ be a set of d 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 S: $\mathbb{F}_q^{|\mathcal{X}|} \to \mathbb{F}_q^d$. A distinguisher is a randomized function D: $\left(\mathbb{F}_q^d, \mathbb{F}_q^{|\mathcal{X}|}\right) \to \{0, 1\}$. The set of probes $\mathcal{P}$ can be simulated with shares in $\mathcal{S}$ if and only if there exists a simulator S such that for any distinguisher D and any inputs shares $\mathcal{X}$, we have:*

$$\Pr\left[\mathsf{D}\big(\mathsf{G}_{\mathcal{P}}(\mathcal{X}), \mathcal{X}\big) = 1\right] = \Pr\left[\mathsf{D}\big(\mathsf{S}(\mathcal{S}), \mathcal{X}\big) = 1\right] \ ,$$

where the probability is over the random coins in G, S and D.

(Strong) Non-Inference. We then recall the first set of composable security notions introduced in [2]. The probes of a gadget are separated as follows:

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

Definition 4 (d-(Strong) Non-Inference ((S)NI)[2]) *Let G be a gadget with sharings as inputs and outputs. G is d-NI (resp., d-SNI), if any probes consisting of t_{int} internal probes and t_{out} outputs probes with $t_{int} + t_{out} \leq d$ can be simulated with $t_{int} + t_{out}$ (resp., t_{int}) shares of each input sharing.*

As shown in Lemma 1, both d-NI and d-SNI imply the d-private security.

Lemma 1 (NI/SNI implies probing security [2]) *If a gadget G is d-SNI or d-NI, then G is a d-private circuit if any d shares in each input sharing are independent of the secrets and all input sharings are independently encoded.*

More importantly, in the proof of probing security, NI and SNI can reduce the elaboration from trying all tuples of probes of a full circuit to only verifying each small gadget. Informally speaking, any composition of d-NI and d-SNI gadgets is still d-NI if each sharing is used at most once as input of any d-NI gadget and the input sharings of a gadget come from different gadgets.

2.4 Different types of gadgets

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

The first type of gadgets is linear gadgets that implement linear functions. As the encoder is usually homomorphic (for example, the encoder of the Boolean sharing) over linear functions, linear gadgets can be correctly constructed by applying linear functions on the shares of the same index, which we will denote as *the trivial implementation of a linear function*. It becomes more difficult for (nonlinear) gadgets implementing nonlinear functions such as multiplication,

since the encoder is usually not homomorphic over nonlinear functions. The last type of gadgets is the refresh gadget (a.k.a, the refreshing) that re-randomizes a sharing, which is usually needed for the composition of gadgets. Existing works (e.g., [4, 3, 1]) have provided different refresh gadgets that are asymptotically more efficient than multiplication gadgets. In the rest of the paper, we mainly focus on a typical nonlinear gadget: multiplication gadget that implements the multiplication over $\mathbb{F}_q$ in the masked domain.

3 New security notions for MIMO gadgets

To motivate, this section begins by recalling the limitation of NI/SNI with MIMO gadgets. Then, to ease understanding, we serve intuition for our new security notions in sub-section 3.2. The core concept will be the notion of t-chunk that describes a set of shares from the input or output sharings of a gadget. The formal definitions are finally given in sub-section 3.3.

3.1 Limitation of NI/SNI with MIMO gadgets

The notions of NI and SNI are not perfectly suitable for MIMO gadgets. To see this, let's consider, for example, the compositions of two gadgets, as illustrated in Figure 3. In Figure 3-(a), the composition of G_1 and G_2 subjects to the rule in Lemma 1, and thus is 3-SNI. Figure 3-(b) shows an improper composition, where two probes (one internal and one output) of G_2 requires 4 input variables to simulate, which cannot be further simulate with the input of G_1 since G_1 is 3-SNI. Figure 3-(c) fixes the issue of Figure 3-(b) by adding SNI refreshings, which however comes at huge overheads. Note that a similar illustration can be found in [13, Figure 5], where the authors considered a linear operation between two outputs of a nonlinear gadget.

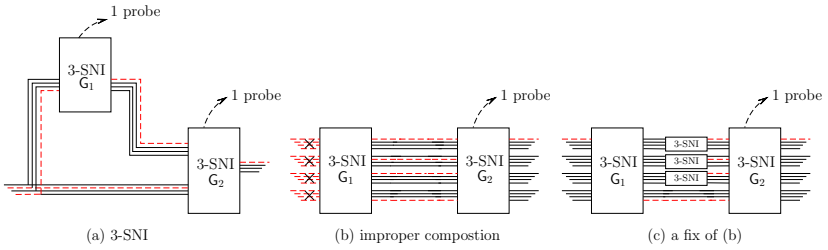


Fig. 3. Limitation of (S)NI.

In the rest of this section, we investigate more suitable security notions for gadgets with multiple input and output. However, for example, the packed multiplication that we will introduce in Section 4 is neither d-MIMO-SNI nor d-PINI, but is d-SNI. It indicates that there should exist some security notions between MIMO-SNI and SNI (stronger than SNI and weaker than MIMO-SNI) and more suitable to the packed multiplication. In this respect, we put forward a set of new security notions.

3.2 Intuition behind the new security notions

The notion of simulatability captures that a set of output shares and t_{int} internal shares can be simulated with some input shares called *propagated shares*. In this respect, how to define the output shares and the propagated shares is critical in different security notions. Let $\hat{\mathbf{x}}_1, \ldots, \hat{\mathbf{x}}_\ell$ be ℓ sharings that can be either input sharings or output sharings of a gadget. For an integer t, we define the types of set $\mathcal{X}$ consisting of some shares in $\hat{\mathbf{x}}_1, \ldots, \hat{\mathbf{x}}_\ell$ as follows:

i. $|\mathcal{X}| = t$, i.e., $\mathcal{X}$ consists of t shares in $\hat{\mathbf{x}}_1, \ldots, \hat{\mathbf{x}}_\ell$.
ii. $|\mathcal{X}| = \ell t$, and $\mathcal{X}$ consists of t shares in each sharing of $\hat{\mathbf{x}}_1, \ldots, \hat{\mathbf{x}}_\ell$.

It can be seen that, in (S)NI, output and propagated shares relate to types i and ii respectively. The only difference between SNI and NI is the values of the parameters t for output and propagated shares. And in MIMO-SNI, output and propagated shares relate to types ii and i respectively, which makes it a stronger property than (S)NI. It is because, compared with (S)NI, MIMO-SNI allows that a larger set of output shares can be simulated with a smaller set of propagated shares. Examples can be found in Figures 5-(a) (b) and (c).

For our new security notions, we introduce a new type of set $\mathcal{X}$ as follows:

iii. $\mathcal{X}$ consists of a *t-chunk* of $\hat{\mathbf{x}}_1, \ldots, \hat{\mathbf{x}}_\ell$, where the *t-chunk* is defined below, and we also depict an example in Figure 4.

Definition 5 *A t-chunk of sharings $\hat{\mathbf{x}}_1, \ldots, \hat{\mathbf{x}}_\ell \in \mathbb{F}_q^n, \ldots, \mathbb{F}_q^n$ is a subset of a set made up of the following two parts:*

1. *(α part) $\{\hat{\mathbf{x}}_k[i] \mid k \in \mathcal{K}, i \in \mathcal{I}\}$ for $\mathcal{K} \subseteq [\ell]$, $\mathcal{I} \subseteq [n]$ and $|\mathcal{K}| + |\mathcal{I}| = t_\alpha$.*
2. *(β part) t_β shares from $\hat{\mathbf{x}}_1, \ldots, \hat{\mathbf{x}}_\ell$.*

such that $t \geq t_\alpha + t_\beta$.

It should be noted that the t-chunk is only defined with sharings, rather than the packed sharings.

The rationale of the t-chunk definition. The t-chunk is defined in accordance with the formalism of packed multiplication given latter in Section 4. We will mostly consider an abstract computation that takes sharings $\hat{\mathbf{x}}_1, \ldots, \hat{\mathbf{x}}_\ell \in \mathbb{F}_q^n, \ldots, \mathbb{F}_q^n$ as inputs and sums (XOR) the rows of $\mathbf{X} \stackrel{\text{def}}{=} \hat{\mathbf{X}} \oplus \hat{\mathbf{Q}}$, resulting in $\tilde{\mathbf{x}} \in \mathbb{F}_q^\ell$, where $\hat{\mathbf{X}}[, k] \stackrel{\text{def}}{=} \hat{\mathbf{x}}_k$ for $k \in [\ell]$ and $\hat{\mathbf{Q}} \in \mathbb{F}_q^{n \times \ell}$ is a random matrix. During the process, there also exist variables $f(\hat{\mathbf{Q}}[i,])$ for any function $f : \mathbb{F}_q^\ell \to \mathbb{F}_q$ and any $i \in [n]$. A specification of such abstract algorithm is the packing in Gadget 1-P, and an example will be depicted in Figure 6. In this case, a certain amount of probes to $f(\hat{\mathbf{Q}}[i,])$, $\hat{\mathbf{X}}$, $\mathbf{X}$ and $\tilde{\mathbf{x}}$ can be simulated with a t-chunk of $\hat{\mathbf{x}}_1, \ldots, \hat{\mathbf{x}}_\ell$ for some $t \geq 0$. More concretely (but informally),

- Let $\mathcal{I} \subseteq [n]$ and $\mathcal{K} \subseteq [\ell]$, the probes to $f(\hat{\mathbf{Q}}[i,])$ for $i \in \mathcal{I}$ can be simulated by sampling the corresponding random distribution, and probes to $\tilde{\mathbf{x}}[k]$ for $k \in \mathcal{K}$ can be simulated with the α part of $\hat{\mathbf{x}}_1, \ldots, \hat{\mathbf{x}}_\ell$ corresponding to $\mathcal{I}$ and $\mathcal{K}$. In the example of Figure 4, the probes of this type relate to $\mathcal{I} = \{2, 3, 4\}$ and $\mathcal{K} = \{5, 6\}$.

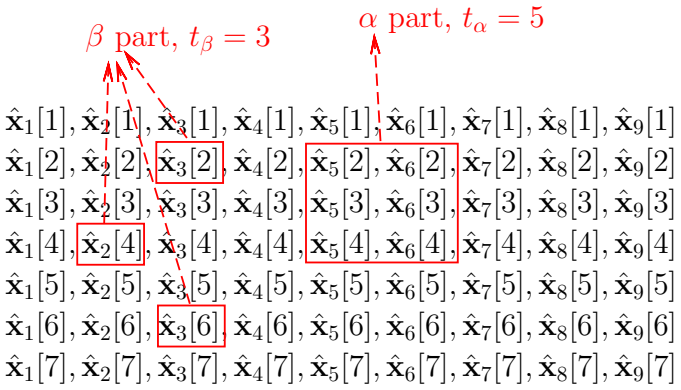


Fig. 4. An examples of an 8-chunk of sharings $\hat{\mathbf{x}}_1, \ldots, \hat{\mathbf{x}}_7$, where $t_\alpha = 5$ and $t_\beta = 3$. Each column of the matrix corresponds to a distinct sharing.

– The probes to $\hat{\mathbf{X}}$ can be simulated with the β part of sharings $\hat{\mathbf{x}}_1, \ldots, \hat{\mathbf{x}}_\ell$. In the example of Figure 4, the probes of this type are $\hat{\mathbf{X}}$[3,2], $\hat{\mathbf{X}}$[2,4] and $\hat{\mathbf{X}}$[3,6].

Below in Lemma 2, we show that the union of two t-chunks is a $2t$-chunk. Its proof is in the full version. This property enables merging several t-chunk probes.

Lemma 2 (Closure of t-chunk under union) *If $\mathcal{S}_1$ and $\mathcal{S}_2$ are t_1-chunk and t_2-chunk of sharings $\hat{\mathbf{x}}_1, \ldots, \hat{\mathbf{x}}_\ell$ respectively, then $\mathcal{S}_1 \cup \mathcal{S}_2$ is a $(t_1 + t_2)$-chunk of $\hat{\mathbf{x}}_1, \ldots, \hat{\mathbf{x}}_\ell$.*

Cautionary note. By definition, a subset of a t-chunk is also a t-chunk. Thus, a t-chunk should also be a t'-chunk for any $t' > t$. Moreover, the partition of $\mathcal{S}$ (into α and β parts) is not unique. For example, the set of share highlighted in Figure 4 can also be 9-chunk, if it is partitioned in a way that β part contains all the highlighted shares and α part is empty. Also note that, there always exists a minimum value of t for any set of shares. For example, the set of share highlighted in Figure 4 can not be t-chunk for any $t < 8$.

3.3 New security notion for MIMO gadgets

In this sub-section, we formally introduce the new security notions. We begin with the first one:

1. d-Chunk Strong Non-Inference and d-Chunk Non-Inference, abbreviated as *d-CNI and d-CSNI* respectively.

They share a similar structure with NI/SNI, but output and propagated shares are replaced with a t-chunk of the output and input sharings respectively, making them to be positioned in-between d-(S)NI and d-MIMO-SNI. The formal definition of d-C(S)NI is as follows.

Definition 6 (d-C(S)NI) *Let* G *be a gadget with sharings as inputs and outputs.* G *is d-CNI (resp., d-CSNI), if any probes consisting of* t_{int} *internal probes and a* t_{out}*-chunk of output sharings with* $t_{int} + t_{out} \leq d$ *can be simulated with a* $(t_{int} + t_{out})$*-chunk (resp.,* t_{int}*-chunk) of input sharings.*

In Figure 5-(a)(b)(c)(d), we give examples to illustrate the differences of the d-C(S)NI, (S)NI and MIMO-SNI. Also note that type *iii* shares cover type *i* shares with the same value of t, and thus, as shown in Lemma 3, d-C(S)NI implies the (S)NI security. The proof is given in the full version.

Lemma 3 *d-CNI* $\Rightarrow$ *d-NI, d-CSNI* $\Rightarrow$ *d-SNI and d-CSNI* $\Rightarrow$ *d-CNI.*

3.4 New security notion for gadgets with packed sharings

While the d-C(S)NI meets the minimal requirement for protecting any cryptographic algorithm, it is (by definition) only for gadgets with sharings as inputs and outputs, and thus incompatible with packed sharings. Such compatibility has the (obvious) advantage of enabling extension to any gadgets that are composed of packing, multiplying and linear gadgets. For example the masked AES S-box that we will present latter in Figure 9, Section 6. The security proof of such composition can be much simplified if there exist secure notions particularly for packing and multiplying gadgets, more generally, for gadgets with packed sharings as inputs or outputs.

Therefore, to facilitate the compositions for gadgets with packed sharings, two other new notions are necessary:

2. For the gadgets with input sharings and output packed sharings, we propose d-Input-Chunk Non-Inference and d-Input-Chunk Strong Non-Inference, abbreviated as d-ICNI and d-ICSNI respectively.
3. For the gadgets with input packed sharings and output sharings, we propose d-Output-Chunk Non-Inference and d-Output-Chunk Strong Non-Inference, abbreviated as d-OCNI and d-OCSNI respectively

The formal definitions are in Definitions 7 and 8. Also see Figure 5-(e)(f) for the corresponding illustrations.

Definition 7 (d-IC(S)NI) *Let* G *be a gadget with sharings as inputs and packed sharings as outputs.* G *is d-ICNI (resp., d-ICSNI), if any probes consisting of* t_{int} *internal probes and* t_{out} *shares from output packed sharings with* $t_{int} + t_{out} \leq d$ *can be simulated with a* $(t_{int} + t_{out})$*-chunk (resp.,* t_{int}*-chunk) of input sharings.*

Definition 8 (d-OC(S)NI) *Let* G *be a gadget with packed sharings as inputs and sharings as outputs.* G *is d-OCNI (resp., d-OCSNI), if any probes consisting of* d_{int} *internal probes and a* d_{out}*-chunk of output sharings with* $t_{int} + t_{out} \leq d$ *can be simulated with* $t_{int} + t_{out}$ *(resp.,* t_{int}*) shares of each input packed sharing.*

In Sections 4 and 5, we will propose constructions for d-CSNI and d-CNI packed gadgets that we will use in tailored analyzes of some relevant circuits

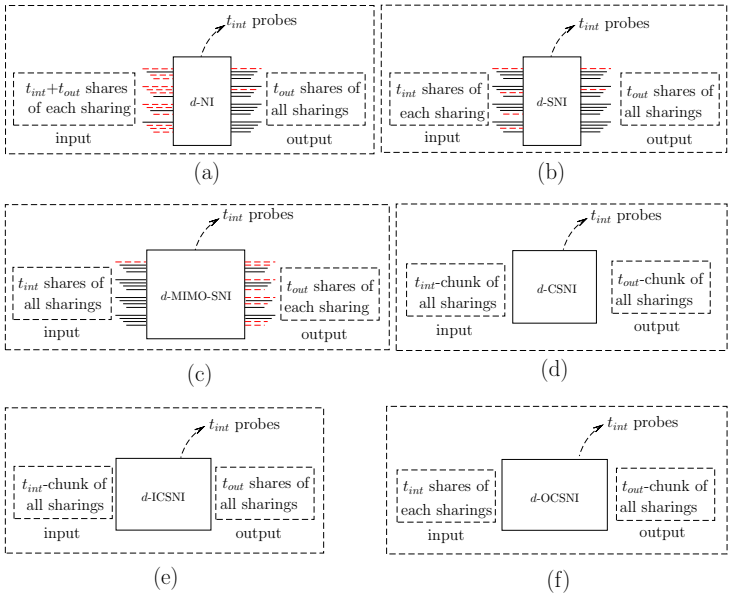


Fig. 5. Difference between the security notions.
(a) d-NI: t_{out} output probes and t_{int} internal probes can be simulated with propagated shares that consist of $t_{int} + t_{out}$ shares of each input sharing.
(b) d-SNI: output probes are the same as d-NI case, and the propagated shares consist of t_{int} shares of each input sharing.
(c) d-MIMO-SNI: the output probes consist of t_{out} shares of each output sharing, and the propagated share is only t_{int} input share of all input sharings.
(d) d-CSNI: output probes consists of a t_{out}-chunk of output sharings, and the propagated shares consist of a t_{int}-chunk of output sharings.
(e) d-ICSNI: output probes are shares from packed sharings, and the propagated shares are the same as d-CSNI case.
(f) d-OCSNI: output probes are the same as d-CSNI case, and the propagated shares are shares from packed sharings.

in Sections 6 and 7.1. We leave the proposition and proof of more generic composition rules as an important goal for further research and present in the full version of the paper first steps in this direction.

Composability of all the new notions (i.e., d-C(S)NI, d-IC(S)NI and d-OC(S)NI) can be proved by using the probe propagation framework introduced in [11, 3] (see a description in the full version).

4 Packed multiplication gadget

4.1 Construction

We consider the element-wise product (a.k.a., the entrywise product or the Hadamard product) of two secret vectors. That is, for $\mathbf{x} \stackrel{\text{def}}{=} \big(\mathbf{x}[1], \ldots, \mathbf{x}[\ell]\big)$ and $\mathbf{y} \stackrel{\text{def}}{=} \big(\mathbf{y}[1], \ldots, \mathbf{y}[\ell]\big)$, we consider computing $\mathbf{z} = \mathbf{x} \odot \mathbf{y} \stackrel{\text{def}}{=} \big(\mathbf{x}[1]\mathbf{y}[1], \ldots, \mathbf{x}[\ell]\mathbf{y}[\ell]\big)$ in the masked domain, where $\odot$ denotes the element-wise multiplication over $\mathbb{F}_q^\ell$. The inputs of the packed multiplication gadget are $\ell \times 2$ Boolean sharings:

$$\{\hat{\mathbf{x}}_i\}_{i=1}^{\ell} \stackrel{\text{def}}{=} \big\{\big(\hat{\mathbf{x}}_i[1], \ldots, \hat{\mathbf{x}}_i[n]\big)\big\}_{i=1}^{\ell} \text{ and } \quad \{\hat{\mathbf{y}}_i\}_{i=1}^{\ell} \stackrel{\text{def}}{=} \big\{\big(\hat{\mathbf{y}}_i[1], \ldots, \hat{\mathbf{y}}_i[n]\big)\big\}_{i=1}^{\ell}$$

And the outputs should also be ℓ Boolean sharings $\{\hat{\mathbf{z}}_i\}_{i=1}^{\ell}$ such that:

$$\sum_{i=1}^{n} \hat{\mathbf{z}}_k[i] = \big(\sum_{i=1}^{n} \hat{\mathbf{x}}_k[i]\big)\big(\sum_{i=1}^{n} \hat{\mathbf{y}}_k[i]\big), \text{ for any } k \in [\ell].$$

The gadget requires an $(n-1) \times \ell$ matrix $\mathbf{A}$ such that any $d < n$ columns of $[\mathbf{I}, \mathbf{A}]$ are independent. In other words, $[\mathbf{I}, \mathbf{A}]$ is the generating matrix $\big($with the size $(n-1) \times (\ell+n-1)\big)$ of a liner code with dual distance $d+1$. A typical example of $\mathbf{A}$ is an $(n-1) \times \ell$ MDS matrix, and in this case, $d = n-1$.

The packed multiplication can be divided into two sub-gadgets: *Packing* and *Multiplying*. Generally speaking, the first gadget manipulates $\{\hat{\mathbf{x}}_i\}_{i=1}^{\ell}$ and $\{\hat{\mathbf{y}}_i\}_{i=1}^{\ell}$ separately to compute the packed sharings $(\tilde{\mathbf{x}}, \hat{\mathbf{u}}) \in (\mathbb{F}_q^\ell, \mathbb{F}_q^{n-1})$ and $(\tilde{\mathbf{y}}, \hat{\mathbf{v}}) \in (\mathbb{F}_q^\ell, \mathbb{F}_q^{n-1})$, and the second gadget computes the result from the packed sharings. More details are elaborated as follows:

- Packing: This sub-gadget packs the sharings $\{\hat{\mathbf{x}}_i\}_{i=1}^{\ell}$ into a packed sharing that is a tuple $(\tilde{\mathbf{x}}, \hat{\mathbf{u}}) \in (\mathbb{F}_q^\ell, \mathbb{F}_q^{n-1})$, such that for any $k \in [\ell]$, x_k can be reconstructed from $\tilde{\mathbf{x}}[k]$ and $\hat{\mathbf{u}}$ via $x_k = \tilde{\mathbf{x}}[k] \oplus \hat{\mathbf{u}}\mathbf{A}[, k]$. The packed sharings should also meet the requirement of security, that is, any d elements of $(\tilde{\mathbf{x}}, \hat{\mathbf{u}})$ are independent of the secret variables $\mathbf{x}$.
  Similarly, $\{\hat{\mathbf{y}}_i\}_{i=1}^{\ell}$ are also packed into a packed sharings $(\tilde{\mathbf{y}}, \hat{\mathbf{v}})$ in the same vein.
- Multiplying: This sub-gadget computes the sharings of $\mathbf{x} \odot \mathbf{y}$ from the packed sharings $(\tilde{\mathbf{x}}, \hat{\mathbf{u}})$ and $(\tilde{\mathbf{y}}, \hat{\mathbf{v}})$. At a high level, for each $k \in [\ell]$, this sub-gadget perform a calculation with two-stages that first calculates outer product of the input shares $(\tilde{\mathbf{x}}[k], \hat{\mathbf{v}})^{\mathrm{T}} \times (\tilde{\mathbf{y}}[k], \hat{\mathbf{u}})$, and then compresses the results with some randomness. More importantly, the random matrix $\mathbf{R}$ and the calculation of $\mathbf{S}$ are shared (amortized) for different values of k.

We give the packed multiplication gadget in Gadget 1, which is made up of Gadget 1-P and Gadget 1-M for packing and multiplying respectively. We also present examples of Gadget 1-P and Gadget 1-M in Figures 6 and 7 respectively.

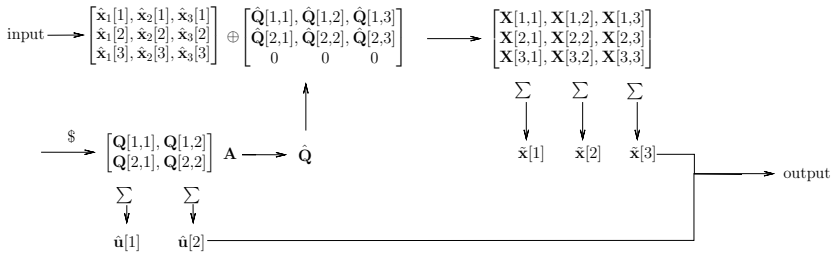


Fig. 6. Illustration of Gadget 1-P for $n = 3$ and $\ell = 3$

$\mathbf{a}_k^{\mathrm{T}} = \mathbf{A}[, k]$ for $k \in [\ell]$
$\mathbf{R}$ is a 2×2 symmatric random matrix
$\mathbf{R}_{\mathrm{diag}}$ is the diagonal matrix of $\mathbf{R}$

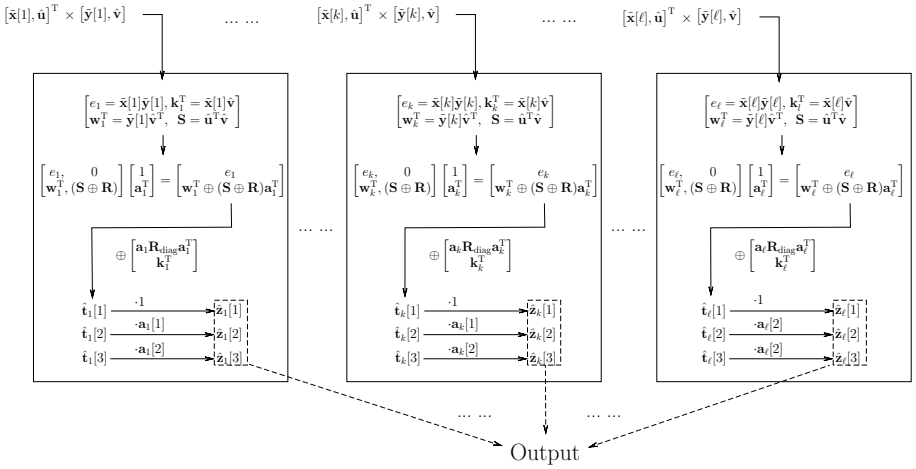


Fig. 7. Illustration of Gadget 1-M for $n = 3$

Gadget 1 Packed Multiplication

Input: Boolean sharings $\{\hat{\mathbf{x}}_i\}_{i=1}^{\ell} \in (\mathbb{F}_q^n, \ldots, \mathbb{F}_q^n)$ and $\{\hat{\mathbf{y}}_i\}_{i=1}^{\ell} \in (\mathbb{F}_q^n, \ldots, \mathbb{F}_q^n)$.
Output: Boolean sharings $\{\hat{\mathbf{z}}_i\}_{i=1}^{\ell} \in (\mathbb{F}_q^n, \ldots, \mathbb{F}_q^n)$.

1: The gadget ensures that:

$$\hat{\mathbf{z}}_k[1] \oplus \ldots \oplus \hat{\mathbf{z}}_k[n] = \big(\sum_{i=1}^{n} \hat{\mathbf{x}}_k[i]\big)\big(\sum_{i=1}^{n} \hat{\mathbf{y}}_k[i]\big), \text{ for any } k \in [\ell].$$

2: $\mathbf{A}$ is an $(n-1) \times \ell$ matrix over $\mathbb{F}_q$ such that any d columns of $[\mathbf{I}, \mathbf{A}]$ are independent.

Gadget 1-P: Packing

Input: Boolean sharings $\{\hat{\mathbf{x}}_i\}_{i=1}^{\ell}$
Output: Packed sharings $(\tilde{\mathbf{x}}, \hat{\mathbf{u}}) \in (\mathbb{F}_q^{\ell}, \mathbb{F}_q^{n-1})$
The gadget ensures that: $\mathbf{x}_k = \tilde{\mathbf{x}}[k] \oplus \hat{\mathbf{u}}\mathbf{A}[,k]$, for any $k \in [\ell]$.

1: Randomly generate a matrix $\mathbf{Q} \in \mathbb{F}_q^{(n-1)\times(n-1)}$
▷ Amortization: The size of $\mathbf{Q}$ is independent of ℓ
2: $\hat{\mathbf{Q}} := \mathbf{Q}\mathbf{A}$
3: $\mathbf{X} := [\hat{\mathbf{x}}_1^{\mathrm{T}}, \ldots, \hat{\mathbf{x}}_{\ell}^{\mathrm{T}}] \oplus [\hat{\mathbf{Q}}; \mathbf{0}^{\ell}]$ ▷ $\mathbf{0}^{\ell}$ denotes an ℓ-length zero vector.
4: $\hat{\mathbf{u}} := \sum_{i=1}^{n-1} \mathbf{Q}[i,]$ and $\tilde{\mathbf{x}} := \sum_{i=1}^{n} \mathbf{X}[i,]$

For the packing from $\{\hat{\mathbf{y}}_i\}_{i=1}^{\ell}$ to $(\hat{\mathbf{v}}, \tilde{\mathbf{y}})$: Repeat Gadget 1-P with input $\{\hat{\mathbf{y}}_i\}_{i=1}^{\ell}$. It ensures that: $\mathbf{y}_k = \tilde{\mathbf{y}}[k] \oplus \hat{\mathbf{v}}\mathbf{A}[,k]$, for any $k \in [\ell]$.

Gadget 1-M: Multiplying

Input: Packed sharings $(\tilde{\mathbf{x}}, \hat{\mathbf{u}})$ and $(\tilde{\mathbf{y}}, \hat{\mathbf{v}})$.
Output: Boolean sharings $\{\hat{\mathbf{z}}_k\}_{k=1}^{\ell}$.
The gadget ensures that $\sum_{i=1}^{n} \hat{\mathbf{z}}_k[i] = \big(\tilde{\mathbf{x}}[k] \oplus \hat{\mathbf{u}}\mathbf{A}[,k]\big)\big(\tilde{\mathbf{x}}[k] \oplus \hat{\mathbf{v}}\mathbf{A}[,k]\big)$, for any $k \in [\ell]$.

1: Randomly generate a symmetric matrix $\mathbf{R} \in \mathbb{F}_q^{(n-1)\times(n-1)}$
2: Let $\mathbf{R}_{\mathrm{diag}}$ be the diagonal matrix such that $\mathbf{R}_{\mathrm{diag}}[i,i] = \mathbf{R}[i,i]$ for $i \in [n-1]$
3: **for** $k = 1$; $k \le \ell$; $k{+}{+}$ **do**
4: Let $\mathbf{a}_k^{\mathrm{T}} = \mathbf{A}[,k]$
5: $e_k := \tilde{\mathbf{x}}[k]\tilde{\mathbf{y}}[k], \mathbf{k}_k := \tilde{\mathbf{x}}[k]\hat{\mathbf{v}}, \mathbf{w}_k^{\mathrm{T}} := \hat{\mathbf{u}}^{\mathrm{T}}\tilde{\mathbf{y}}[k], \mathbf{S} := \hat{\mathbf{u}}^{\mathrm{T}}\hat{\mathbf{v}}$
▷ Compute the outer product: $\begin{bmatrix} e_k, & \mathbf{k}_k \\ \mathbf{w}_k^{\mathrm{T}}, & \mathbf{S} \end{bmatrix} = \big[\tilde{\mathbf{x}}[k], \hat{\mathbf{u}}\big]^{\mathrm{T}} \times \big[\tilde{\mathbf{y}}[k], \hat{\mathbf{v}}\big]$
▷ Amortization: $\mathbf{S}$ only need to be computed once for different values of k
6: $\hat{\mathbf{t}}_k^{\mathrm{T}} := \begin{bmatrix} e_k, & 0 \\ \mathbf{w}_k^{\mathrm{T}}, & (\mathbf{S} \oplus \mathbf{R}) \end{bmatrix} \begin{bmatrix} 1 \\ \mathbf{a}_k^{\mathrm{T}} \end{bmatrix} \oplus \begin{bmatrix} \mathbf{a}_k\mathbf{R}_{\mathrm{diag}}\mathbf{a}_k^{\mathrm{T}} \\ \mathbf{k}_k^{\mathrm{T}} \end{bmatrix}$
▷ Amortization: $\mathbf{R}$ and $\mathbf{R}_{\mathrm{diag}}$ are re-used for different values of k
7: $\hat{\mathbf{z}}_k := \hat{\mathbf{t}}_k \odot [1, \mathbf{a}_k]$
8: **end for**

4.2 Correctness of Gadget 1

In the following, we claim the correctness of Gadget 1, and the proof is given in the full version.

Theorem 1 *The correctness of Gadget 1-P and Gadget 1-M are ensured, i.e., for any* $k \in [\ell]$, $\mathbf{x}_k = \tilde{\mathbf{x}}[k] \oplus \hat{\mathbf{u}}\mathbf{A}[,k]$, $\mathbf{y}_k = \tilde{\mathbf{y}}[k] \oplus \hat{\mathbf{v}}\mathbf{A}[,k]$, *and* $\sum_{i=1}^{n} \hat{\mathbf{z}}_k[i] = \big(\tilde{\mathbf{x}}[k] \oplus \hat{\mathbf{u}}\mathbf{A}[,k]\big)\big(\tilde{\mathbf{x}}[k] \oplus \hat{\mathbf{v}}\mathbf{A}[,k]\big)$.

4.3 Security of Gadget 1

We first describe some intuitions behind the construction with respect to the security. Then, we give the security claim of Gadget 1 in Theorem 2, where the proof will be given in the full version.

Gadget 1-P first generates a uniformly distributed matrix $\mathbf{Q}$, which is then multiplied by $\mathbf{A}$. And, the result is used to mask the input sharings, resulting in $\mathbf{X}$. As any d columns of $[\mathbf{I}, \mathbf{A}]$ are independent, any d columns of $[\mathbf{Q}, \mathbf{QA}]$ are uniformly distributed. We can see that all probes (at most d) to Gadget 1-P should relate to no more than d columns of $[\mathbf{Q}, \mathbf{QA}]$. To ease the analysis, we can consider a simple case that the entries of $\mathbf{Q}$ are unknown (and there is no probe to the calculation of $\mathbf{QA}$), the process of summing the rows of $[\mathbf{Q}, \mathbf{X}]$ should be randomized by uniform random elements, preventing the leaks of inputs. Then, regarding the case that $\mathbf{Q}$ leaks, one can refer to the rationale of the t-chunk definition in Section 3.2.

The intuition behind the construction of Gadget 1-M is similar, but analysis will be more scrupulous, since the random matrix $\mathbf{R}$ is symmetric.

Theorem 2 *Gadget 1-P is d-ICSNI, Gadget 1-M is d-OCNI, and Gadget 1 is d-CSNI.*

5 Linear gadgets

In this section, we discuss how to implement a linear transformation $\mathsf{L} : \mathbb{F}_q^{\ell} \to \mathbb{F}_q^{\ell'}$ with sharings. First, sub-section 5.1 shows that the trivial implementation of a linear function is d-NI. Though, such a trivial implementation suffers from limitations in the composition with d-CSNI gadgets (e.g., the packed multiplication), which is shown in sub-section 5.2. This motivates the construction of a more secure d-CNI linear gadget in sub-section 5.3.

5.1 Trivial implementation

Gadget 2 shows the trivial implementation of a linear operation with Boolean sharings $\{\hat{\mathbf{x}}_i\}_{i=1}^{\ell}$. The gadget manipulates shares with different indices separately. Each internal probe relates to at most one index of $\{\hat{\mathbf{x}}_i\}_{i=1}^{\ell}$, and any t_{out} shares of $\{\hat{\mathbf{z}}_i\}_{i=1}^{\ell}$ relates to at most t_{out} indices of $\{\hat{\mathbf{x}}_i\}_{i=1}^{\ell}$, and in total any t_{int} internal probes and t_{out} shares of $\{\hat{\mathbf{z}}_i\}_{i=1}^{\ell}$ can be simulated with at most $(t_{int} + t_{out})$ shares of $\{\hat{\mathbf{x}}_i\}_{i=1}^{\ell}$. Thus, Gadget 2 is d-NI for any $d \leq n$.

However, Gadget 2 is not d-CNI. For example, if $\mathsf{L}(\{\hat{\mathbf{x}}_k[i]\}_{k=1}^{\ell}) = \sum_{k=1}^{\ell} \hat{\mathbf{x}}_k[i]$ for $i \in [n]$, then for $t \leq d$, any t shares of $\hat{\mathbf{z}}$ depend on t shares of each of $\hat{\mathbf{x}}_k[i]$

Gadget 2 Trivial linear operation

Input: Boolean sharings$\{\hat{\mathbf{x}}_i\}_{i=1}^{\ell} \in (\mathbb{F}_q^n, \ldots, \mathbb{F}_q^n)$
Output: Boolean sharings $\{\hat{\mathbf{z}}\}_{i=1}^{\ell'} \in (\mathbb{F}_q^n, \ldots, \mathbb{F}_q^n)$
1: **for** $i = 1$; $i \leq n$; $i{+}{+}$ **do**
2: $\quad \hat{\mathbf{z}}_1[i], \ldots, \hat{\mathbf{z}}_{\ell'}[i] = \mathsf{L}(\hat{\mathbf{x}}_1[i], \ldots, \hat{\mathbf{x}}_\ell[i])$
3: **end for**

for $i \in [n]$, rather than a t-chunk of input sharings. An exception is when shares of input sharings are operated separately, which is shown in Lemma 4, and the proof is given in the full version.

Lemma 4 *Any gadget that manipulates the shares of input sharings separately (i.e., there is no single variable related to more than one input shares), is d-CNI for any $d \geq 0$.*

5.2 Why a *d*-CNI linear gadget is necessary?

While trivially implemented linear gadgets are quite efficient, its composition with the d-CSNI packed multiplication gadget (described in Section 4) is not. Below we elaborate with an example.

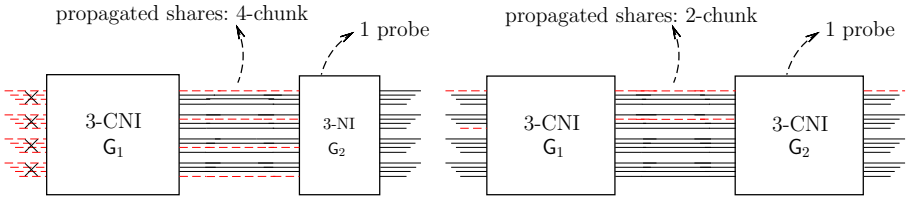
Figure 8-(a) shows an improper composition: G_1 and G_2 are 3-CSNI and 3-NI respectively, and one probe of G_2 can be simulated with one share of each G_2's input sharing, which however cannot be simulated with the input of G_2. To fix this issue, one can rely on the strategy of adding refreshings between the two gadgets in the same way as Figure 3-(c) in Section 2.3. Note that a d-SNI refresh gadget for one sharing of size n asymptotically requires up to $O(n \log n)$ random elements [4, 1], and with all the sharings, it leads to an inefficient composition. Figure 3-(b) shows a more efficient solution, where G_2 is changed with a 3-CNI gadget to make the composition work. The latter solution (of Figure 3-(b)) raises the following question:

> Can a d-CNI linear gadget be more efficient than the strategy of combining a trivial linear gadget with d-SNI refreshings?

5.3 New construction of linear operation

We answer the question affirmatively. In Gadget 3, we give a new construction of linear operation for Boolean sharings. It first refreshes each input sharing by using the so-called locality refreshing [25, 15], which requires $n - 1$ random elements. Then, it performs the linear operation on the refreshed sharings. In total, Gadget 3 uses $\ell(n{-}1)$ random elements for ℓ input sharings. In Theorem 3, we claim the security of Gadget 3, and the proof is given in the full version.

Theorem 3 *Gadget 3 is d-CNI for any d such that $d \leq n$.*



(a) improper compostion of d-CNI and d-NI gadgets

(b) proper compostion of two d-CNI gadgets

Fig. 8. An example to show the necessity of d-CNI linear gadget

Gadget 3 d-CNI Linear operation

Input: Boolean sharings $\{\hat{\mathbf{x}}_i\}_{i=1}^{\ell} \in (\mathbb{F}_q^n, \ldots, \mathbb{F}_q^n)$
Output: Boolean sharings $\{\hat{\mathbf{z}}_i\}_{i=1}^{\ell'} \in (\mathbb{F}_q^n, \ldots, \mathbb{F}_q^n)$
1: **for** $k = 1$; $k \leq \ell$; $k{+}{+}$ **do**
2: Generate a uniformly distributed vector $\mathbf{r}_k \in \mathbb{F}_q^{n-1}$
3: $\hat{\mathbf{y}}_k[1\!:\!n-1] = \hat{\mathbf{x}}_k[1\!:\!n-1] \oplus \mathbf{r}_k$
4: $\hat{\mathbf{y}}_k[n] = \hat{\mathbf{x}}_k[n] \oplus \sum_{i=1}^{n-1} \mathbf{r}_k[i]$
5: **end for**
6: Call Gadget 2 with input sharings $\{\hat{\mathbf{y}}_i\}_{i=1}^{\ell}$ and output sharings $\{\hat{\mathbf{z}}_i\}_{i=1}^{\ell'}$

5.4 Linear gadgets for packed sharings

The linear gadget investigated above only considers (Boolean) sharings, which is already sufficient to protect the cryptographic algorithms. For the packed sharings, the linear gadget are more complicated and may come at high overhead. The main reason is that, the trivial implementation of linear transformation gadget is based on the premise that Boolean sharings encode each secret independently, which however is not standing for the packed sharings. Besides, the code-based masking also face this issue, and a similar reasoning can be found in [38, Sction 5.2].

An exception is that the addition over packed sharings can be trivially implemented by manipulating shares with different indices separately. That is, for input packed sharings $(\tilde{\mathbf{x}}, \hat{\mathbf{u}})$ and $(\tilde{\mathbf{y}}, \hat{\mathbf{v}})$, the trivial addition is $(\tilde{\mathbf{x}} \oplus \tilde{\mathbf{y}}, \hat{\mathbf{u}} \oplus \hat{\mathbf{v}})$. In Lemma 5, we give the security of this trivial addition, which can be regarded as a variant of d-NI for the packed sharings (note that the d-NI is defined only for gadgets with input and output sharings). The proof will be given in the full version.

Lemma 5 *For a trivial addition gadget with two input packed sharings, any t_{out} shares of output packed sharings and t_{int} internal probes can be simulated with $t_{int} + t_{out}$ shares of each of input packed sharings.*

6 Application to AES SubBytes

6.1 Implementation approach using the tower field method

AES-128, the internal states, including the round keys, are viewed as a set of 16 variables (say, $\{x_1, \ldots, x_{16}\}$) in $\mathbb{F}_{2^8}$. In its SubBytes step, an S-box is computed over each of the 16 states. The S-box is a nonlinear function $\mathbb{F}_{2^8} \to \mathbb{F}_{2^8}$ that consists of the inverse in $\mathbb{F}_{2^8}$ and an affine transformation. In the field inversion can be decomposed into several multiplications in $\mathbb{F}_{2^4}$ (that can be fully tabulated) and linear operations using the tower field method [29]:

1. $(a_h, a_l) := \delta(x) \in \big(\mathbb{F}_{2^4}, \mathbb{F}_{2^4}\big)$
2. $a := \lambda a_h^2 \oplus a_l(a_h \oplus a_l) \in \mathbb{F}_{2^4}$
3. $a' := (a^2 a)^4 a^2 \in \mathbb{F}_{2^4}$
4. $a'_h := a' a_h \in \mathbb{F}_{2^4}$
5. $a'_l := a'(a_h + a_l) \in \mathbb{F}_{2^4}$
6. $\mathsf{S}(x) := \mathrm{Aff}\Big(\delta^{-1}\big((a'_h, a'_l)\big)\Big) \in \mathbb{F}_{2^8}$

In detail, the input $x \in \mathbb{F}_{2^8}$ is mapped to $a_h, a_l \in \mathbb{F}_{2^4}$ using a linear isomorphism mapping $\delta : \mathbb{F}_{2^8} \to (\mathbb{F}_{2^4}, \mathbb{F}_{2^4})$, and λ is a constant in $\mathbb{F}_{2^4}$. After computations over $\mathbb{F}_{2^4}$ in steps 2 to 5, the inverse isomorphism mapping $\delta^{-1} : (\mathbb{F}_{2^4}, \mathbb{F}_{2^4}) \to \mathbb{F}_{2^8}$ maps (a'_h, a'_l) back to an element in field $\mathbb{F}_{2^8}$, and finally, an affine transformation $\mathrm{Aff} : \mathbb{F}_{2^8} \to \mathbb{F}_{2^8}$ yields the S-box output.

We use MDS matrices from the Reed-Solomon code [34], and thus $n = d + 1$. By the MDS conjecture [36], $d \times \ell$ MDS matrix over $\mathbb{F}_{2^4}$ shall satisfy $\ell + d \leq |\mathbb{F}_{2^4}| = 16$. Thus, we set $\ell = 8$ and implement 8 S-boxes together by using the packed multiplication (16 S-boxes can be achieved by invoking this implementation twice). The input and output of masked S-boxes are 8 sharings. The implementation is optimized by separating the packing and multiplying gadgets to reduce the number of calls to packing, as well as to re-use the packed sharings to the largest extent. The process is shown in Figure 9, in which P and M denote the packing and multiplying of Gadget 1-P and Gadget 1-M with $\ell = 8$ respectively. The $()^2$, $()^4$, δ, λa^2 and $\oplus$ are trivial implementations of the corresponding linear operations, and the last gadget that is a combination of inverse isomorphism and affine is implemented by Gadget 3.

In the security analysis, to be strictly consistent with the definition of circuits, where all variables are in the same finite field, we map each variables (say a) in $\mathbb{F}_{2^4}$ to a variable (say b) in $\mathbb{F}_{2^8}$, such that the most significant 4 bits of b are identical to the 4 bits of a, and the least significant 4 bits of b are zeros. Then, each function over $\mathbb{F}_{2^4}$ is isomorphically mapped to a gate over $\mathbb{F}_{2^8}$ by which the function is performed only over the most significant 4 bits of the variables. The function $\delta : \mathbb{F}_{2^8} \to (\mathbb{F}_{2^4}, \mathbb{F}_{2^4})$ $\big($resp., $\delta^{-1} : (\mathbb{F}_{2^4}, \mathbb{F}_{2^4}\big) \to \mathbb{F}_{2^8})$ is isomorphically mapped to a gate $\mathbb{F}_{2^8} \to (\mathbb{F}_{2^8}, \mathbb{F}_{2^8})$ $\big($resp., $\mathbb{F}_{2^8} \to (\mathbb{F}_{2^8}, \mathbb{F}_{2^8})\big)$ by which each output (resp., input) is mapped to a variable in $\mathbb{F}_{2^8}$ by the same vein as before. Note that these mappings are only for the security analysis and do not impact the efficiency of the implementation.

Proposition 1 (The SubBytes implementation is d-CSNI) *The composed gadget in Figure 9 is d-CSNI.*

The full proof is given in the full version.

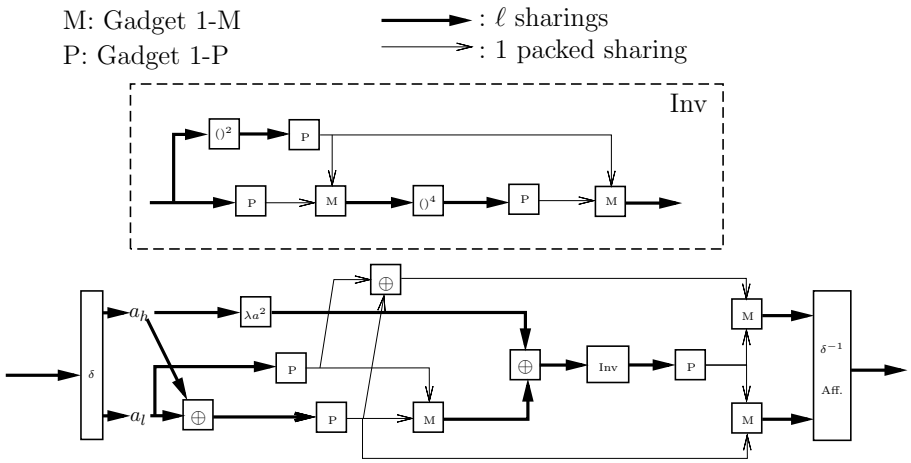


Fig. 9. Masked AES S-box with packed multiplication.

Though we adopt the tower field method [29] and separate the packing and multiplying gadgets for the sake of reducing the cost to the utmost. We believe a simpler implementation using the multiplication chain [35] in a larger field $\mathbb{F}_{2^8}$ will be interesting as well. In this respect, we describe such a masked AES implementation in the full version.

6.2 Implementation results

It can be seen that, the implementation of 8 S-boxes contains 6 instances of Gadget 1-P and 5 instances of Gadget 1-M. The random requirements of Gadget 1-P and Gadget 1-M are d^2 and $d(d+1)/2$ 4-bit variables respectively. The δ^{-1} and affine operation are implemented together by Gadget 3, which requires $8d\ell$ bytes of randomness. At last, the total random bits for 16 AES S-boxes is

$$\Big(\big(d^2 \times 6 + (d(d+1)/2) \times 5\big) \times 4 + d\ell \times 8\Big) \times 2 = 68d^2 + 20d + 16d\ell.$$

For $\ell = 8$, the above result is $68d^2 + 148d$.

The S-boxes are implemented with security orders $d = 4, 8$ based on the ARM Cortex M architecture. The multiplication by matrix $\mathbf{A}$ at line 2 of Gadget 1-P and line 6 of Gadget 1-M are tabulated, which in total requires $16d\ell$ bytes of memory. For the consistency with the state-of-the-art results, the randomness in our implementations can be obtained from a constrained TRNG that outputs 32-bit of fresh randomness every 80 clock cycles, which is also used in [9] and recommended in [27]. For the comparison with the state-of-the-art implementations, we consider the implementations of bitslice AES S-boxes reported in [19, 9] as the benchmarks.

The performance results are summarized in Table 2. Compared with the work of [9], our implementation saves 55% and 68% cycles for the generation of

randomness for $d = 4$ and 8 respectively. The code sizes of our implementations are larger, which is due to the loop unrolling of our implementation. Indeed, our implementations are slightly slower than the bitsliced methods in computation, which is because that bitsliced methods perfectly fit the bitwise AND and XOR instructions. By contrast, our implementations are based on the multiplication in GF(2^4), which is not directly supported in microprocessors and can only rely on pre-computed tables. Nevertheless, we emphasize that this computational loss could be mitigated or eliminated via the following two approaches:

1. One can optimize the matrix $\mathbf{A}$ to make the corresponding multiplication more efficient. Sometimes an MDS matrix is not needed: even though $d < n-1$, the ratio of cost to security order may be better (than using the MDS matrix).
2. One can implement the masked AES on hardware, where the field multiplication and linear transformation can be optimized in bit-level.

We refer to them as future works. Finally, despite the computational loss, our implementation still achieves a gain of up to 33% in total speed when $d = 8$.

Regarding computational cost, the issue of field multiplication in software indicates that bitsliced implementations may be more efficient. However, the bitsliced consumes more randomness. With same value of security order d and the number of parallel multiplications (say, ℓ), larger field (say, $\mathbb{F}_{2^4}$ or $\mathbb{F}_{2^8}$) may give a smaller number of shares n for a packed sharing. Generally, if $\ell + d \leq |\mathbb{F}_q|$, we can choose $\mathbf{A}$ in Gadget 1 an MDS matrix, and then we have $n = \ell + d$. But for bitsliced case, $|\mathbb{F}_q| = 2$, and thus $n > \ell + d$. Therefore, the situation of combining bitsliced implementation with the packed multiplication is more complicated: operation can be more efficient (with the bitwise AND instruction) at the cost of more randomness bits. We refer to this investigation as a future work.

Last but not least, we make the source codes of our AES-Sboxes implementation available on https://github.com/wjwangcrypto/Packed_mul.

Table 2. Summary of performances for 16 AES S-boxes

	Cycles for Computation	Cycles for Generating Randomness	Total Cycles	Code size	RAM size
[19, R.-P. method], $d = 4$	19 232	34 944	54 176	4KB	unreported
[19, Bitsliced method], $d = 4$	11 502	17 472	28 974	3.1KB	unreported
[9, Bitsliced method], $d = 4$	9 222	9 282	18 504	unreported	unreported
Our work, $d = 4$	15 998	4 200	**20 198**	9.8KB	10.9KB
[19, R.-P. method], $d = 8$	70 840	163, 072	233 912	4KB	unreported
[19, Bitsliced method], $d = 8$	34 798	81 536	116 334	3.1KB	unreported
[9, Bitsliced method], $d = 8$	27 028	43 316	70 344	unreported	unreported
Our work, $d = 8$	33 142	13 840	**46 982**	17KB	11.8KB

7 Application to GHASH, AES-GCM, and more

7.1 A brief description of GHASH and AES-GCM

Authenticated encryption aims at ensuring both confidentiality and integrity simultaneously [10], and has became the de facto standard for secure data transferring. The authenticated encryption algorithm AES-GCM was proposed by McGrew and Viega in [30] and standardized by NIST since 2007. It combines an encryption based on the widely used AES algorithm in counter mode and an authenticator based on the GHASH function involving multiplications in $\mathrm{F}_{2^{128}}$. The authenticator mixes ciphertexts, potential associated data and a secret parameter derived from the encryption key to produce a tag.

It is compulsory to seek for side-channel secure implementations for such a standard. A crucial step is to secure the GHASH function, which is essentially a *polynomial-evaluation hash*. Its takes $\iota + 1$ variables $s_0, \ldots, s_\iota$ in $\mathrm{F}_{2^{128}}$ as well as an authentication key $h \in \mathrm{F}_{2^{128}}$ as inputs, and evaluates Equation 2 below.

$$\mathrm{tag} = h^\iota s_0 \oplus h^{\iota-1} s_1 \oplus \ldots \oplus h s_\iota \ . \tag{1}$$

A sequential calculation of the polynomial-evaluation hash can be built by the Horner's rule [24]:

$$x_i = \begin{cases} 0 & \text{for } i = 0 \\ (x_{i-1} \oplus s_i)h & \text{for } i = 1, \ldots, \iota \end{cases} \ , \tag{2}$$

where the output $\mathrm{tag} = x_\iota$.

For the underlying block cipher AES, the implementation approach has been discussed in Section 6. Here we concentrate on the other main indigent GHASH. Note that various SCAs against GHASH have been reported in e.g., [8, 7], which enable recovering the key h and creating forgeries. It is thus unsurprising that masking GHASH has received quite a lot attention, see e.g., [32, 37]. However, existing masked implementations of GHASH only considered protecting against known SCAs, leaving out *provable security*. Here we will fill in the gap. In detail, we study the case that h and $s_0, \ldots, s_\iota$ are encoded into sharings: $\hat{\mathbf{h}}$ and $\hat{\mathbf{s}}_0, \ldots, \hat{\mathbf{s}}_\iota$, and the masked GHASH outputs the sharing of the tag. The crux is to masking the polynomial-evaluation hash (Equation 2), on which we will elaborate in the next sub-section.

7.2 Provably secure masked implementation of polynomial-evaluation hash

To mask the polynomial-evaluation hash, the most straightforward approach is to apply ISW multiplication (more concretely, the generalized version for finite field in [35]) in the sequential calculation of Equation 2. This approach consumes $\iota+1$ ISW multiplications, each of which consists of $(d+1)^2$ bilinear multiplications and requires $64(d+1)d$ random bits. Based on the above, the cost of this approach is estimated and summarized in Table 3.

Note that the computation of polynomial-evaluation hash can be parallelized. In detail, assuming $\ell \mid \iota$, the parallelized version computes $\{x_1^{(i)}, \ldots, x_\ell^{(i)}\}$ from $i = 0$ to $i = \frac{\iota}{\ell}$ as follows:

$$x_k^{(i)} = \begin{cases} s_k h^k & \text{for } i = 0 \\ (x_k^{(i-1)} \oplus s_{k+i\ell})h^k & \text{for } i = 1, \ldots, \frac{\iota}{\ell} \end{cases}, \text{ for } k \in [\ell] \tag{3}$$

Finally, the summation $\sum_{k=1}^{\ell} x_k^{(\frac{\iota}{\ell})}$ is taken as the tag. The computation of $\{x_1^{(i)}, \ldots, x_\ell^{(i)}\}$ for $i \in [\frac{\iota}{\ell}]$ can be parallelized and thus fits our packed multiplication of Gadget 1. In Figure 10, we present our new approach based on packed multiplication.

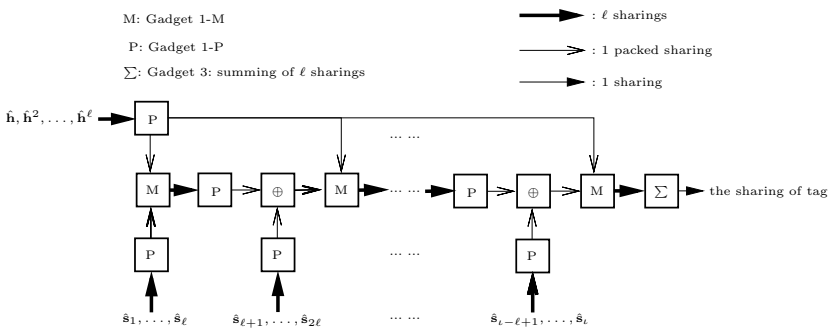


Fig. 10. Masked polynomial-evaluation hash with packed multiplication.

Based on the probing propagation framework, it is easy to see that the composed gadget in Figure 10 is d-CSNI. To estimate the cost, we use the MDS matrix from the Reed-Solomon code for the matrix $\mathbf{A}$ of our packed multiplication gadgets, and thus $n = d + 1$. By the MDS conjecture [36], ℓ and d can be arbitrarily large as long as $\ell + d \leq |\mathbb{F}_{2^{128}}| = 2^{128}$. The estimated cost of this approach is also given in Table 3. It can be seen that, asymptotically, the new scheme with packed multiplication achieves a gain of cost up to ℓ times from the straightforward approach.

Table 3. Estimated costs of the masked polynomial-evaluation hash over $\mathbb{F}_{2^{128}}$

	Sequential Implementation with ISW multiplication	Figure 10 with Packed multiplication
Randomness complexity (in bits)	$64(\iota+1)d(d+1)$	$\frac{64(\iota+1)d(d+1)+128(\iota+1)d^2}{\ell}$
Computational complexity *	$(\iota+1)(d+1)^2$	$\frac{(\iota+1)(d+1)^2}{\ell}$

* Metric: the number of bilinear multiplications.

7.3 More applications of the masked polynomial-evaluation hash

Besides the GCM, polynomial-evaluation hashes have wide applications, see [39, 31]. We thus believe our approach have a great impact. To demonstrate, we take disk encryption as another example. For this purpose, Halevi proposed a mode named TET (short for linear-Transformation; ECB; linear-Transformation) [23]. The mode can be seen as the ECB encryption sandwiched between two layers of "blockwise-universal hash". An instance of such hashes proposed in [23] was named Blockwise Polynomial-Evaluation (BPE). With inputs $x_1, \ldots, x_\tau \in \mathbb{F}_{2^p}^\iota$ and key $(\beta, \tau) \in (\mathbb{F}_{2^p}, \mathbb{F}_{2^p})$, BPE firstly computes

$$s = x_1\tau \oplus x_2\tau^2 \oplus \ldots \oplus x_\iota\tau^\iota. \tag{4}$$

Then, the result is obtained by

$$y_i = x_i \oplus s \oplus \alpha^{i-1}\beta, \text{ for } i \in [\iota],$$

where $\alpha \in \mathbb{F}_{2^p}$ is a constant. It is clear that BPE is essentially a polynomial-evaluation hash following Equation 1, and thus it can also be parallelized and implemented in the same vein as that of Figure 10.

Acknowledgments. We would like to thank the anonymous reviewers of Asiacrypt 2020. Weijia Wang was partly supported by the Program of Qilu Young Scholars (Grant No. 61580082063088) of Shandong University. Chun Guo was partly supported by the Program of Qilu Young Scholars (Grant No. 6158008996-3177) of Shandong University & National Key Research and Development Project under Grant No.2018YFA0704702 & Major Scientific and Technological Innovation Project of Shandong Province, China under Grant No.2019JZZY010133 & Major Scientific and Technological Innovation Project of Shandong Province, China under Grant No.2017CXGC0704. Gaëtan Cassiers and François-Xavier Standaert are resp. Research Fellow and Senior Associate Researcher of the Belgian Fund for Scientific Research (FNRS-F.R.S.). This work has been funded in parts by the European Union through the ERC project SWORD (724725) and the European Union and Walloon Region FEDER USERMedia project 501907379156. Yu Yu was supported by the National Key Research and Development Program of China (Grant No. 2018YFA0704701), National Natural Science Foundation of China (Grant No. 61872236 and 61971192), and the National Cryptography Development Fund (Grant No. MMJJ20170209).

References

1. Barthe, G., Belaïd, S., Dupressoir, F., Fouque, P., Grégoire, B., Standaert, F., Strub, P.: Improved parallel mask refreshing algorithms: generic solutions with parametrized non-interference and automated optimizations. J. Cryptogr. Eng. **10**(1), 17–26 (2020), https://doi.org/10.1007/s13389-018-00202-2

2. Barthe, G., Belaïd, S., Dupressoir, F., Fouque, P., Grégoire, B., Strub, P., Zucchini, R.: Strong non-interference and type-directed higher-order masking. In: Weippl, E.R., Katzenbeisser, S., Kruegel, C., Myers, A.C., Halevi, S. (eds.) CCS 2016. pp. 116–129. ACM (2016), https://doi.org/10.1145/2976749.2978427
3. Barthe, G., Dupressoir, F., Faust, S., Grégoire, B., Standaert, F., Strub, P.: Parallel implementations of masking schemes and the bounded moment leakage model. In: Coron, J., Nielsen, J.B. (eds.) EUROCRYPT 2017. LNCS, vol. 10210, pp. 535–566 (2017), https://doi.org/10.1007/978-3-319-56620-7_19
4. Battistello, A., Coron, J., Prouff, E., Zeitoun, R.: Horizontal side-channel attacks and countermeasures on the ISW masking scheme. In: Gierlichs, B., Poschmann, A.Y. (eds.) CHES 2016. LNCS, vol. 9813, pp. 23–39. Springer (2016), https://doi.org/10.1007/978-3-662-53140-2_2
5. Belaïd, S., Benhamouda, F., Passelègue, A., Prouff, E., Thillard, A., Vergnaud, D.: Randomness complexity of private circuits for multiplication. In: Fischlin, M., Coron, J. (eds.) EUROCRYPT 2016. LNCS, vol. 9666, pp. 616–648. Springer (2016), https://doi.org/10.1007/978-3-662-49896-5_22
6. Belaïd, S., Benhamouda, F., Passelègue, A., Prouff, E., Thillard, A., Vergnaud, D.: Private multiplication over finite fields. In: Katz, J., Shacham, H. (eds.) CRYPTO 2017. LNCS, vol. 10403, pp. 397–426. Springer (2017), https://doi.org/10.1007/978-3-319-63697-9_14
7. Belaïd, S., Coron, J., Fouque, P., Gérard, B., Kammerer, J., Prouff, E.: Improved side-channel analysis of finite-field multiplication. In: Güneysu, T., Handschuh, H. (eds.) CHES 2015. LNCS, vol. 9293, pp. 395–415. Springer (2015), https://doi.org/10.1007/978-3-662-48324-4_20
8. Belaïd, S., Fouque, P., Gérard, B.: Side-channel analysis of multiplications in GF(2128) - application to AES-GCM. In: Sarkar, P., Iwata, T. (eds.) ASIACRYPT 2014. LNCS, vol. 8874, pp. 306–325. Springer (2014), https://doi.org/10.1007/978-3-662-45608-8_17
9. Belaïd, S., Goudarzi, D., Rivain, M.: Tight private circuits: Achieving probing security with the least refreshing. In: Peyrin, T., Galbraith, S.D. (eds.) ASIACRYPT 2018. LNCS, vol. 11273, pp. 343–372. Springer (2018), https://doi.org/10.1007/978-3-030-03329-3_12
10. Bellare, M., Namprempre, C.: Authenticated encryption: Relations among notions and analysis of the generic composition paradigm. J. Cryptology **21**(4), 469–491 (2008), https://doi.org/10.1007/s00145-008-9026-x
11. Cassiers, G., Standaert, F.: Improved bitslice masking: from optimized non-interference to probe isolation. IACR Cryptology ePrint Archive **2018**, 438 (2018)
12. Cassiers, G., Standaert, F.: 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). https://doi.org/10.13154/tches.v2019.i2.162-198
13. Cassiers, G., Standaert, F.: Trivially and efficiently composing masked gadgets with probe isolating non-interference. IEEE Trans. Inf. Forensics Secur. **15**, 2542–2555 (2020), https://doi.org/10.1109/TIFS.2020.2971153
14. Chari, S., Jutla, C.S., Rao, J.R., Rohatgi, P.: Towards sound approaches to counteract power-analysis attacks. In: Wiener, M.J. (ed.) CRYPTO '99. LNCS, vol. 1666, pp. 398–412. Springer (1999), https://doi.org/10.1007/3-540-48405-1_26
15. Coron, J., Greuet, A., Zeitoun, R.: Side-channel masking with pseudo-random generator. In: Canteaut, A., Ishai, Y. (eds.) EUROCRYPT 2020. LNCS, vol. 12107, pp. 342–375. Springer (2020), https://doi.org/10.1007/978-3-030-45727-3_12

16. Damgård, I., Ishai, Y., Krøigaard, M.: Perfectly secure multiparty computation and the computational overhead of cryptography. In: Gilbert, H. (ed.) EUROCRYPT 2010. LNCS, vol. 6110, pp. 445–465. Springer (2010), https://doi.org/10.1007/978-3-642-13190-5_23
17. Duc, A., Dziembowski, S., Faust, S.: Unifying leakage models: From probing attacks to noisy leakage. In: Nguyen, P.Q., Oswald, E. (eds.) EUROCRYPT 2014. LNCS, vol. 8441, pp. 423–440. Springer (2014), https://doi.org/10.1007/978-3-642-55220-5_24
18. Faust, S., Paglialonga, C., Schneider, T.: Amortizing randomness complexity in private circuits. In: Takagi, T., Peyrin, T. (eds.) ASIACRYPT 2017. LNCS, vol. 10624, pp. 781–810. Springer (2017), https://doi.org/10.1007/978-3-319-70694-8_27
19. Goudarzi, D., Rivain, M.: How fast can higher-order masking be in software? In: Coron, J., Nielsen, J.B. (eds.) EUROCRYPT 2017. LNCS, vol. 10210, pp. 567–597 (2017), https://doi.org/10.1007/978-3-319-56620-7_20
20. Groß, H., Mangard, S., Korak, T.: Domain-oriented masking: Compact masked hardware implementations with arbitrary protection order. In: Bilgin, B., Nikova, S., Rijmen, V. (eds.) ACM 2016. p. 3. ACM (2016), https://doi.org/10.1145/2996366.2996426
21. Grosso, V., Standaert, F.: Masking proofs are tight and how to exploit it in security evaluations. In: Nielsen, J.B., Rijmen, V. (eds.) Advances in Cryptology - EUROCRYPT 2018 - 37th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Tel Aviv, Israel, April 29 - May 3, 2018 Proceedings, Part II. Lecture Notes in Computer Science, vol. 10821, pp. 385–412. Springer (2018), https://doi.org/10.1007/978-3-319-78375-8_13
22. Grosso, V., Standaert, F., Faust, S.: Masking vs. multiparty computation: How large is the gap for aes? In: Bertoni, G., Coron, J. (eds.) CHES 2013. LNCS, vol. 8086, pp. 400–416. Springer (2013), https://doi.org/10.1007/978-3-642-40349-1_23
23. Halevi, S.: Invertible universal hashing and the TET encryption mode. In: Menezes, A. (ed.) CRYPTO 2007. LNCS, vol. 4622, pp. 412–429. Springer (2007), https://doi.org/10.1007/978-3-540-74143-5_23
24. Horner, W.G.: Xxi. a new method of solving numerical equations of all orders, by continuous approximation. Philosophical Transactions of the Royal Society of London (109), 308–335 (1819)
25. Ishai, Y., Kushilevitz, E., Li, X., Ostrovsky, R., Prabhakaran, M., Sahai, A., Zuckerman, D.: Robust pseudorandom generators. In: Fomin, F.V., Freivalds, R., Kwiatkowska, M.Z., Peleg, D. (eds.) ICALP 201. LNCS, vol. 7965, pp. 576–588. Springer (2013), https://doi.org/10.1007/978-3-642-39206-1_49
26. Ishai, Y., Sahai, A., Wagner, D.A.: Private circuits: Securing hardware against probing attacks. In: Boneh, D. (ed.) CRYPTO 2003. LNCS, vol. 2729, pp. 463–481. Springer (2003), https://doi.org/10.1007/978-3-540-45146-4_27
27. Journault, A., Standaert, F.: Very high order masking: Efficient implementation and security evaluation. In: Fischer, W., Homma, N. (eds.) CHES 2017. LNCS, vol. 10529, pp. 623–643. Springer (2017), https://doi.org/10.1007/978-3-319-66787-4_30
28. Karpman, P., Roche, D.S.: New instantiations of the CRYPTO 2017 masking schemes. In: Peyrin, T., Galbraith, S.D. (eds.) ASIACRYPT 2018. LNCS, vol. 11273, pp. 285–314. Springer (2018), https://doi.org/10.1007/978-3-030-03329-3_10

29. Kim, H., Hong, S., Lim, J.: A fast and provably secure higher-order masking of AES s-box. In: Preneel, B., Takagi, T. (eds.) CHES 2011. LNCS, vol. 6917, pp. 95–107. Springer (2011), https://doi.org/10.1007/978-3-642-23951-9_7
30. McGrew, D.A., Viega, J.: The Galois/Counter Mode of Operation (GCM), http://luca-giuzzi.unibs.it/corsi/Support/papers-cryptography/gcm-spec.pdf
31. Naor, M., Reingold, O.: A pseudo-random encryption mode
32. Oshida, H., Ueno, R., Homma, N., Aoki, T.: On masked galois-field multiplication for authenticated encryption resistant to side channel analysis. In: Fan, J., Gierlichs, B. (eds.) COSADE 2018. LNCS, vol. 10815, pp. 44–57. Springer (2018), https://doi.org/10.1007/978-3-319-89641-0_3
33. Prouff, E., Rivain, M.: Masking against side-channel attacks: A formal security proof. In: Johansson, T., Nguyen, P.Q. (eds.) EUROCRYPT 2013. LNCS, vol. 7881, pp. 142–159. Springer (2013), https://doi.org/10.1007/978-3-642-38348-9_9
34. Reed, I.S., Solomon, G.: Polynomial codes over certain finite fields. Journal of the society for industrial and applied mathematics **8**(2), 300–304 (1960)
35. Rivain, M., Prouff, E.: Provably secure higher-order masking of AES. In: Mangard, S., Standaert, F. (eds.) CHES 2010. LNCS, vol. 6225, pp. 413–427. Springer (2010), https://doi.org/10.1007/978-3-642-15031-9_28
36. Segre, B.: Curve razionali normali ek-archi negli spazi finiti. Annali di Matematica Pura ed Applicata **39**(1), 357–379 (1955)
37. Seo, S.C., Kim, H.: SCA-resistant GCM implementation on 8-bit AVR microcontrollers. IEEE Access **7**, 103961–103978 (2019). https://doi.org/10.1109/ACCESS.2019.2930986
38. Wang, W., Méaux, P., Cassiers, G., Standaert, F.: Efficient and private computations with code-based masking. IACR Trans. Cryptogr. Hardw. Embed. Syst. **2020**(2), 128–171 (2020), https://doi.org/10.13154/tches.v2020.i2.128-171
39. Wegman, M.N., Carter, L.: New hash functions and their use in authentication and set equality. J. Comput. Syst. Sci. **22**(3), 265–279 (1981), https://doi.org/10.1016/0022-0000(81)90033-7