IACR Transactions on Cryptographic Hardware and Embedded Systems
ISSN 2569-2925, Vol. 2025, No. 1, pp. 104–127. DOI:10.46586/tches.v2025.i1.104-127

Bit t-SNI Secure Multiplication Gadget for Inner Product Masking

John Gaspoz and Siemen Dhooghe

COSIC, ESAT, KU Leuven, Leuven, Belgium
{john.gaspoz,siemen.dhooghe}@esat.kuleuven.be

Abstract. Masking is a sound countermeasure to protect against differential power analysis. Since the work by Balasch *et al.* in ASIACRYPT 2012, inner product masking has been explored as an alternative to the well known Boolean masking. In CARDIS 2017, Poussier *et al.* showed that inner product masking achieves higher-order security versus Boolean masking, for the same shared size, in the *bit-probing model*. Wang *et al.* in TCHES 2020 verified the inner product masking's security order amplification in practice and proposed new gadgets for inner product masking. Finally, Wu *et al.* in TCHES 2022 showed that this security amplification comes from the bit-probing model, but that Wang *et al.*'s gadgets are not higher-order bit-probing secure reducing the computation's practical security. The authors concluded their work with the open question of providing an inner product multiplication gadget which maintains the masking's bit-probing security, and conjectured that such gadget maintains the practical security order amplification of the masking during its computation.
In this paper, we answer positively to Wu *et al.*'s open problems. We are the first to present a multiplication gadget for inner product masking which is proven secure in the bit-level probing model using the t-Strong Non-Interference (SNI) property. Moreover, we provide practical evidence that the gadget indeed maintains the security amplification of its masking. This is done via an evaluation of an assembly implementation of the gadget on an ARM Cortex-M4 core. We used this implementation to take leakage measurements and show no leakage happens for orders below the gadget's bit-probing security level either for its univariate or multivariate analysis.

Keywords: Inner Product Masking · Masking · Non-interference · Probing Security · Side-channel Analysis · Software

1 Introduction

Cryptographic primitives implemented in hardware and software are known to be vulnerable to side-channel attacks such as Differential Power Analysis (DPA) [KJJ99] where an adversary exploits leaked information from the device (*e.g.*, the power consumption) in order to extract secret information. Masking is one of the most prevalent countermeasure against side-channel attacks. In its essence, a d^{th}-order Boolean masking encodes the sensitive processed values into $d+1$ shares by adding random values such that the observation of up to d shares is independent of the sensitive variables. In addition, masking transforms the elementary operations of the primitive's circuit into, so-called, masked gadgets which operate over the shares and provide the same functionality as the unmasked gadgets. In line with the concept of randomizing a secret value, a collection of masking types has been introduced over the past two decades. Maskings, such as the widely deployed and investigated Boolean masking [GP99, CJRR99], exploit simple encodings that

Received: 2024-07-15 Accepted: 2024-09-15 Published: 2024-12-09

facilitate efficient masked implementations. Alongside Boolean masking, maskings such as Inner Product Masking (IPM) [BFG+17], multiplicative masking [Gol02], polynomial masking [GM11], or the more general code-based masking such as (orthogonal) direct sum masking [BCC+14] have been investigated. While suffering from a typically larger overhead compared to Boolean masking, these other forms of masking benefit from enhanced security properties. In inner product masking, an n-sharing of a secret value x is computed given a public vector $L = (1, L_1, \dots, L_{n-1})$ as $(x_0 = x \oplus \sum_{i=1}^{n-1} L_i x_i,\, x_1,\, \dots,\, x_{n-1}) \in \mathbb{F}_{2^k}^n$. In contrast to Boolean masking, where each bit of the random values are directly XOR'd with the secret data, inner product masking XOR's various bits of the shares together via multiplication with the public vector L. As shown by Balasch *et al.* [BFG+17], inner product masking is secure against transition-based leakages and has a higher statistical security order (also known as 'security order amplification').

While the link between the type of masking and its security order was already observed by Balasch *et al.* [BFG+17], Poussier *et al.* [PGS+17] provide a formal reasoning of the security order amplification by introducing the bit-probing model and connecting inner product masking to direct sum masking. Inner product masking is rewritten from an error correction code viewpoint on the bit-level as $Z = X\mathbf{G} + Y\mathbf{H}$ where the sensitive data $X \in \mathbb{F}_{2^k}^m$ is masked into $Z \in \mathbb{F}_{2^k}^n$ by the randomness $Y \in \mathbb{F}_{2^k}^\ell$ after being transformed into code words using their respective generator matrices $\mathbf{G}$ and $\mathbf{H}$ constructed as follows

$$\mathbf{G} = \begin{pmatrix} 1 & 0 \dots 0 \end{pmatrix}, \qquad \mathbf{H} = \begin{pmatrix} L_1 & 1 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ L_{m-1} & 0 & \dots & 1 \end{pmatrix}.$$

The bit probing security of inner product masking can be related to the minimal dual distance of the $\mathbf{H}$ code in its direct sum masking representation. As a result, the best probing security leading to the security order amplification effect can be obtained by choosing an L vector such that the dual distance $d_{\mathbf{H}}^{\perp}$ is maximized.

Concerning the practical verification of the bit probing model, Wu *et al.* [WCG+22] showed that previous inner product masking gadgets did not maintain their security order amplification during their computation in practice and that these gadgets were not higher-order bit probing secure. They concluded with the open problem of providing gadgets for inner product masking maintaining their bit-probing security order.

1.1 Previous Works

Independently to this work, Wang *et al.* [WYS19] introduced the first code-based masking scheme with provable secure order amplification which employs a construction based on the re-masking of tensor products to securely compute the multiplication of shares. This approach allows for optimization in computation through the use of precomputed tables. However, while the bit-probing security of their gadgets is formally proven, the composability of their gadget is only supported by an informal security analysis. Additionally, Wang *et al.*'s security claim remains theoretical and no practical side-channel leakage evaluations support the theoretically predicted security order of the masked gadgets on a real-world device. In 2020, Wang *et al.* [WMCS20] introduced new gadgets to compute masked linear and nonlinear operations for a code-based masking (which includes inner product masking) which were proven to satisfy the Strong Non-Inference (SNI) security property by Barthe *et al.* [BBD+16]. The proposed multiplier follows a three-step procedure in which the inputs are first transformed into a matrix via a typical Boolean masking product-then-refresh operation followed by a temporary switch between code-based masking and additive sharing (*e.g.*, Boolean masking) and a multiplication with a pre-computed matrix. Next, the additive sharings are transformed back into codewords and a final compression step is performed to obtain the final codewords outputs. Unfortunately, the multiplication gadget suffers from a flaw. Wu *et al.* [WCG+22], by assessing an AES implementation based on this multiplication gadget, show that there is leakage in its implementation during the temporary transformation from code-based masking to additive masking. Moreover, the authors show that although the gadget satisfies word-level strong non-inference where probes view entire field elements, the multiplication gadget does not guarantee bit-level strong non-inference. This makes the authors conclude that there is a need for a masked multiplier that is able to maintain a consistent bit-probing security order through its computation.

1.2 Contributions

We positively answer the open question by Wu *et al.* [WCG+22] from TCHES 2022. Namely, we provide a new multiplication gadget for inner product masking which maintains the bit-probing security level of the masking. Namely, we show that the gadget is t-Strong Non-Interference (t–SNI) using the bit-probing model.

Moreover, we show that the implementation of our masked gadget also practically maintains the statistical order amplification of inner product masking's encoding therefore acknowledging the findings by Wu *et al.*. More specifically, we provide an ARM assembly implementation of our secure multiplier. This implementation is put on an ARM Cortex-M4 core where we performed practical leakage tests which show that the implementation does not leak (univariate or multivariate) in statistical orders not surpassing its bit-probing security level. More precisely, we evaluate a two-share four-bit assembly implementation with an expected second-order security given the best parameters for the inner product masking and show its successful practical results against first and second-order bivariate analysis. The assembly implementation is provided in the supplementary material of this submission and will be made public after publication.

Finally, we provide efficiency results of the inner product multiplication gadget and compare them to the results of using Boolean masking. We show that on top of inner product masking providing higher orders of security while using less memory for each sharing, it also outperforms Boolean masking in terms of the number of online operations or cycles on a microprocessor, however, it requires a significant prepossessing phase. While optimized inner product gadgets are not the focus of our work, we indicate the importance of research on secure inner product masking from a practical perspective.

2 Preliminaries

2.1 Notations

We represent elements of a field or vector space (in this work $\mathbb{F}_{2^k}$ and $\mathbb{F}_2^k$, respectively) by lower-case letters while we indicate matrices by bold uppercase letters. With $\oplus$, we denote the field addition in some binary field $\mathbb{F}_{2^n}$, and $+$ denotes the arithmetic addition. We represent the n-share masking of a secret $x \in \mathbb{F}_{2^k}$ as $\bar{x} = (x_0, \ldots, x_{n-1}) \in \mathbb{F}_{2^k}^n$ and use the notation $\bar{x}[i]$ to indicate the i^{th} bit of $\bar{x} \in \mathbb{F}_2^{nk}$. Individual shares are labeled as $s_j^i \in \mathbb{F}_{2^k}$ to denote the j^{th} share of the shares vector $\bar{s}^i = (s_0^i, \ldots, s_{n-1}^i)$. Given a positive integer n, we denote by $[n]$ the set $\{0, ..., n-1\}$.

In this work, we denote vectors as rows. For a matrix $\mathbf{A}$ and a vector m, $\mathbf{A} = [\mathbf{A}; m]$ is the concatenation of the rows of $\mathbf{A}$ and m. We write $\mathbf{A}[i, *]$ (resp., $\mathbf{A}[*, i]$) to denote the i^{th} row (resp., column) of $\mathbf{A}$. Note that the indexing of the matrix's columns increases from right to left in order to mimic standard binary representation ranging from the least significant bit to the most significant bit. Given two matrices $\mathbf{A}$ and $\mathbf{B}$ we denote $\mathbf{A} \odot \mathbf{B}$ the element-wise multiplication (or the Hadamard multiplication). We define the function

$v : \mathbb{F}_{2^k} \to \mathbb{F}_2^k$ such that given $x \in \mathbb{F}_{2^k}$, $v(x)$ produces the $\mathbb{F}_2^k$ representation of x in a polynomial basis. We provide the definition of a dual code $C^\perp$.

Definition 1 (Dual Code)**.** The dual code of a linear code $C \subset \mathbb{F}_{2^k}^n$ is the linear code defined by

$$C^\perp = \{x \in \mathbb{F}_{2^k}^n \mid \langle x, c\rangle = 0, \forall c \in C\}$$

where $\langle x, c\rangle = \sum_{i=1}^n x_i c_i$ is a scalar product.

Finally, given a linear code C, we denote the dual minimal distance $d_C^\perp$, meaning the minimum Hamming weight of non-zero codewords in $C^\perp$.

Definition 2 (Dual Distance)**.** The dual distance $d_C^\perp$ of a linear code $C \subset \mathbb{F}_{2^k}^n$ is the minimum Hamming weight of any non-zero codeword in the dual code $C^\perp$, *e.g.*,

$$d_C^\perp = \min\{wt(x) \mid x \in C^\perp, x \neq 0\}$$

where $wt(x)$ denotes the Hamming weight of x.

2.2 Boolean Masking

Boolean masking is a widely-deployed countermeasure against side-channel analysis which was introduced by Chari *et al.* [CJRR99] and Goubin-Patarin [GP99]. As a mean to conceal a secret Boolean variable $x \in \mathbb{F}_2$, a $(n-1)^{th}$-order Boolean masking splits such a variable into n shares $\bar{x} = (x_0, x_1, \ldots, x_{n-1})$ where shares $x_1, \ldots, x_{n-1}$ are drawn from a uniform distribution and x_0 is computed such that $x_0 = x \oplus \sum_{i=1}^{n-1} x_i$.

2.3 Inner Product Masking

Inner product masking was first introduced by Balasch *et al.* [BFG15, BFG+17] as an alternative to Boolean masking. Inner product masking is defined as follows.

Definition 3 (L-IPM)**.** An L-IPM of $x \in \mathbb{F}_{2^k}$ with a freely chosen public parameter $L = (1, L_1, ..., L_{n-1}) \in \mathbb{F}_{2^k}^n \backslash\{0\}$, is a vector $(x_0, ..., x_{n-1}) \in \mathbb{F}_{2^k}^n$ such that $x = \sum_{i=0}^{n-1} L_i x_i$.

Given the definition of inner product masking, we highlight the definition of a uniform masking.

Definition 4 (Uniform L-IPM)**.** Denote the set of all L-IPM vectors of $x \in \mathbb{F}_{2^k}$ by

$$IPM_L(x) = \{(x_0, ..., x_{n-1}) | (x_1, ..., x_{n-1}) \in \mathbb{F}_{2^k}^{n-1} \text{ s.t. } x_0 = x \oplus \sum_{i=1}^{n-1} L_i x_i\}\,.$$

We call an IPM masking "uniform" when the stochastic vector $(x_0, ..., x_{n-1})$ is uniformly randomly drawn from $IPM_L(x)$, *i.e.*, where $x_1, ..., x_{n-1}$ are uniformly random distributed.

2.4 Security Definitions

We recall the security models and notions considered in this paper. We differentiate between two types of security orders based on the level of granularity of probed information, namely word-level and bit-level probing models.

We begin with the definition of the probing adversary and security model by Ishai, Sahai and Wagner [ISW03].

Definition 5. (Probing Model) A t^{th}-order probing adversary $\mathscr{A}$ can learn a bounded number t of wires (called probes) of a circuit. A circuit is said to be t-probing secure if any set of at most t probes can be perfectly simulated by a simulator $\mathscr{S}$ without any knowledge of the input shares of the circuit (*i.e.*, the distribution of the probed values is independent of any secret value).

We distinguish two types of probes, namely word-level probes and bit-level probes.

Definition 6. (Word-level Probe) A word-level probe reveals an element over $\mathbb{F}_{p^k}$ (*e.g.*, a byte in the case of $\mathbb{F}_{2^8}$).

In a word-level probing model, an n-share inner product masking has a security order $t_w = n - 1$ [BFG+17]. Similarly, Boolean masking (which can be seen as a special case of IPM with public values $L_i = 1$) guarantees the same word-level probing security order.

Definition 7. (Bit-level Probe) A bit-level probe (bit-probe) reveals a single bit element regardless of the original field in which the shares are encoded.

In the works by Poussier *et al.* [PGS+17] and Cheng *et al.* [CGC+21], it is observed that inner product masking achieves higher-orders of security in the bit-level model versus Boolean masking (or versus the word-level model). This increase in security is called the 'security order amplification' effect.

The Non-Interference (NI) and Strong Non-Interference (SNI) definitions introduced by Barthe *et al.* [BBD+16] are used to quantify the security of gadget's composition. We adapt these definitions such that they work with bit-probes.

Definition 8. (t-Non-Interference) A gadget is called t–NI if for any set of t_{in} bit-probes on intermediate variables and every set of t_{out} bit-probes on output shares such that $t_{in} + t_{out} \leq t$, the totality of the probes can be simulated by t bit-probes on each input.

Definition 9. (t–Strong Non-Interference) A gadget is called t–SNI if for any set of t_{in} bit-probes on intermediate variables and every set of t_{out} bit-probes on output shares such that $t_{in} + t_{out} \leq t$, the totality of the probes can be simulated by only t_{in} bit-probes on each input.

Note that throughout the work the t–NI and t–SNI properties are evaluated at bit-level instead of the usual word level. In the usual word-level NI definitions, the bit-level probes are replaced by word-level probes.

2.5 Security Order Amplification of IPM

Certain parameters for inner product masking (and more generally code-based masking) have been shown to provide a beneficial impact on their security order. Poussier *et al.* [PGS+17] formalized this effect by identifying IPM as a particular case of Direct Sum Masking (DSM). In DSM, a sensitive variable $X \in \mathbb{F}_{2^k}^m$ is encoded as $Z = X\mathbf{G} + Y\mathbf{H}$ with a masked data $Z \in \mathbb{F}_{2^k}^n$ after being transformed into code words from C and D thanks to their respective generator matrices $\mathbf{G}$ and $\mathbf{H}$ for which the bit-probing security is given as follows.

Proposition 1 ([PGS+17])**.** *Let C and D be two codes of generator matrices $\mathbf{G}$ and $\mathbf{H}$ defining a DSM encoding. Let k and m, respectively, be the dimensions of C and D. The bit-probing security of the DSM encoding defined by C and D is equal to the minimal distance of the dual code (called the dual minimal distance) of D minus 1.*

Thus, the best bit-probing security t_b of an inner product masking is achieved by selecting a public vector L such that the dual minimal distance $d_L^{\perp}$ (abusing the notation where we denote L also as the code with generator matrix $(I|L)$) is maximized. This minimal distance also has an effect on the distribution of the bits in a uniform inner product masking as observed by Poussier *et al.* [PGS+17, Proposition 2].

Lemma 1. *For a uniform n-share L-IPM of $x \in \mathbb{F}_{2^k}$ given a public vector $L \in \mathbb{F}_{2^k}^n$, all sets of $t = d_L^\perp - 1$ bits are jointly uniform distributed.*

Following this coding-theoretic approach, Cheng *et al.* [CGC+21] proposed a framework based on the dual minimal distance and the kissing number to choose optimal codes for inner product maskings maximizing the security in the bit-probing model. Table 1 summarizes the optimal expected security order t_b given the number of shares n and k the bit size of the field.

Table 1: Some optimal parameters t_b for an IPM [CGC+21] for t_b the bit-probing order and t_w the word-probing order.

	$\mathbb{F}_{2^k}$	$d_L^\perp$	t_b	t_w
$n = 2$	$k = 4$	3	**2**	1
	$k = 8$	4	**3**	1
$n = 3$	$k = 4$	6	**5**	2
	$k = 8$	8	**7**	2

3 Multiplication Gadget

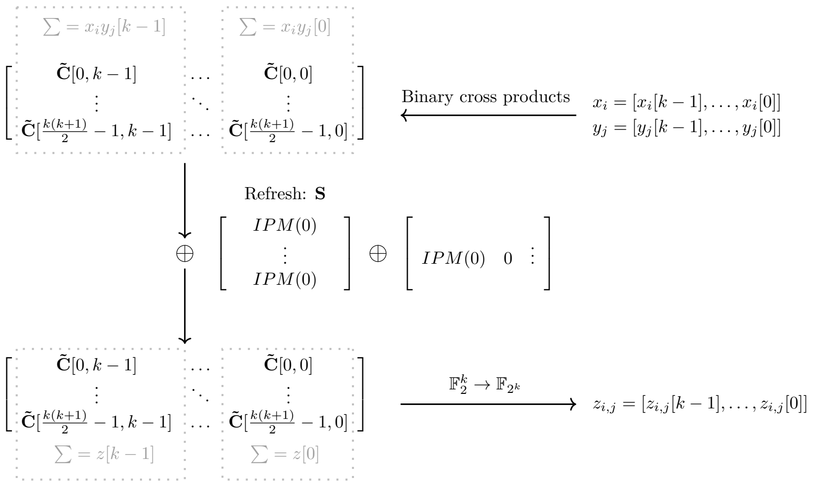
In this section, we introduce our t-SNI multiplication gadget tailored for inner product masking. Given two n-share inner product maskings $\bar{x}, \bar{y} \in \mathbb{F}_{2^k}^n$ of the secrets $x, y \in \mathbb{F}_{2^k}$, respectively, the gadget outputs a single n-share masking $\bar{z}$ such that $x \times y = z \in \mathbb{F}_{2^k}$.

3.1 Design Rationale

In order to preserve the security order amplification of the masking, a secure multiplication gadget must be proven secure in the bit-probing model. Let us first observe that a cross product $x_i y_j \in \mathbb{F}_{2^k}$ is constructed as

$$x_i y_j = \left(\sum_{u=0}^{k-1} x_i[u] 2^u \right) \times \left(\sum_{u=0}^{k-1} y_j[u] 2^u \right) .$$

Such a multiplication introduces an overflow of some binary cross products which are then reduced by the irreducible polynomial $q(x)$ defining the field $\mathbb{F}_{2^k}$. However, such a multiplication fails to provide the simple t-NI property (recall that we work in the bit-probing model) as the simulation of, for example, the most significant bit of $x_i y_j$ would require more than one bit of each shared input. Thus, in order to ensure the t-SNI security property, whenever nonlinear operations are performed between inner product maskings in $\mathbb{F}_{2^k}$, the multiplication needs to be operated securely at the binary level. To do so, each binary cross product needs to be securely re-masked while preserving the correctness and t-SNI security after the recomposition from $\mathbb{F}_2^k$ to $\mathbb{F}_{2^k}$. As illustrated in Figure 1, for every share $x_i, y_j \in \mathbb{F}_{2^k}$, the multiplication gadget computes all cross products $z_{i,j} = x_i y_j \in \mathbb{F}_{2^k}$ using a bitwise multiplication schoolbook algorithm placing every required binary cross products into a matrix $\tilde{\mathbf{C}}$. Overflowed cross products are managed using the irreducible polynomial $q(x)$. Hence, the binary cross products are placed into the matrix such that the XOR of all the elements at column k would result into one bit $z_{i,j}[k]$. Once the matrix is composed, a first layer of masking is applied to all binary components of the matrix in order to keep the value y secret using a uniform inner product masking of zero. Then, a second layer of re-masking is performed in order to keep the value x secret in a similar fashion. Finally, each cross product is mapped from $\mathbb{F}_2^k$ to $\mathbb{F}_{2^k}$ by summing the masked bits at each column resulting in the masked share $z_{i,j}$. To clarify, while security and coding properties are considered over $\mathbb{F}_2$, multiplication is performed over $\mathbb{F}_{2^k}$.



$\sum = x_i y_j[k-1]$
$\sum = x_i y_j[0]$
$\tilde{\mathbf{C}}[0, k-1]$
$\tilde{\mathbf{C}}[0,0]$
$\tilde{\mathbf{C}}[\frac{k(k+1)}{2}-1, k-1]$
$\tilde{\mathbf{C}}[\frac{k(k+1)}{2}-1, 0]$
Binary cross products
$x_i = [x_i[k-1], \dots, x_i[0]]$
$y_j = [y_j[k-1], \dots, y_j[0]]$
Refresh: $\mathbf{S}$
$IPM(0)$
$IPM(0)$
$IPM(0)$ 0
$\mathbb{F}_2^k \to \mathbb{F}_{2^k}$
$z_{i,j} = [z_{i,j}[k-1], \dots, z_{i,j}[0]]$
$\sum = z[k-1]$
$\sum = z[0]$

Figure 1: Illustration of Gadget-2

3.2 Detailed Description

Consider the inner product masking $\bar{x} = (x_0, \dots, x_{n-1})$ and $\bar{y} = (y_0, \dots, y_{n-1}) \in \mathbb{F}_{2^k}^n$ such that $x = \sum_{i=0}^{n-1} L_i x_i$ (resp., with y), and $q(x)$ the irreducible polynomial of the field $\mathbb{F}_{2^k}$.

The gadget is preceded by a preprocessing gadget (`Gadget-1` or Algorithm 1) which generates mask matrices $\mathbf{S}, \mathbf{R}_i$, and helper matrices $\mathbf{V}^p$ for the reduction mod $q(x)$. While the mask matrices $\mathbf{S}, \mathbf{R}_i$ need to be re-masked for each multiplication gadget, the helper matrices $\mathbf{V}^p$ can be re-used over all multiplication gadgets. This gadget also needs a secure generation of IPM sharings of zero (denoted $IPM_{free}(0)$) which is discussed in Section 3.4. After preprocessing, the multiplication is divided in two parts.

1. The first part (`Gadget-2` or Algorithm 2) computes every cross product $z_{i,j} = x_i y_j \in \mathbb{F}_{2^k}$. The multiplication follows the schoolbook $\mathbb{F}_{2^k}$ algorithm where every bit of x_i is sequentially multiplied with every bit of y_j followed by a shift operation. Afterwards, every overflowing cross product (*e.g.*, at index greater or equal to the bit size k) is reduced by the irreducible polynomial $q(x)$. Note that given the bit size k, the list of indices of binary cross products which are reduced (which overflow) is known and deterministic. Every required binary cross product is computed and placed in the matrices $(\mathbf{C}^0, \dots, \mathbf{C}^k)$ where $\mathbf{C}^0$ contains the non-overflow binary cross products whereas $\mathbf{C}^p$ for $p \geq 1$ are made of k copies per row of the overflowed binary cross products $x_i[i']y_j[j']$.

   The reduction mod $q(x)$ is ensured using the element-wise multiplication of the matrices $\mathbf{C}^{p>0}$ and $\mathbf{V}^{p>0}$. The end result is a cross product $z_{i,j}$ which is the $\mathbb{F}_{2^k}$ multiplication of x_i and y_j.

2. The second part (`Gadget-3` or Algorithm 3) compresses the cross products $z_{i,j}$. It multiplies the $z_{i,j}$ with L_j and sums them over the indices j to generate the output shares z_i.

3.3 Example

We illustrate a concrete example of the multiplication gadget over a two-share masking $\bar{x} = (x_0, x_1)$ and $\bar{y} = (y_0, y_1)$ over a field $\mathbb{F}_{2^3} \equiv \mathbb{F}_2[x]/x^3 + x + 1$.

`Gadget-1` Given the irreducible polynomial $q(x) = x^3 + x + 1$, the helper matrices $\mathbf{V}^1$ and $\mathbf{V}^2$ are constructed and filled with $v(q(x)) = (0, 1, 1)$ and $v(xq(x)) = (1, 1, 0)$, respectively. These matrices enable the reduction over $\mathbb{F}_{2^3}$.

$$\mathbf{V}^1 = \begin{bmatrix} 0 & 1 & 1 \\ 0 & 1 & 1 \end{bmatrix} \quad \mathbf{V}^2 = \begin{bmatrix} 1 & 1 & 0 \end{bmatrix}$$

The algorithm then generates 12 two-share IPMs of zero $(\bar{s}^0, \ldots, \bar{s}^{11})$ and 6 two-share IPMs of zero $(\bar{r}^0, \ldots, \bar{r}^5)$, each individual share consisting of three bits, from which the mask matrices are constructed as follows.

$$\mathbf{S}_{0,0} = \begin{bmatrix} s_0^0 \oplus r_0^0 \\ s_0^1 \oplus r_0^1 \\ s_0^2 \oplus r_0^2 \\ s_0^3 \oplus r_0^3 \\ s_0^4 \oplus r_0^4 \\ s_0^5 \oplus r_0^5 \end{bmatrix} \quad \mathbf{S}_{0,1} = \begin{bmatrix} s_1^0 \\ s_1^1 \\ s_1^2 \\ s_1^3 \\ s_1^4 \\ s_1^5 \end{bmatrix} \quad \mathbf{S}_{1,0} = \begin{bmatrix} s_0^6 \oplus r_1^0 \\ s_0^7 \oplus r_1^1 \\ s_0^8 \oplus r_1^2 \\ s_0^9 \oplus r_1^3 \\ s_0^{10} \oplus r_1^4 \\ s_0^{11} \oplus r_1^5 \end{bmatrix} \quad \mathbf{S}_{1,1} = \begin{bmatrix} s_1^6 \\ s_1^7 \\ s_1^8 \\ s_1^9 \\ s_1^{10} \\ s_1^{11} \end{bmatrix}$$

`Gadget-2` The second gadget takes as input some shares x_i, $y_j \in \mathbb{F}_{2^k}$, the pre-computed matrices $\mathbf{V}^1, \mathbf{V}^2$ and the matrix $\mathbf{S}_{i,j}$. As this gadget is called for every cross product, we only detail the multiplication of x_0 and y_0. The multiplication follows the schoolbook $\mathbb{F}_{2^k}$ algorithm where every bit of x_i is sequentially multiplied with every bit of y_j followed by a square operation. The reduction to the irreducible polynomial is computed in two stages. First, binary cross products are stored in different matrices $\mathbf{C}^0, \mathbf{C}^1$, and $\mathbf{C}^2$ depending on the need to reduce them by the irreducible polynomial. Namely, $\mathbf{C}^0$ holds binary cross products which do not require a reduction, $\mathbf{C}^1$ holds duplicates of binary cross products which require reduction by $q(x)$, and $\mathbf{C}^2$ holds duplicates of binary cross products which require reduction by $xq(x)$. The reduction by the irreducible polynomial is achieved via the element-wise multiplication between the matrices $\mathbf{C}^p$ and $\mathbf{V}^p$.

$$\mathbf{C^0} = \begin{bmatrix} x_0[0]y_0[2] & x_0[0]y_0[1] & x_0[0]y_0[0] \\ x_0[1]y_0[1] & x_0[1]y_0[0] & 0 \\ x_0[2]y_0[0] & 0 & 0 \end{bmatrix}$$

$$\mathbf{C^1} = \begin{bmatrix} x_0[1]y_0[2] & x_0[1]y_0[2] & x_0[1]y_0[2] \\ x_0[2]y_0[1] & x_0[2]y_0[1] & x_0[2]y_0[1] \end{bmatrix} \quad \mathbf{C}^1 \odot \mathbf{V}^1 = \begin{bmatrix} 0 & x_0[1]y_0[2] & x_0[1]y_0[2] \\ 0 & x_0[2]y_0[1] & x_0[2]y_0[1] \end{bmatrix}$$

$$\mathbf{C^2} = \begin{bmatrix} x_0[2]y_0[2] & x_0[2]y_0[2] & x_0[2]y_0[2] \end{bmatrix} \quad \mathbf{C}^2 \odot \mathbf{V}^2 = \begin{bmatrix} x_0[2]y_0[2] & x_0[2]y_0[2] & 0 \end{bmatrix}$$

The binary cross products from $\mathbf{C}^0$, $\mathbf{C}^1$, and $\mathbf{C}^2$ are gathered into a single matrix $\tilde{\mathbf{C}}$ for which the XOR of every row would yield $x_0 y_0 \in \mathbb{F}_{2^k}$.

$$\tilde{\mathbf{C}} = \begin{bmatrix} x_0[0]y_0[2] & x_0[0]y_0[1] & x_0[0]y_0[0] \\ x_0[1]y_0[1] & x_0[1]y_0[0] & 0 \\ x_0[2]y_0[0] & 0 & 0 \\ 0 & x_0[1]y_0[2] & x_0[1]y_0[2] \\ 0 & x_0[2]y_0[1] & x_0[2]y_0[1] \\ x_0[2]y_0[2] & x_0[2]y_0[2] & 0 \end{bmatrix}$$

Having constructed the binary cross product matrix for the correct computation, we describe how it is refreshed. Let us first detail the matrix $\mathbf{S_{0,0}}$ in its binary representation.

```
Algorithm 1 (Gadget-1) - Mask Matrices and Helper Matrices
Input: $q(x)$
Output: $\mathbf{S} \in \mathbb{F}_2^{\frac{nk(k+1)}{2} \times nk}$, $\mathbf{V}^p \in \mathbb{F}_2^{(k-p)\times k}$
  ▷ Construct the matrices $\mathbf{V}^p \in \mathbb{F}_2^{(k-p)\times k}$ (can be pre-computed once for all)
  for $p = 1$ to $k$ do
      for $i = 0$ to $k - p$ do
          $\mathbf{V}^p[i, *] = v(x^{p-1}q(x))$
      end for
  end for
  ▷ Compute $\bar{s}^i = (s_0^i, \ldots, s_{n-1}^i)$ $n$-share IPMs of 0
  for $i = 0$ to $\frac{nk(k+1)}{2}$ do
      $\bar{s}^i = IPM_{free}(0) \in \mathbb{F}_{2^k}^n$
  end for
  ▷ Compute $\bar{r}^j = (r_0^j, \ldots, r_{n-1}^j)$ $n$-share IPMs of 0
  for $j = 0$ to $\frac{k(k+1)}{2}$ do
      $\bar{r}^j = IPM_{free}(0) \in \mathbb{F}_{2^k}^n$
  end for
  ▷ Compute matrices $\mathbf{S}_{i,j}$
  for $i = 0$ to $n$ do
      for $j = 0$ to $n$ do
          for $\ell = 0$ to $\frac{k(k+1)}{2}$ do
              $\mathbf{S}_{i,j}[\ell, *] = s_j^{i\frac{k(k+1)}{2}+\ell}$
          end for
      end for
  end for
  ▷ Compute matrices $\mathbf{R}_i$ and update $\mathbf{S}_{i,0}$
  for $i = 0$ to $n$ do
      for $j = 0$ to $\frac{k(k+1)}{2}$ do
          $\mathbf{R}_i = [\mathbf{R}_i; r_i^j]$
      end for
      $\mathbf{S}_{i,0} = \mathbf{S}_{i,0} \oplus \mathbf{R}_i$
  end for
```

Recall that this matrix is constructed with the shares $(s_0^0, \ldots, s_0^5)$ and $(r_0^0, \ldots, r_0^5)$.

$$\mathbf{S}_{0,0} = \begin{bmatrix} s_0^0[2] \oplus r_0^0[2] & s_0^0[1] \oplus r_0^0[1] & s_0^0[0] \oplus r_0^0[0] \\ s_0^1[2] \oplus r_0^1[2] & s_0^1[1] \oplus r_0^1[1] & s_0^1[0] \oplus r_0^1[0] \\ s_0^2[2] \oplus r_0^2[2] & s_0^2[1] \oplus r_0^2[1] & s_0^2[0] \oplus r_0^2[0] \\ s_0^3[2] \oplus r_0^3[2] & s_0^3[1] \oplus r_0^3[1] & s_0^3[0] \oplus r_0^3[0] \\ s_0^4[2] \oplus r_0^4[2] & s_0^4[1] \oplus r_0^4[1] & s_0^4[0] \oplus r_0^4[0] \\ s_0^5[2] \oplus r_0^5[2] & s_0^5[1] \oplus r_0^5[1] & s_0^5[0] \oplus r_0^5[0] \end{bmatrix}$$

Next, every binary cross product of $\tilde{\mathbf{C}}$ is refreshed with the matrix $\mathbf{S}_{0,0}$.

$$\begin{bmatrix} x_0[0]y_0[2] \oplus s_0^0[2] \oplus r_0^0[2] & x_0[0]y_0[1] \oplus s_0^0[1] \oplus r_0^0[1] & x_0[0]y_0[0] \oplus s_0^0[0] \oplus r_0^0[0] \\ x_0[1]y_0[1] \oplus s_0^1[2] \oplus r_0^1[2] & x_0[1]y_0[0] \oplus s_0^1[1] \oplus r_0^1[1] & s_0^1[0] \oplus r_0^1[0] \\ x_0[2]y_0[0] \oplus s_0^2[2] \oplus r_0^2[2] & s_0^2[1] \oplus r_0^2[1] & s_0^2[0] \oplus r_0^2[0] \\ s_0^3[2] \oplus r_0^3[2] & x_0[1]y_0[2] \oplus s_0^3[1] \oplus r_0^3[1] & x_0[1]y_0[2] \oplus s_0^3[0] \oplus r_0^3[0] \\ s_0^4[2] \oplus r_0^4[2] & x_0[2]y_0[1] \oplus s_0^4[1] \oplus r_0^4[1] & x_0[2]y_0[1] \oplus s_0^4[0] \oplus r_0^4[0] \\ x_0[2]y_0[2] \oplus s_0^5[2] \oplus r_0^5[2] & x_0[2]y_0[2] \oplus s_0^5[1] \oplus r_0^5[1] & s_0^5[0] \oplus r_0^5[0] \end{bmatrix}$$

Algorithm 2 (Gadget-2) - Binary Cross Products

```
Input: $x, y \in \mathbb{F}_{2^k}$, $\mathbf{V}^p \in \mathbb{F}_2^{(k-p)\times k}$, $\mathbf{S} \in \mathbb{F}_2^{\frac{k(k+1)}{2}\times k}$
Output: $z = xy \in \mathbb{F}_{2^k}$
  ▷ Compute $\mathbf{C}^0 \in \mathbb{F}_2^{k\times k}$ initialized at 0
  for $i = 0$ to $k$ do
      for $j = 0$ to $(k-i)$ do
          $\mathbf{C}^0[i, i+j] = x[i]y[j]$
      end for
  end for
  ▷ Compute matrices $\mathbf{C}^p \in \mathbb{F}_2^{(k-p)\times k}$ initialized at 0
  for $p = 1$ to $k$ do
      for $i = p$ to $k$ do
          $m = (x[i]y[j], \dots, x[i]y[j]) \in \mathbb{F}_2^k$, with $j = (k-1-i)+p$
          $\mathbf{C}^p = [\mathbf{C}^p; m]$
      end for
      $\mathbf{C}^p = \mathbf{C}^p \odot \mathbf{V}^p$
  end for
  ▷ Compute $\tilde{\mathbf{C}}$ initialized at 0
  for $p = 0$ to $k$ do
      $\tilde{\mathbf{C}} = [\tilde{\mathbf{C}}; \mathbf{C}^p]$
  end for
  $\tilde{\mathbf{C}} = \tilde{\mathbf{C}} \oplus \mathbf{S}$
  ▷ Compute $z$
  $z = \sum_{i=0}^{\frac{k(k+1)}{2}-1} \tilde{\mathbf{C}}[i, *]$
```

Lastly, every row of $\tilde{\mathbf{C}}$ is XOR'd together and the k bit vector is mapped from $\mathbb{F}_2^k$ to its field representation $\mathbb{F}_{2^k}$ to obtain the masked cross product

$$z_{0,0} = x_0 y_0 \oplus (s_0^0 \oplus \dots \oplus s_0^5) \oplus (r_0^0 \oplus \dots \oplus r_0^5) \in \mathbb{F}_{2^k}.$$

Algorithm 3 (Gadget-3) - Secure Multiplication

```
Input: $\bar{x}, \bar{y} \in \mathbb{F}_{2^k}^n$, $\mathbf{V}^p \in \mathbb{F}_2^{(k-p)\times k}$, $\mathbf{S} \in \mathbb{F}_2^{\frac{nk(k+1)}{2}\times nk}$
Output: $\bar{z} \in \mathbb{F}_{2^k}^n$ such that $\sum_i L_i z_i = (\sum_i L_i x_i)(\sum_i L_i y_i)$
  ▷ Compute $z_{i,j}$
  for $i = 0$ to $n$ do
      for $j = 0$ to $n$ do
          $z_{i,j} = \texttt{Gadget-2}(x_i, y_j, \mathbf{S}_{i,j}, \mathbf{V})$
      end for
  end for
  ▷ Computation of the output shares $z_i$
  for $i = 0$ to $n$ do
      $z_i = \sum_{j=0}^{n-1} L_j z_{i,j}$ (over $\mathbb{F}_{2^k}$)
  end for
```

`Gadget-3` The last gadget receives the generated matrices from `Gadget-1` and iteratively calls `Gadget-2` to compute the intermediate masked cross products.

$$z_{0,0} = \texttt{Gadget-2}(x_0, y_0, \mathbf{S}_{0,0}, \mathbf{V}) = x_0 y_0 \oplus (s_0^0 \oplus \dots \oplus s_0^5) \oplus (r_0^0 \oplus \dots \oplus r_0^5)$$

$$z_{0,1} = \texttt{Gadget-2}(x_0, y_1, \mathbf{S}_{0,1}, \mathbf{V}) = x_0 y_1 \oplus (s_1^0 \oplus \ldots \oplus s_1^5)$$
$$z_{1,0} = \texttt{Gadget-2}(x_1, y_0, \mathbf{S}_{1,0}, \mathbf{V}) = x_1 y_0 \oplus (s_0^6 \oplus \ldots \oplus s_0^{11}) \oplus (r_1^0 \oplus \ldots \oplus r_1^5)$$
$$z_{1,1} = \texttt{Gadget-2}(x_1, y_1, \mathbf{S}_{1,1}, \mathbf{V}) = x_1 y_1 \oplus (s_1^6 \oplus \ldots \oplus s_1^{11})$$

It then compresses the cross products to compute the output shares $z_0 = z_{0,0} \oplus L_1 z_{0,1}$, $z_1 = z_{1,0} \oplus L_1 z_{1,1}$.

3.4 Securely Generating $IPM_{free}(0)$

Algorithm 1 requires the generation of inner product maskings of zero which are used to refresh shared values. We explain how this can be done securely. We first provide a security notion for a gadget generating a zero-sharing by the work of Belaïd *et al.* [BCRT23]. We slightly adapt this notion such that it works for non-Boolean maskings following Definition 4.

Definition 10 (Free Encoding of Zero [BCRT23])**.** Let G be a gadget without input and a single n-shared output z where any set of k bits are uniformly distributed. G is said to be t-free if for every set W of (internal and output) probed wires of G with $|W| \leq t$, there exists a set V of cardinality $|V| \leq |W|$ such that for every set $O \subsetneq [1:n]/V$, any set of $k - |W|$ bits of $z|_O$ is uniformly distributed and mutually independent of the probed values on W and the outputs $z|_V$.

The above definition needs to hold for the gadget generating the $IPM_{free}(0)$ sharing from Algorithm 1 as it is used in the bit-probing SNI proof in Section 4.

In this work, we propose a specific t-free secure IPM sharing of zero. Namely, we use the work by Ishai *et al.* [IKL+13] and create $t+1$ regular $IPM(0)$ sharings and then XOR them. For clarity, we call the t-free gadget $IPM_{free}(0)$ (which is the XOR of $IPM(0)$ sharings) and the regular generation we denote as $IPM(0)$. We show that this method is t-free.

Lemma 2. *The XOR of $t+1$ generated $IPM(0)$ is t-free from Definition 10.*

Proof. We assume that each regular generated $IPM(0)$ has the property that any set of k bits are uniformly distributed. We need to show that given $t' \leq t$ probes, there exists a set V of output bits such that any set of $k - t'$ bits of the XOR of $t+1$ $IPM(0)$ sharings is uniformly distributed and mutually independent of the probed values and the V outputs.

The set V is simply constructed by either adding nothing to V when a simple $IPM(0)$ is probed or when the XOR between the $IPM(0)$ is probed, that XOR's output leads to a single output bit of the gadget which is thus added to V. Given that $t+1$ $IPM(0)$ are XORed and that there are only t bit-probes, there is at least one $IPM(0)$ which is not probed and which provides the uniformity of the output following Definition 4 of the non-probed outputs of V. □

Since solely probing the creation of an $IPM_{free}(0)$ sharing does not provide secret information (at least one probe should be placed on secret information), we only require a $(t-1)$-free generator. Nevertheless, the secure creation of a single secure $IPM_{free}(0)$ sharing now requires t times the cost over the naive insecure approach.

4 Correctness and t-SNI Security Proofs

In this section, we provide formal proofs of correctness and security for the multiplication gadget (Algorithm 3) where we show that the gadget has compositional security in the t-strong non-interference framework with the bit-probing model.

Lemma 3. *Algorithm 2, with inputs x_i, y_j and $S_{i,j}$, outputs $z = x_i y_j \oplus s_j \oplus r_i \in \mathbb{F}_{2^k}$ for $j = 0$ and $z = x_i y_j \oplus s_j \in \mathbb{F}_{2^k}$ for $j > 0$ with r_i the i^{th} share of some $IPM_{free}(0)$ and s_j the j^{th} share of some $IPM_{free}(0)$.*

Proof. We find the following equalities

$$\begin{aligned}
z[\ell] &= \sum_{t=0}^{\frac{k(k+1)}{2}-1} \tilde{\mathbf{C}}[t,\ell] = \sum_{t=0}^{k-1} \tilde{\mathbf{C}}^0[t,\ell] \oplus \sum_{p=1}^{k-1} \sum_{t=0}^{k-1-p} \tilde{\mathbf{C}}^p[t,\ell] \oplus \sum_{t=0}^{\frac{k(k+1)}{2}-1} \mathbf{S}[t,\ell] \\
&= \sum_{t=0}^{\ell} x_i[t] y_j[\ell - t] \oplus \sum_{p=1}^{k-1} \sum_{t=p}^{k-1} (x_i[t] y_j[k-1-t+p]) v(x^{p-1} q(x))[\ell] \oplus \sum_{t=0}^{\frac{k(k+1)}{2}-1} \mathbf{S}[t,\ell] \\
&= (x_i y_j)[\ell] \oplus \sum_{t=0}^{\frac{k(k+1)}{2}-1} \mathbf{S}[t,\ell] = \begin{cases} (x_i y_j)[\ell] \oplus \sum_{t=0}^{\frac{k(k+1)}{2}-1} (s_j^{i\frac{k(k+1)}{2}+t}[\ell] \oplus r_i^t[\ell]) & j = 0\,, \\ (x_i y_j)[\ell] \oplus \sum_{t=0}^{\frac{k(k+1)}{2}-1} s_j^{i\frac{k(k+1)}{2}+t}[\ell] & j \neq 0\,, \end{cases}
\end{aligned}$$

where the second-to-last equation comes from the schoolbook multiplication over $\mathbb{F}_{2^k}$.

Since the sum of separate inner product maskings of zero remains an inner product masking of zero, we have that $\sum_{t=0}^{\frac{k(k+1)}{2}-1} s_j^{i\frac{k(k+1)}{2}+t}[\ell]$ is the j^{th} share of an inner product masking of zero. Similar, for $\sum_{t=0}^{\frac{k(k+1)}{2}-1} r_i^t[\ell]$ which is the i^{th} share of some other inner product masking of zero. □

Theorem 1. *Algorithm 3 is a correct gadget for masked inputs, providing a masking of the multiplication of the secrets over $\mathbb{F}_{2^k}$.*

Proof. Given Lemma 3, we know that $z_{i,0} = x_i y_0 \oplus s_0 \oplus r_i$ and $z_{i,j} = x_i y_j \oplus s_j$ for $j > 0$ and s_j the j^{th} share of some inner product masking of zero (resp, r_i the i^{th} share of another one). As a result, we find that

$$\begin{aligned}
z_i = \sum_{j=0}^{n-1} L_j z_{i,j} &= x_i y_0 \oplus s_0 \oplus r_i \oplus \sum_{j=1}^{n-1} (L_j x_i y_j \oplus L_j s_j) \\
&= x_i (y_0 \oplus \sum_{j=1}^{n-1} L_j y_j) \oplus (s_0 \oplus \sum_{j=1}^{n-1} L_j s_j) \oplus r_i = x_i y \oplus r_i\,.
\end{aligned}$$

Notice then that

$$z_0 \oplus \sum_{i=1}^{n-1} L_i z_i = x_0 y \oplus r_0 \oplus \sum_{i=1}^{n-1} L_i (x_i y \oplus r_i) = (x_0 \oplus \sum_{i=1}^{n-1} L_i x_i) y \oplus (r_0 \oplus \sum_{i=1}^{n-1} L_i r_i) = xy\,.$$

Thus, Algorithm 3 correctly implements an inner-product masked multiplication. □

Theorem 2. *Algorithm 3 is t-SNI with $t = d_L^{\perp} - 1$.*

Proof. The security proof essentially follows from the potential observations the adversary can make, which are depicted in Figures 4a and 4b, and the shape of the refreshing matrix $\mathbf{S}$, which is depicted in Figures 2 and 3a, which consists of both horizontal and vertical inner product maskings of zero. In the security proof, we show that the $\mathbf{R}_i$ matrices, together with the $\mathbf{S}_{i,j}$ matrices, refresh the input shares and that each probe can only observe a single bit of the $\mathbf{R}_i$'s independent $IPM_{free}(0)$ maskings. Due to Lemma 1 which shows that up to t bits in an inner product masking with dual minimal distance $t+1$ are jointly uniform distributed, the multiplication gadget is shown to be t^{th}-order SNI secure.

Let $\Omega = (\mathcal{I}, \mathcal{O})$ be a set of t observations on the internal and on the output bits, respectively, where $|\mathcal{I}| = t_1$ such that $t_1 + |\mathcal{O}| \leq t$. We construct a perfect simulator $\mathcal{S}$ of the adversary's bit-probes, which can make use of at most t_1 bits of $\bar{x}$ and $\bar{y}$. Let $w_1, \ldots, w_t$ be the set of probed wires. We classify the internal wires in the following groups:

(1) $\bar{x}[i], \bar{y}[i]$ (the i^{th}-bit of the masking)
(2) $\bar{s}^i[j], \bar{r}^j[i], \mathbf{S}_{i',j'}[i,j], \mathbf{R}_{i',j'}[i,j]$
(3) A wire in $IPM_{free}(0)$ for $\bar{s}^i$.
(4) A wire in $IPM_{free}(0)$ for $\bar{r}^j$.
(5) $\tilde{\mathbf{C}}^*[*,*] = \bar{x}[i]\bar{y}[j] \oplus \mathbf{S}_{*,*}[*,*]$
(6) Any sum of $\tilde{\mathbf{C}}[*,\ell]$ to create $z_{i,j}[\ell]$
(7) $(L_j z_{i,j})[\ell]$
(8) Any sum of $(L_j z_{i,j})[\ell]$ to create $z_i[\ell]$

We define two sets of indices I and J with $|I| \leq t_1$, $|J| \leq t_1$, such that the values of the probed wires, denoted w_h with $h = 1, \ldots, t$, can be perfectly simulated given only the knowledge of $\bar{x}[i]_{i \in I}$ and $\bar{y}[j]_{j \in J}$. The sets are constructed as follows.

– Initially I and J are empty.
– For every wire as in the group (1) add i to I and J.
– For every wire as in the group (2) and (5) add i to I and j to J.
– For every wire as in group (3) following Definition 10, the set V (of cardinality at most the number of probed wires) is added to J and i to I.
– For every wire as in group (4) following Definition 10, the set V (of cardinality at most the number of probed wires) is added to I and j to J.
– For every wire as in the group (6), (7), and (8) nothing is added to I and J.

Since the adversary is allowed to make at most t_1 internal probes, we have that $|I| \leq t_1$ and $|J| \leq t_1$.

We now show that the simulator $\mathcal{S}$ is able to simulate any internal wire w_h as follows.

1. For each observation as in category (1), then $i \in I, J$ and by definition the simulator has access to $\bar{x}[i]$ or $\bar{y}[i]$. Hence the values are perfectly simulated.
2. For each observation as in category (2), the simulator assigns a random and independent value to the observed wire. For a probe on $\mathbf{S}_{i',0}[i,j]$ the simulator assigns two random and independent values for $s_0^{i'\frac{k(k+1)}{2}+i}[j]$ and for $r^i_{i'}[j]$.
3. For each observation as in categories (3) or (4), following Definition 10, the simulator assigns a random and independent value to the observed wire and the output specified by the set V. We then know that the other shares of that $IPM_{free}(0)$ are still uniform and independent of the observed values, thus acting as independent uniform randomness.

   We note that in case categories (2)-(4) were probed t times, the simulator can simply generate all probed values as random following the algorithm since no input shares are needed for this generation. Thus, we assume that categories (2)-(4) are probed at most $t-1$ times.
4. For each observation as in category (5), by definition the simulator has access to $\bar{x}[i]$ and $\bar{y}[j]$. The value $\bar{x}[i]\bar{y}[j]$ can be computed as in the real algorithm and the randomness $\mathbf{S}_{*,*}[*,*]$ is either freshly generated or repeated if it was already probed in categories (2)-(4).
5. For each observation as in category (6), if one of the $\tilde{\mathbf{C}}[*,\ell]$ or their respective randomness in $\mathbf{S}$ composing $z_{i,j}[\ell]$ has already been probed, then we can simulate it as in the previous steps. Otherwise, from viewing the bits $\tilde{\mathbf{C}}[*,\ell]$, a probe in this category views only one column in an $\mathbf{S}_{i,j}$ block. Furthermore, from Figure 2, we

observe that a single column of the $\mathbf{S}_{i,j}$ block constitutes single bits of separate and independent (horizontal) IPMs of zero. As a result, a single column of $\mathbf{S}_{i,j}$ is jointly uniform random. Thus, all the $\tilde{\mathbf{C}}[*,\ell]$ in the sum of $z_{i,j}[\ell]$ are jointly uniform random and can be simulated as such.

$$\mathbf{S} = \overset{IPM(0)}{\begin{pmatrix} \mathbf{R}_0 & \dots & 0 \\ \mathbf{R}_1 & \dots & 0 \\ \vdots & \ddots & \vdots \\ \mathbf{R}_{n-1} & \dots & 0 \end{pmatrix}} \oplus \overset{IPM(0)}{\begin{pmatrix} \mathbf{S}_{0,0} & \dots & \mathbf{S}_{0,n-1} \\ \mathbf{S}_{1,0} & \dots & \mathbf{S}_{1,n-1} \\ \vdots & \ddots & \vdots \\ \mathbf{S}_{n-1,0} & \dots & \mathbf{S}_{n-1,n-1} \end{pmatrix}}$$

Figure 2: The shape of the $\mathbf{S}$ matrix. The shares of the $IPM_{free}(0)$ from the $\mathbf{R}_i$ matrices are stored in their rows.

$$\overset{i^{\text{th}}\text{-share of an } IPM(0)}{\begin{pmatrix} r_i^0[k-1] & \dots & r_i^0[0] \\ r_i^1[k-1] & \dots & r_i^1[0] \\ \vdots & \ddots & \vdots \\ r_i^{\frac{k(k+1)}{2}}[k-1] & \dots & r_i^{\frac{k(k+1)}{2}}[0] \end{pmatrix}}$$

(a) $\mathbf{R}_i$ block.

$$\overset{j^{\text{th}}\text{-share of an } IPM(0)}{\begin{pmatrix} s_j^{im}[k-1] & \dots & s_j^{im}[0] \\ s_j^{im+1}[k-1] & \dots & s_j^{im+1}[0] \\ \vdots & \ddots & \vdots \\ s_j^{(i+1)m-1}[k-1] & \dots & s_j^{(i+1)m-1}[0] \end{pmatrix}}$$

(b) $\mathbf{S}_{i,j}$ block with $m = \frac{k(k+1)}{2}$.

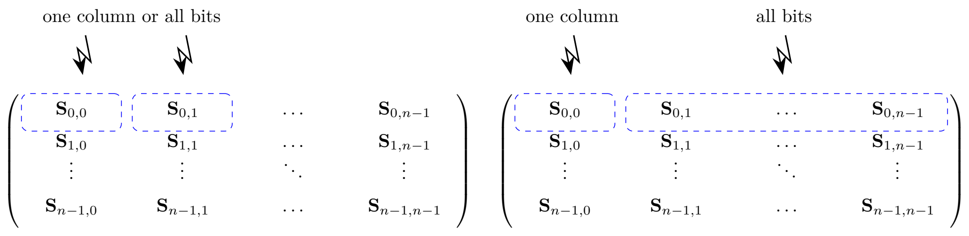
Figure 3: Layout of the bits in a $\mathbf{R}_i$ or $\mathbf{S}_{i,j}$ block.

6. For each observation as in category (7), we assume a worst case scenario where such a probe reveals every bit of $z_{i,j}$ (unless $j = 0$) since, in the worst case, L_j is such that it combines all bits of $z_{(i,j)}$ (*e.g.*, if the representation of $L_j \in \mathbb{F}_{2^k}$ as a $\mathbb{F}_2^{k\times k}$ matrix has a row full of ones). Thus, for $j > 0$, we assume a probe in this category views a full $\mathbf{S}_{i,j}$ block. In case $j = 0$, then L_0 is the identity matrix and only one column of $\mathbf{S}_{i,0}$ is observed.

   The adversary only has t probes and each probe either views one $\mathbf{S}_{i,j}$ block for $j > 0$ or a single column of $\mathbf{S}_{i,0}$. Since a row of blocks of the $\mathbf{S}$ matrix consists of (horizontal) inner product maskings of zero, from Definition 4 we know that all $\mathbf{S}_{i,j}$ blocks for $j > 0$ are jointly uniform distributed. From the addition of the $\mathbf{R}_i$ matrix (which is one share of vertical IPMs of zero following Figure 2), the $\mathbf{S}_{i,0}$ block is jointly uniform with all $\mathbf{S}_{i,j}$ blocks for $j > 0$. Since the adversary can only place t probes and each probe views at most one bit of each (vertical) inner product masking from the $\mathbf{R}_i$ matrices (see Figure 3a), it follows from Lemma 1 that all the bits from the $\mathbf{R}_i$ are jointly uniform random. As a result, the observations in category (7) can be simulated as joint uniform randomness unless one of the values was already probed before. More specifically, in case any bit $\mathbf{R}_i[\ell_0,\ell_1]$ was already probed, as in categories (2)-(4), the simulator was given the indices $\ell_0 \in I, \ell_1 \in J$ to simulate $x[\ell_0]y[\ell_1]$.

7. For each observation as in category (8), following Figure 4b, a probe observes all blocks $\mathbf{S}_{i,j}$ for $j > 0$ and a column of $\mathbf{S}_{i,0}$.

   The proof follows the same steps as in category (7), namely all blocks $S_{i,j}$ for $j > 0$ together with a column from $S_{i,0}$ are jointly uniform random due to the $S_{i,0}$ block being masked with R_i. Since each probe in this category only views a single column of $S_{i,0}$, each probe only views a single bit of the inner product maskings of zero that are placed in the $\mathbf{R}_i$ matrices (see Figure 3a). From Lemma 1 and the restriction of the adversary to maximally place t probes, it follows that the probed bits of the $\mathbf{R}_i$ matrices are jointly uniform random. As a result, up to t rows of blocks $S_{*,j}$ for $j > 0$ with t columns from the blocks $S_{*,0}$ are jointly uniform random and can be simulated as such. In case any bit $\mathbf{R}_i[\ell_0, \ell_1]$ was already probed, as in categories (2)-(4), the simulator was given the indices $\ell_0 \in I, \ell_1 \in J$ to simulate $x[\ell_0]y[\ell_1]$.


one column or all bits
one column
all bits
$\mathbf{S}_{0,0}$ $\mathbf{S}_{0,1}$ $\dots$ $\mathbf{S}_{0,n-1}$
$\mathbf{S}_{1,0}$ $\mathbf{S}_{1,1}$ $\dots$ $\mathbf{S}_{1,n-1}$
$\mathbf{S}_{n-1,0}$ $\mathbf{S}_{n-1,1}$ $\dots$ $\mathbf{S}_{n-1,n-1}$

(a) Probing category (6) or (7).

(b) Probing category (8).

Figure 4: The observed bits of $\mathbf{S}$ when probing category (6)-(8). For category (6)-(7), the adversary either gets one bit column in $\mathbf{S}_{i,0}$ (one column shown in Figure 3b) or all bits in $\mathbf{S}_{i,j}$ with $j > 0$. For category (8), we get one bit column and all bits in $\mathbf{S}_{i,j}$ with $j > 0$.

Since the simulation of category (8) added no extra information to the simulator, we also proved that the output of the gadget can be simulated using no additional information. $\square$

5 Efficiency

In this section, we discuss both the algorithmic and practical performances of Gadget-3 and compare it with the standard multiplication gadget for Boolean masking proposed by Ishai *et al.* [ISW03] (referred to as ISW in the rest of the paper).

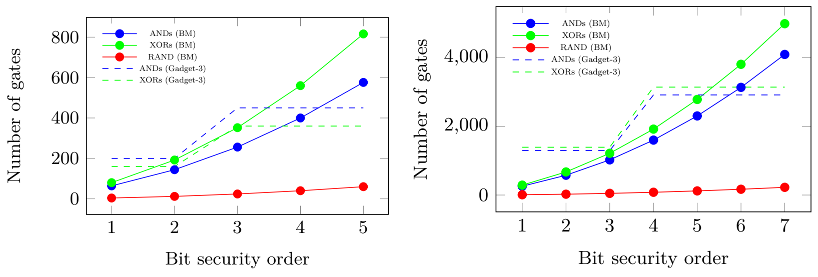
5.1 Algorithmic Efficiency

For a given number of shares n in a field $\mathbb{F}_{2^k}$, the comparison uses the number of random bits as well as the number of AND and XOR gates as metrics. ISW requires n^2 field multiplications. Given that we want to provide asymptotic metrics, we translate each field multiplication of ISW into a number of binary gates. Following the results of [RH04], we provide the optimistic estimation that a multiplier in $\mathbb{F}_{2^k}$ requires k^2 AND gates and k^2 XOR gates. The number of gates in Gadget-3 corresponds to the count of (unoptimized) assembly instructions utilized in the algorithm. At first sight, the results in Table 2 appear to indicate lower metrics for the ISW multiplication gadget. However, keep in mind that, due to the security order amplification by inner product masking, Gadget-3 can achieve a higher security order while keeping n relatively small compared to Boolean masking where the number of shares increases with the security order.

Figure 5a shows the results obtained for the field $\mathbb{F}_{2^4}$ with colored full and dashed lines corresponding to ISW and Gadget-3, respectively. Similarly, Figure 5b shows the results obtained for the field $\mathbb{F}_{2^8}$. The expected security order for inner product masking follows the metrics provided in Table 1 where two-share and three-share maskings reach a maximum bit-level security order of three and seven, respectively. Thus, in terms of gate count, we observe that both implementations utilize a fairly similar number of gates for third-order security. However, for fifth-order security for the field $\mathbb{F}_{2^4}$ (resp. sixth-order security for the field $\mathbb{F}_{2^8}$) achieved with a three-share inner product masking, the ISW multiplier exhibits a higher number of gates compared to our Gadget-3 multiplier. The major limitation of Gadget-3 is the large number of random bits required compared to ISW. As observed in Figure 6, Gadget-1 generates numerous secure $IPM_{free}(0)$ instances (detailed in Section 3.4), each necessitating multiple random bits and encoding via inner product masking, which involves a substantial number of XOR gates.

Table 2: Algorithmic cost in the number of XOR gates, AND gates, and fresh random bits of Gadget-1 (without the generation of $\mathbf{V}^p$) and Gadget-3 compared to the ISW multiplication in $\mathbb{F}_{2^k}$ for n (word-level) shares. We denote t the bit-probing security level of the gadget.

	#ANDs	#XOR's	#Random bits
ISW	n^2k^2	$k^2n^2 + 2nk(n-1)$	$\frac{1}{2}kn(n-1)$
Gadget-3	$\frac{1}{2}k(k+1)^2n^2$	$\frac{1}{2}kn(k^2n + 3kn - 2)$	-
Gadget-1	-	$\frac{1}{2}k^2(k+1)(kn^2t - kt + n)$	$\frac{1}{2}t(n^2-1)k^2(k+1)$



(a) For the field $\mathbb{F}_{2^4}$. **(b)** For the field $\mathbb{F}_{2^8}$.

Figure 5: Comparison between the number of gates and random bits required in ISW and Gadget-3 over the same security orders (where its security order increases with the number of shares n) over different fields.

5.2 Practical Efficiency

We present in Table 3 a clock cycles comparison of Gadget-3's implementation with the standard ISW multiplication gadget for Boolean masking in $\mathbb{F}_{2^4}$. Since our gadget follows a schoolbook algorithm in order to compute a multiplication between two shares, we first compare it to ISW which also uses a schoolbook Galois Field multiplication. We observe that the second-order secure Gadget-3 outperforms the schoolbook ISW for a similar degree of security. This confirms the numbers from the algorithmic efficiency in Table 2. Nevertheless, ISW benefits from software implementation optimizations, such as the utilization of log-a-log tables to calculate the cross products which computes in fewer cycles than Gadget-3 for the two-share case. However, the three-share variant of Gadget-3 enables a bit security order up to 5 and results in the minimal number of cycles – surpassing even the log-a-log optimized ISW algorithm. Nevertheless, these positive results must be tempered by the high cost of randomness generation required for Gadget-1, which is currently the bottleneck of the multiplication gadget. We believe that optimizations might be applicable to Gadget-3 and Gadget-1 and leave them as future work.

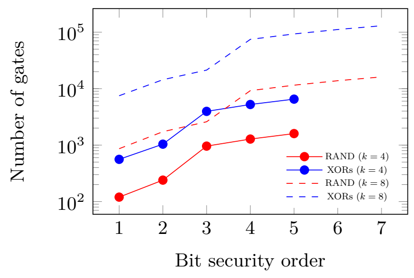


Figure 6: The number of gates and random bits required in Gadget-1 for different fields $\mathbb{F}_{2^k}$ over its provided security orders.

Table 3: Cycle count of Gadget-1 and Gadget-3 ARM Cortex-M4 implementation compared to the ISW multiplication in $\mathbb{F}_{2^4}$ for a similar bit-probing security order. Note that, following Table 1 for $\mathbb{F}_{2^4}$, inner product masking achieves second-order security for $n = 2$ and fifth-order security for $n = 3$.

Bit security order			1	2	3	4	5	6
Cycle count	ISW	log-a-log	663	1472	2618	4101	5921	8078
		Schoolbook	1803	4037	7178	11226	16181	22043
		Gadget-3	2387	2387	5408	5408	5408	-
		Gadget-1	8973	8973	32803	32803	32803	-

6 Practical Side-Channel Leakage Evaluation

In the previous sections, we showed the theoretical security of Gadget-1 (*e.g.*, the secure randomness generation) and Gadget-3 and proved them to be t-SNI secure in the bit-probing model. In this section, we complement this analysis with a practical side-channel leakage evaluation on a real-world device and confirm these theoretical results.

In order to reduce the number of sample points in the traces, our evaluation targets an ARM assembly implementation tailored specifically for two-share inner product maskings over a field $\mathbb{F}_{2^4} \equiv \mathbb{F}_2[x]/x^4 + x + 1$. In an attempt to avoid unexpected interactions between the shares, we place each binary variable in a separate register. Based on this implementation, we evaluate the multiplication gadget with inner product maskings encoded from different values for the public vector L ranging from best to worst bit probing security and show the resulting practical security impact.

6.1 Measurement Setup

Measurement tests are conducted on a CW308T-STM32F target board which embeds a STM32F415RG Cortex-M4 microcontroller running at an internal 24 MHz operating frequency. For trace acquisition, we used a NewAE CW308 UFO board and a Tektronix DPO70404C oscilloscope with a sample rate of 125MS/s. In addition, an external 8 MHz clock frequency was used for the Cortex-M4 core. All our measurements are synchronized by aligning the oscilloscope with the external clock. Finally, the Arm GNU toolchain[1] is used for compiling the Cortex-M4 assembly implementation.

[1]gcc-arm-11.2-2022.02-x86_64-arm-none-eabi

6.2 Leakage Assessment

The side-channel security evaluation focuses on leakage detection via the widely used Test Vector Leakage Assessment (TVLA) [GJJR11] methodology using a non-specific, fixed vs. random t-test statistic. This procedure provides metrics of potential leakage independently of a specific attack scenario. In such a procedure, traces are first collected from the target device operating on a fixed or a random input plaintext and placed in two respective sets namely S_{fixed} and S_{random}. In order to avoid unexpected influences, traces from the two sets are captured in a random fashion. Next, the Welch's (two-tailed) t-test is computed. This test is computed on every sample point to determine the validity of the null hypothesis, which is that the samples in both sets were drawn from the same population (*i.e.*, the mean of the two sets are similar). From a side-channel analysis viewpoint, the indistinguishability of the sets means that the masking is secure. In the literature, to determine if the null hypothesis is rejected (*i.e.*, that the masked implementation leaks) the threshold value is usually defined at $t = \pm 4.5$ (which provides a confidence of roughly 0.99999). This value is, however, not universal and could be adapted to a higher value (see [DZD+17, Table 1], [BGG+14, Appendix A]) as long traces will, with high probability, exceed the ± 4.5 threshold due to false positives.

Since software implementations split the computation over multiple clock cycles, higher-order evaluation of a software implementation needs to be performed in a multivariate setting. In this setup, the general methodology consists of preprocessing each trace using a combination function and then conducting a first-order attack on the preprocessed traces. As summarized in Table 1, a two-share inner product masking defined over $\mathbb{F}_{2^4}$ with optimal parameters is expected to resist a second-order attack. Hence, we need a bivariate t-test for which we use the SCALib [CB23] library which utilizes the optimal combination function (*i.e.*, the centered product [PRB09]) to merge each trace at two specific time points. However, this approach significantly increases computational complexity, making it impractical for traces with many sample points. Therefore, we downsample, keeping only a subset of points before preprocessing. Downsampling carries the risk of missing significant leakages and the test may provide a false sense of security. Hence, the downsampling value needs to be carefully selected. With the secure $IPM_{free}(0)$ randomness generation taken in account, our assembly implementation results in a trace of 60000 sample points from a sampling rate of 125MS/s, and knowing that 6808 assembly instructions are used, we obtain that the number of sample points per assembly instruction is approximately equal to 8.8. Thus, a downsampling of at most 8.8 would not skip any point of interest. In practice, we chose to retain one every five sample points from the original traces resulting in a reasonable 12000 total sample points. While this allows for a bivariate analysis to be evaluated in a reasonable time, it still results in a large number of individual t-tests which requires adjusting the threshold t_h value utilized in the bivariate analysis. For this, we follow the methodology by Ding *et al.* [DZD+17] and obtain a $t_h = 6.416$ for $12000^2/2$ sample points for an $\alpha = 0.01$.

6.3 Results

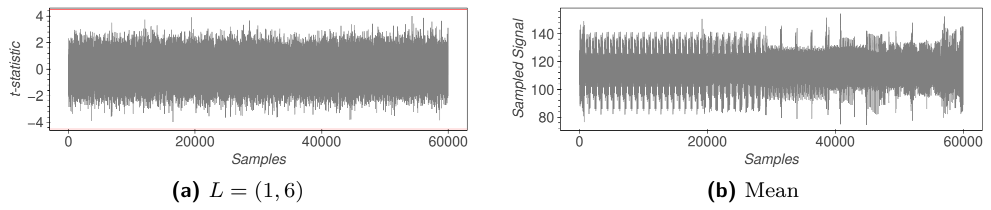
First, we discuss the univariate first-order t-test result for one million measurements shown in Figure 7a. Two-share software masked implementations based on Boolean masking usually struggle to guarantee their first-order security due to unintended interactions between values in the microcontroller [BGG+14, PV17, MPW22, GHP+21]. Transition leakage is a typical leakage source where an overwrite of one share (*e.g.*, $x_1 = r$) in a register (or a memory cell) currently holding another share (*e.g.*, $x_0 = x \oplus r$) may reveal the secret information ($HD(x \oplus r, r) = HW(x)$). As a result, two-share implementations require additional fixes to guarantee first-order security [GHP+21, GD23, SSB+21]. However, first-order implementations based on inner product masking benefit from a natural protection against transition leakage. Assuming $L_1 \neq 1$, the Hamming distance between two inner product shares is still uniformly distributed [BFG+17, Section 6.3].

Next, we discuss the second-order bivariate t-test results. Figure 8d shows the evaluation result of both Gadget-3 and Gadget-1 (*e.g.*, the secure randomness $IPM_{free}(0)$ generation) with the optimal value $L = (1, 6)$ for which $t_b = 2$. Figure 10a shows the evolution of the maximum peak value with an increasing number of traces evaluated up to one million. These results confirm our theoretical expectations from Section 4 as no significant evidence of leakage was detected for one million measurements and the maximum threshold value does not show any increase through the evaluation. Figure 8c displays the evaluation of the same parameters but with the randomness deactivated (*i.e.*, the masking is deactivated).

Lastly, we display the results of the evaluation of Gadget-1 with weaker parameters for which $t_b = 1$ namely: $L_1 = 2$ and $L_1 = 1$ (*i.e.*, Boolean masking) in Figure 8b and Figure 8a, respectively. As expected, we observe second-order leakage in both evaluations. However, for $L_1 = 2$, the results in Figure 8b show less evidence of leakage. Similarly to the optimal parameter configuration where $L = (1, 6)$, we show the evolution of the maximum threshold values for $L_1 = 2$ and $L_1 = 1$ in Figure 10b where we observe increasing values as more traces are evaluated. For the sake of completeness, we evaluate the existing code-based multiplication gadget introduced by Wang *et al.* [WMCS20]. As observed in Wu *et al.*'s practical analysis [WCG+22], due to an internal switch from a code-based to an additive sharing and a security based on word-level probing (instead of bit-level) the multiplication gadgets fails to maintain the security order amplification effect during its computation. This is shown in Figure 9 where a second-order secure inner product masking multiplication breaks in a bivariate analysis.

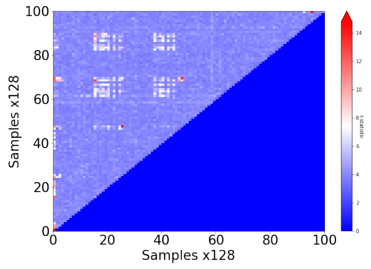
6.4 Evaluation Tools Limitations

In an attempt to reinforce theoretical and practical results, two tools were considered. To complement the SNI security proof in Section 4, the utilization of the formal verification tool MaskVerif [BBFG18] was attempted. Unfortunately, MaskVerif is not suitable for this task. MaskVerif does not support custom encoding functions as it is designed to work exclusively with input and output shares specifically constructed using Boolean masking. Consequently, one would need to transform the Boolean input shares into inner product masking shares, which is not feasible with this tool. While Boolean masking can be trivially constructed from inner product masking (e.g., by defining $L_i = 1$), the process of mixing multiple bits of randomness to each sensitive bit —from Boolean masking input shares— cannot be easily accomplished. Next, to guarantee the first and second probing security and to accompany the practical side-channel leakage evaluation, the leakage simulator PROLEAD_SW [ZMM23] was used. While first order security was easily verified on the assembly implementation, the bivariate analysis requires an excessive amount or RAM (e.g., more than 1TB) which prevented the completion of the evaluation.

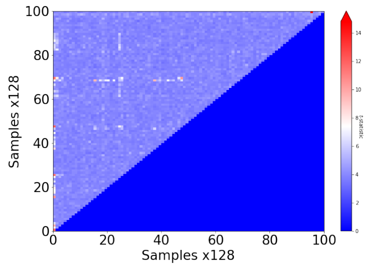


(a) $L = (1, 6)$ **(b)** Mean

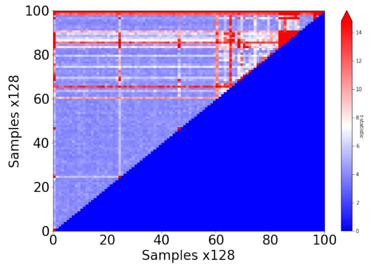
Figure 7: First-order t-test *non-specific*, *fixed vs. random* t-test results of the ARM assembly multiplication Gadget-3 and randomness generation Gadget-1. The ±4.5 threshold is marked by red lines. Experiments are done using 1M traces.



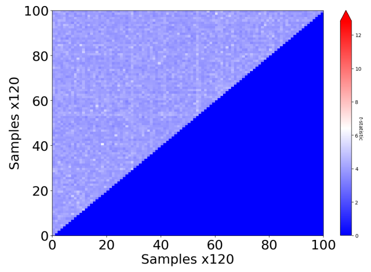
(a) $L = (1, 1)$



(b) $L = (1, 2)$



(c) $L = (1, 6)$ RNG-off



(d) $L = (1, 6)$

Figure 8: Second-order bivariate *non-specific*, *fixed vs. random* t-test results of the ARM assembly multiplication. The experiments with $L = (1, 6)$ use 1M traces evaluate both Gadget-1 and Gadget-3, the other tests use 100k traces on Gadget-1. Figures 8a-8c are expected to leak and Figure 8d is expected to be secure.

7 Conclusion and Future Work

In this work, we detailed a new secure multiplication gadget tailored for inner product masking which preserves the security order amplification (*i.e.*, bit-probing security order) during its computation. Namely, we designed a gadget where the bit-level probing security is preserved throughout the computation by leveraging inner product maskings of zero as randomness distributed to every binary cross product. We showed that for a similar security order —thanks to the security order amplification effect— our multiplier exhibits a lower number of gates than the standard ISW multiplication gadget for Boolean masking. In addition, we compared the practical efficiency of our multiplication gadget against a Boolean masked gadget. We find that the preprocessing phase is much heavier compared to Boolean masking. However, the gadget's online phase shows promising results which could improve over Boolean masking in certain security orders. Finally, we conducted

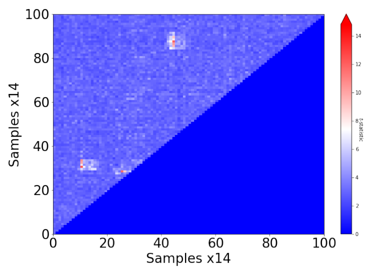
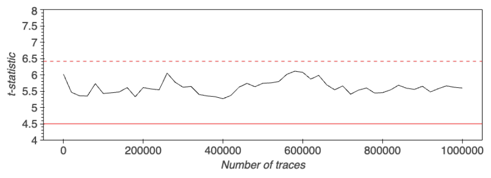
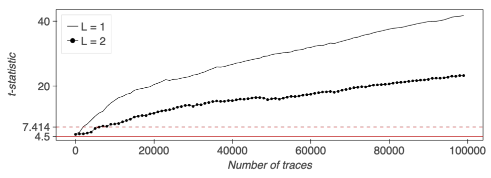


Figure 9: Second-order bivariate *non-specific*, *fixed vs. random* t-test results of the ARM assembly `CodeL` and `CodeMuL` gadgets from [WCG+22] with $L = (1, 91)$.



(a) $L = (1, 6)$



(b) $L = (1, 1)$ and $L = (1, 2)$

Figure 10: Evolution of the maximum value for the second-order bivariate t-test of the ARM implementation with increasing number of traces. The red line is the 4.5 threshold and the dashed red line represents the adjusted threshold 6.416 obtained in Section 6.2. The experiment with $L_1 = 6$ is done with 1M traces and the others with 100k traces.

practical side-channel leakage evaluation on an ARM assembly implementation where no significant evidence of leakage was detected for one million measurements. While we took significant steps forward in inner product masking, we list some potential future works.

Reduced randomness Given the significant number of random bits required by our multiplier, we believe it is possible to optimize the gadget for general parameters (meaning, any field, any parameter L, and any number of shares) with respect to its randomness requirement. In the security proof in Section 4, we note that the simulator is given no additional input shares for categories (4), (5), and (6). This indicates that some of the randomness could be removed while preserving the bit-level probing security. In addition, we rely on secure generation of $IPM_{free}(0)$ sharings which comes at a high cost. Immediately integrating random bits into Gadget 2 and 3 would reduce the overhead.

Implementation efficiency We did not place efficiency as a primary priority, opting to leave potential improvements for future work. Our multiplication gadget may benefit from bit-slice or vectorized techniques that we leave as future work.

Extension to code-based masking We detailed a multiplication gadget which preserves the bit-probing security level of its masking even in a composable secure setting, however, the function of the gadget could be generalized to maintain the bit-probing security level of any code-based masking scheme. To be clear, the current multiplication gadget is not correct for an arbitrary code-based masking since it first calculates the $\mathbb{F}_{2^k}$ multiplication of the shares $x_i y_j$ before multiplying the cross products with the L matrix. For the correctness of Algorithm 3, we need the property that $x_i(Ly_j) = L(x_i y_j)$ which does not hold when L is an arbitrary bit matrix. Instead, for a code-based masking gadget, one needs to first securely calculate the multiplication of a share with the matrix L before calculating cross products. We leave this interesting generalization as future work.

References

[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 Edgar R. Weippl, Stefan Katzenbeisser, Christopher Kruegel, Andrew C. Myers, and Shai Halevi, editors, *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, Vienna, Austria, October 24-28, 2016*, pages 116–129. ACM, 2016.

[BBFG18] Gilles Barthe, Sonia Belaïd, Pierre-Alain Fouque, and Benjamin Grégoire. maskverif: a formal tool for analyzing software and hardware masked implementations. *IACR Cryptol. ePrint Arch.*, page 562, 2018.

[BCC$^+$14] Julien Bringer, Claude Carlet, Hervé Chabanne, Sylvain Guilley, and Houssem Maghrebi. Orthogonal direct sum masking - A smartcard friendly computation paradigm in a code, with builtin protection against side-channel and fault attacks. In David Naccache and Damien Sauveron, editors, *Information Security Theory and Practice. Securing the Internet of Things - 8th IFIP WG 11.2 International Workshop, WISTP 2014, Heraklion, Crete, Greece, June 30 - July 2, 2014. Proceedings*, volume 8501 of *Lecture Notes in Computer Science*, pages 40–56. Springer, 2014.

[BCRT23] Sonia Belaïd, Gaëtan Cassiers, Matthieu Rivain, and Abdul Rahman Taleb. Unifying freedom and separation for tight probing-secure composition. In Helena Handschuh and Anna Lysyanskaya, editors, *Advances in Cryptology - CRYPTO 2023 - 43rd Annual International Cryptology Conference, CRYPTO 2023, Santa Barbara, CA, USA, August 20-24, 2023, Proceedings, Part III*, volume 14083 of *Lecture Notes in Computer Science*, pages 440–472. Springer, 2023.

[BFG15] Josep Balasch, Sebastian Faust, and Benedikt Gierlichs. Inner product masking revisited. In Elisabeth Oswald and Marc Fischlin, editors, *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*, volume 9056 of *Lecture Notes in Computer Science*, pages 486–510. Springer, 2015.

[BFG$^+$17] Josep Balasch, Sebastian Faust, Benedikt Gierlichs, Clara Paglialonga, and François-Xavier Standaert. Consolidating inner product masking. In Tsuyoshi Takagi and Thomas Peyrin, editors, *Advances in Cryptology - ASIACRYPT 2017 - 23rd International Conference on the Theory and Applications of Cryptology and Information Security, Hong Kong, China, December 3-7, 2017, Proceedings, Part I*, volume 10624 of *Lecture Notes in Computer Science*, pages 724–754. Springer, 2017.

[BGG$^+$14] Josep Balasch, Benedikt Gierlichs, Vincent Grosso, Oscar Reparaz, and François-Xavier Standaert. On the cost of lazy engineering for masked software implementations. In Marc Joye and Amir Moradi, editors, *Smart Card Research and Advanced Applications - 13th International Conference, CARDIS 2014, Paris, France, November 5-7, 2014. Revised Selected Papers*, volume 8968 of *Lecture Notes in Computer Science*, pages 64–81. Springer, 2014.

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

[CGC$^+$21] Wei Cheng, Sylvain Guilley, Claude Carlet, Sihem Mesnager, and Jean-Luc Danger. Optimizing inner product masking scheme by a coding theory approach. *IEEE Trans. Inf. Forensics Secur.*, 16:220–235, 2021.

[CJRR99] Suresh Chari, Charanjit S. Jutla, Josyula R. Rao, and Pankaj Rohatgi. Towards sound approaches to counteract power-analysis attacks. In Michael J. Wiener, editor, *Advances in Cryptology - CRYPTO '99, 19th Annual International Cryptology Conference, Santa Barbara, California, USA, August 15-19, 1999, Proceedings*, volume 1666 of *Lecture Notes in Computer Science*, pages 398–412. Springer, 1999.

[DZD+17] A. Adam Ding, Liwei Zhang, François Durvaux, François-Xavier Standaert, and Yunsi Fei. Towards sound and optimal leakage detection procedure. In Thomas Eisenbarth and Yannick Teglia, editors, *Smart Card Research and Advanced Applications - 16th International Conference, CARDIS 2017, Lugano, Switzerland, November 13-15, 2017, Revised Selected Papers*, volume 10728 of *Lecture Notes in Computer Science*, pages 105–122. Springer, 2017.

[GD23] John Gaspoz and Siemen Dhooghe. Threshold implementations in software: Micro-architectural leakages in algorithms. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2023(2):155–179, 2023.

[GHP+21] Barbara Gigerl, Vedad Hadzic, Robert Primas, Stefan Mangard, and Roderick Bloem. Coco: Co-design and co-verification of masked software implementations on cpus. In Michael Bailey and Rachel Greenstadt, editors, *30th USENIX Security Symposium, USENIX Security 2021, August 11-13, 2021*, pages 1469–1468. USENIX Association, 2021.

[GJJR11] Gilbert Goodwill, Benjamin Jun, Joshua Jaffe, and Pankaj Rohatgi. A testing methodology for side channel resistance, 2011. https://csrc.nist.gov/csrc/media/events/non-invasive-attack-testing-workshop/documents/08_goodwill.pdf. Retrieved on March 31th, 2022.

[GM11] Louis Goubin and Ange Martinelli. Protecting AES with shamir's secret sharing scheme. In Bart Preneel and Tsuyoshi Takagi, editors, *Cryptographic Hardware and Embedded Systems - CHES 2011 - 13th International Workshop, Nara, Japan, September 28 - October 1, 2011. Proceedings*, volume 6917 of *Lecture Notes in Computer Science*, pages 79–94. Springer, 2011.

[Gol02] Jovan Dj. Golic. Multiplicative masking and power analysis of AES. *IACR Cryptol. ePrint Arch.*, page 91, 2002.

[GP99] Louis Goubin and Jacques Patarin. DES and differential power analysis (the "duplication" method). In Çetin Kaya Koç and Christof Paar, editors, *Cryptographic Hardware and Embedded Systems, First International Workshop, CHES'99, Worcester, MA, USA, August 12-13, 1999, Proceedings*, volume 1717 of *Lecture Notes in Computer Science*, pages 158–172. Springer, 1999.

[IKL+13] Yuval Ishai, Eyal Kushilevitz, Xin Li, Rafail Ostrovsky, Manoj Prabhakaran, Amit Sahai, and David Zuckerman. Robust pseudorandom generators. In *ICALP (1)*, volume 7965 of *Lecture Notes in Computer Science*, pages 576–588. Springer, 2013.

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

[KJJ99] Paul C. Kocher, Joshua Jaffe, and Benjamin Jun. Differential power analysis. In Michael J. Wiener, editor, *Advances in Cryptology - CRYPTO '99, 19th Annual International Cryptology Conference, Santa Barbara, California, USA, August 15-19, 1999, Proceedings*, volume 1666 of *Lecture Notes in Computer Science*, pages 388–397. Springer, 1999.

[MPW22] Ben Marshall, Dan Page, and James Webb. MIRACLE: micro-architectural leakage evaluation A study of micro-architectural power leakage across many devices. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2022(1):175–220, 2022.

[PGS$^+$17] Romain Poussier, Qian Guo, François-Xavier Standaert, Claude Carlet, and Sylvain Guilley. Connecting and improving direct sum masking and inner product masking. In Thomas Eisenbarth and Yannick Teglia, editors, *Smart Card Research and Advanced Applications - 16th International Conference, CARDIS 2017, Lugano, Switzerland, November 13-15, 2017, Revised Selected Papers*, volume 10728 of *Lecture Notes in Computer Science*, pages 123–141. Springer, 2017.

[PRB09] Emmanuel Prouff, Matthieu Rivain, and Régis Bevan. Statistical analysis of second order differential power analysis. *IEEE Trans. Computers*, 58(6):799–811, 2009.

[PV17] Kostas Papagiannopoulos and Nikita Veshchikov. Mind the gap: Towards secure 1st-order masking in software. In Sylvain Guilley, editor, *Constructive Side-Channel Analysis and Secure Design - 8th International Workshop, COSADE 2017, Paris, France, April 13-14, 2017, Revised Selected Papers*, volume 10348 of *Lecture Notes in Computer Science*, pages 282–297. Springer, 2017.

[RH04] Arash Reyhani-Masoleh and M. Anwar Hasan. Low complexity bit parallel architectures for polynomial basis multiplication over gf(2^{m}). *IEEE Trans. Computers*, 53(8):945–959, 2004.

[SSB$^+$21] Madura A. Shelton, Niels Samwel, Lejla Batina, Francesco Regazzoni, Markus Wagner, and Yuval Yarom. Rosita: Towards automatic elimination of power-analysis leakage in ciphers. In *28th Annual Network and Distributed System Security Symposium, NDSS 2021, virtually, February 21-25, 2021.* The Internet Society, 2021.

[WCG$^+$22] Qianmei Wu, Wei Cheng, Sylvain Guilley, Fan Zhang, and Wei Fu. On efficient and secure code-based masking: A pragmatic evaluation. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2022(3):192–222, 2022.

[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.

[WYS19] Weijia Wang, Yu Yu, and François-Xavier Standaert. Provable order amplification for code-based masking: How to avoid non-linear leakages due to masked operations. *IEEE Trans. Inf. Forensics Secur.*, 14(11):3069–3082, 2019.

[ZMM23] Jannik Zeitschner, Nicolai Müller, and Amir Moradi. Prolead_sw - probing-based software leakage detection for ARM binaries. *IACR Cryptol. ePrint Arch.*, page 34, 2023.