Fast verification of masking schemes in characteristic two

Nicolas Bordes and Pierre Karpman

Université Grenoble Alpes, France

{nicolas.bordes,pierre.karpman}@univ-grenoble-alpes.fr

Abstract. We revisit the matrix model for non-interference (NI) probing security of masking gadgets introduced by Belaïd *et al.* at CRYPTO 2017. This leads to two main results.
1) We generalise the theorems on which this model is based, so as to be able to apply them to masking schemes over any finite field — in particular $\mathbb{F}_2$ — and to be able to analyse the *strong* non-interference (SNI) security notion. We also follow Faust *et al.* (TCHES 2018) to additionally consider a *robust* probing model that takes hardware defects such as glitches into account.
2) We exploit this improved model to implement a very efficient verification algorithm that improves the performance of state-of-the-art software by three orders of magnitude. We show applications to variants of NI and SNI multiplication gadgets from Barthe *et al.* (EUROCRYPT 2017) which we verify to be secure up to order 11 after a significant parallel computation effort, whereas the previous largest proven order was 7; SNI refreshing gadgets (*ibid.*); and NI multiplication gadgets from Groß *et al.* (TIS@CCS 2016) secure in presence of glitches. We also reduce the randomness cost of some existing gadgets, notably for the implementation-friendly case of 8 shares, improving here the previous best results by 17% (resp. 19%) for SNI multiplication (resp. refreshing).

Keywords: High-order masking, probing model, multiplication gadget, refreshing gadget, linear code.

1 Introduction

Since their introduction in the late last century, side-channel attacks and in particular *Differential Power Analysis* (DPA) [KJJ99] have developed into one of the most efficient attack techniques on implementations of cryptographic primitives. The importance of this new threat and its practical relevance soon lead to the design of appropriate counter-measures, one of the most influential to date being the "ISW" private multiplication circuit of Ishai, Sahai and Wagner [ISW03]. This is a foremost example of a *masking scheme*, where sensitive data are split into several shares using a secret sharing scheme; the crux of the design is then to devise a way to perform field arithmetic over the shares without leaking too much information to the adversary in the process.

A major characteristic of a masking scheme is the *order* at which it is secure: in a probing model such as the one introduced by Ishai, Sahai and Wagner, a circuit secure at order d is such that no adversary can learn information about its input and output even when being given d intermediate values of its computation. The usefulness of increasing the security order is then justified by the fact that under reasonable assumptions, the number of measurements needed for a successful attack increases exponentially in d [DFS15].

Unfortunately, high-order schemes also come with a significant overhead, since the complexity of ISW multiplication is quadratic in d for three relevant metrics: to secure one field multiplication, one needs $2d(d+1)$ sums, $(d+1)^2$ products and $d(d+1)/2$ fresh random masks. This lead to several attempts to find more efficient multiplication circuits, especially with respect to the last two metrics.

A number of new schemes for private multiplication were introduced in the past few years by Belaïd *et al.* [BBP+16,BBP+17]. At EUROCRYPT 2016, they design a new high-order scheme whose randomness complexity is decreased to $\approx d^2/4 + d$, and which can be easily instantiated over any finite field of characteristic two (they also give specific schemes with even lower cost up to order 4). The security of this multiplication is analysed in the composable model of *non interference* (NI) from Barthe *et al.* [BBD+16]. This is slightly weaker than the *strong non-interference* (SNI) security achieved by ISW multiplication but remains of high practical relevance: for instance, one can replace half of the multiplications in a masked AES S-box computation by the ones of [BBP+16] while maintaining the overall strong SNI security for the entire S-box. At CRYPTO 2017, the same authors propose two new schemes, one with linear *bilinear multiplication* complexity, and the other with linear randomness complexity. However, those are complex to securely instantiate and cannot be done so over $\mathbb{F}_2$. As an example, over $\mathbb{F}_{2^8}$, Belaïd *et al.* only manage to instantiate their algorithms at order 2 and 3 respectively; this was later slightly improved to 4 in both cases by Karpman and Roche [KR18]. In this second paper, Belaïd *et al.* also analyse the security of their schemes thanks to a powerful matrix-based model that they introduce. This model is however not complete for schemes defined over small fields such as $\mathbb{F}_2$; while this was not a limitation for their schemes, it precludes its full application to this common case. Finally, Barthe *et al.* introduced some of the most efficient known NI and SNI multiplication and refreshing schemes at EUROCRYPT 2017 [BDF+17], selected instances of which were then later improved by Grégoire *et al.* [GPSS18] and Barthe *et al.* [BBD+18].

The above work are chiefly concerned with software-oriented counter-measures, and the protection of hardware circuits additionally requires to take into account the possibility of physical defects such as glitches. This can for instance be done by generalising probing security to a *robust* variant proposed by Faust *et al.* [FGP+18], or by following the more physical approach of Bloem *et al.* [BGI+18]. As was recently noted by Moos *et al.* [MMSS19], the analysis of masking schemes in this harder model is currently quite less mature than in the software case.

On the implementation side, several recent work investigate the efficiency of high-order masking in practice [GR17,JS17,GJRS18,GPSS18]; they show in particular the increasing feasibility of masking block ciphers at quite high order such as 7, and the possibility of masking at very high order such as 31. Such high-order masking may be useful to secure implementations running on devices with low noise level. This was recently highlighted by a practical attack of Bronchain and Standaert on a protected AES implementation where the low noise and masking order were found to be contributing factors to its feasibility [BS19]. From a technical point-of-view, high-order implementations share the common approach of exploiting bitslicing or vectorisation to amortise the overhead brought by the use of many shares. They also confirm the high cost of randomness generation; for instance, depending on the random number generator performance and the block cipher under consideration Journault and Standaert report that 68–92% of the time is spent generating fresh masks in their 32-share implementations [JS17]. Also, since bitslicing works with operations at the bit level, this strategy requires the masking to be performed over $\mathbb{F}_2$ and these work confirm the importance of high-order masking schemes over this field with low randomness complexity.

1.1 Our contribution

Our work brings two main contributions. On the theoretical side, we extend the matrix model of [BBP+17] to be able to prove the security of schemes defined over any finite field, and $\mathbb{F}_2$ in particular; we also extend it to analyse SNI security, whereas it was only formulated in the NI case by Belaïd *et al.*, and incorporate the robust probing model of Faust *et al.* [FGP+18] to offer limited support for verification in presence of glitches. The extension to $\mathbb{F}_2$ is particularly relevant to concrete masking schemes since up to a few exceptions such as the one of [BBP+17], most schemes are intrinsically defined over this field. A corollary of our new theorems is also a simple proof that a scheme proven secure over $\mathbb{F}_2$ remains so when used over any extension, which is a common practice.

On the practical side, we use this extended model to derive a very efficient implementation of a verification algorithm whose performance beats the state-of-the-art maskVerif tool of Barthe *et al.* [BBC+19] by three orders of magnitude in the case of software multiplication gadgets; we illustrate this on software and hardware multiplication and refreshing schemes from the literature. We then take advantage of our improved verification performance and spend significant computation effort into proving the security of (variants of) the software multiplication gadgets of Barthe *et al.* [BDF+17] at mid-to-high order. This is all the more relevant since those do not have known generic proof of security at any order and are used in concrete implementations [JS17,GPSS18]. We verify NI and SNI gadgets up to order 11 at a total combined cost of close to 2^{55} basic operations, whereas the previously largest proven order was 7. We justify on the way the necessity of performing this kind of verification for schemes that do not have generic proofs by disproving a conjecture of Barthe *et al.* on the security of a natural transformation of NI schemes into SNI ones. Finally, we propose various improvements to decrease the randomness cost of several software gadgets. This results for instance in a decrease of 17% (resp. 19%) over the state-of-the-art for 8-share SNI multiplication (resp. refreshing) schemes, which could then be used as stand-in replacements in the vectorised implementation of Grégoire *et al.* [GPSS18].

1.2 Roadmap

We present the security models and extend the matrix approach from CRYPTO 2017 in Section 2. We then introduce our verification algorithm and discuss its implementation in Sections 3 and 4. We conclude with experimental results and the description of new gadgets in Section 5.

1.3 Notation

We use $\mathbb{K}^{n\times m}$ to denote the ring of matrices of n rows and m columns over the field $\mathbb{K}$. We write $[\![a, a+t]\!]$ for the set of integers $\{a, a+1, \ldots, a+t\}$. Matrices and vectors are named with bold upper- and lower-case variables respectively; $\boldsymbol{I}_n$, $\boldsymbol{0}_{n\times m}$, $\boldsymbol{1}_{n\times m}$ always denote the n-dimensional identity matrix and all-zero and all-one $n \times m$ matrices respectively, over any field $\mathbb{K}$.

2 Security models for masking schemes

2.1 Simulatability and non-interference

We start by recalling the definitions of the models of non-interference (NI), tight non-interference (TNI) and strong non-interference (SNI), introduced by Barthe *et al.* at CCS 2016 [BBD+16]. Our presentation closely follows the one of Belaïd *et al.* [BBP+17].

Definition 1 (Gadgets). *Let $f : \mathbb{K}^n \to \mathbb{K}^m$, u, $v \in \mathbb{N}$; a (u,v)-gadget for the function f is a randomised circuit C such that for every tuple $(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n) \in (\mathbb{K}^u)^n$ and every set of random coins $\mathcal{R}$, $(\boldsymbol{y}_1, \ldots, \boldsymbol{y}_m) \hookleftarrow C(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n; \mathcal{R})$ satisfies:*

$$\left(\sum_{j=1}^{v} \boldsymbol{y}_{1,j}, \ldots, \sum_{j=1}^{v} \boldsymbol{y}_{m,j}\right) = f\left(\sum_{j=1}^{u} \boldsymbol{x}_{1,j}, \ldots, \sum_{j=1}^{u} \boldsymbol{x}_{m,j}\right).$$

One further defines $\boldsymbol{x}_i$ as $\sum_{j=1}^{u} \boldsymbol{x}_{i,j}$, and similarly for $\boldsymbol{y}_i$; $\boldsymbol{x}_{i,j}$ is called the jth share *of $\boldsymbol{x}_i$.*

In this definition, a randomised circuit C is a directed acyclic graph whose vertices represent arithmetic operation *gates* (addition and multiplication) over $\mathbb{K}$ of arity two, or random gates of arity zero whose outputs are i.i.d. over $\mathbb{K}$ for every execution of the circuit, and recorded in the variable $\mathcal{R}$; the edges of the graph are *wires* that connect the input and output of the gates together so as to describe the full computation of a given function.

A *probe* on a circuit C is a map that for every execution $C(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n; \mathcal{R})$ returns the value propagated on one of the wires of C. One may further distinguish between *external* probes on the output wires or output shares $\boldsymbol{y}_{i,j}$'s of C, and the remaining *internal* probes.

Definition 2 (t-Simulatability). *Let C be a (u,v)-gadget for $f : \mathbb{K}^n \to \mathbb{K}^n$, and ℓ, $t \in \mathbb{N}$. A set $\mathcal{P} = \{p_1, \ldots, p_\ell\}$ of probes of C is said to be t-*simulatable *if $\exists\, I_1, \ldots, I_n \subseteq [\![1, u]\!]$; $\#I_i \leq t$ and a randomised function $\pi : (\mathbb{K}^t)^n \to \mathbb{K}^\ell$ such that for any fixed $(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n) \in (\mathbb{K}^u)^n$, $\{p_1, \ldots, p_\ell\} \sim \{\pi(\{\boldsymbol{x}_{1,i},\, i \in I_1\}, \ldots, \{\boldsymbol{x}_{n,i},\, i \in I_n\})\}$.*

Less formally, a set $\mathcal{P}$ of probes on C is t-simulatable if there exists a randomised function that perfectly simulates the distribution of $\{p_1, \ldots, p_\ell\}$ while requiring at most t shares of every input to C. It is important to remark here that a simulation has to be done for every *fixed* input $(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n)$, regardless of the fact that one may randomise these inputs across many executions of C.

Thanks to Definition 2, we may now define the following.

Definition 3 (d-Non-interference). *A (u,v)-gadget C for a function over $\mathbb{K}^n$ is d-*non-interfering *(or d-NI) if and only if for any set $\mathcal{P}$ of at most d probes on C $\exists\, t \leq d$ s.t. $\mathcal{P}$ is t-simulatable.*

Definition 4 (d-Tight non-interference). *A (u,v)-gadget C for a function over $\mathbb{K}^n$ is d-*tight-non-interfering *(or d-TNI) if and only if any set of $t \leq d$ probes on C is t-simulatable.*

Definition 5 (d-Strong non-interference). *A (u,v)-gadget C for a function over $\mathbb{K}^n$ is d-*strong non-interfering *(or d-SNI) if and only if for every set $\mathcal{P}_1$ of d_1 internal probes and every set $\mathcal{P}_2$ of d_2 external probes such that $d_1 + d_2 \leq d$, then $\mathcal{P}_1 \cup \mathcal{P}_2$ is d_1-simulatable.*

It is clear that strong non-interference implies tight non-interference at the same order, which itself implies non-interference. Barthe *et al.* [BBD+16] showed that tight non-interference did not imply strong non-interference, but that the composition of a d-NI gadget with a d-SNI one is d-SNI, while the composition of two d-NI gadgets was not necessarily d-NI. On the other hand they also showed that non-interference and tight non-interference are in fact equivalent, which in proofs allows to select the most convenient notion.

2.2 Matrix model for non-interference

We now recall Theorem 3.5 from Belaïd *et al.* [BBP+17], which defines a powerful matrix model to analyze the (T)NI property of a gadget over a sufficiently large field $\mathbb{K}$ for which all probes are *bilinear*. We then generalise it as Theorem 12 to work with schemes over any finite field (and $\mathbb{F}_2$ in particular), and to also analyse SNI security in Theorem 20.

In all of the following, we restrict our interest to gadgets for binary functions[1] $f : \mathbb{K}^2 \to \mathbb{K}$, and the inputs to f (resp. their sharings in a gadget C) will be denoted a and b (resp. $\boldsymbol{a} = (\boldsymbol{a}_0, \ldots, \boldsymbol{a}_{u-1})^t$, $\boldsymbol{b} = (\boldsymbol{b}_0, \ldots, \boldsymbol{b}_{u-1})^t$). We also write the elements of $\mathcal{R}$ as a vector $\boldsymbol{r} = (\boldsymbol{r}_1, \ldots, \boldsymbol{r}_R)^t$

Definition 6 (Bilinear probe). *A probe p on a $(d+1, v)$-gadget C for a function $f : \mathbb{K}^2 \to \mathbb{K}$ is called* bilinear *iff. it is an affine function in $\boldsymbol{a}_i$, $\boldsymbol{b}_j$, $\boldsymbol{a}_i\boldsymbol{b}_j$, $\boldsymbol{r}_k$; $0 \leq i, j \leq d$, $1 \leq k \leq R$. Equivalently, p is bilinear iff. $\exists \boldsymbol{M} \in \mathbb{K}^{(d+1)\times(d+1)}$, $\boldsymbol{\mu}$, $\boldsymbol{\nu} \in \mathbb{K}^{d+1}$, $\boldsymbol{\sigma} \in \mathbb{K}^R$ and $\tau \in \mathbb{K}$ s.t. $p = \boldsymbol{a}^t\boldsymbol{M}\boldsymbol{b} + \boldsymbol{a}^t\boldsymbol{\mu} + \boldsymbol{b}^t\boldsymbol{\nu} + \boldsymbol{r}^t\boldsymbol{\sigma} + \tau$.*

Definition 7 (Functional dependence). *An expression* $\mathrm{E}(x_1, \ldots, x_n)$ *is said to* functionally depend *on x_n iff. $\exists c_1, \ldots, c_{n-1}$ s.t. the mapping $x_n \mapsto \mathrm{E}(c_1, \ldots, c_{n-1}, x_n)$ is not constant.*

We now introduce the following condition.

Condition 8 ([BBP+17, Condition 3.2]). *A set of bilinear probes $\mathcal{P} = \{p_1, \ldots, p_\ell\}$ on a $(d+1, v)$-gadget C for a function $f : \mathbb{K}^2 \to \mathbb{K}$ satisfies Condition 8 iff. $\exists \boldsymbol{\lambda} \in \mathbb{K}^\ell$, $\boldsymbol{M} \in \mathbb{K}^{(d+1)\times(d+1)}$, $\boldsymbol{\mu}$, $\boldsymbol{\nu} \in \mathbb{K}^{d+1}$, and $\tau \in \mathbb{K}$ s.t. $\sum_{i=1}^{\ell} \boldsymbol{\lambda}_i p_i = \boldsymbol{a}^t\boldsymbol{M}\boldsymbol{b} + \boldsymbol{a}^t\boldsymbol{\mu} + \boldsymbol{b}^t\boldsymbol{\nu} + \tau$ and all the rows of the block matrix $\begin{pmatrix}\boldsymbol{M} & \boldsymbol{\mu}\end{pmatrix}$ or all the columns of the block matrix $\begin{pmatrix}\boldsymbol{M} \\ \boldsymbol{\nu}^t\end{pmatrix}$ are non-zero.*

In other words, this condition states that there exists a linear combination of probes of $\mathcal{P}$ that does not functionally depend on any random scalar and that functionally depends on either all of the shares for a or all of the shares for b.

We are now ready to state the following theorem.

Theorem 9 ([BBP+17, Theorem 3.5]). *Let $\mathcal{P}$ be a set of bilinear probes on a $(d+1, v)$-gadget C for a function $f : \mathbb{K}^2 \to \mathbb{K}$. If $\mathcal{P}$ satisfies Condition 8, then it is not d-simulatable. Furthermore, if $\mathcal{P}$ is not d-simulatable and $\#\mathbb{K} > d+1$, then it satisfies Condition 8.*

The next immediate corollary is more useful in practice.

Corollary 10 ([BBP+17, Corollary 3.7]). *Let C be a $(d+1, v)$-gadget for a function $f : \mathbb{K}^2 \to \mathbb{K}$ for which all probes are bilinear. If C is d-NI, then there is no set of d probes on C satisfying Condition 8. Furthermore, if $\#\mathbb{K} > d+1$ and there is no set of d probes on C satisfying Condition 8, then C is d-NI.*

For the masking schemes of CRYPTO 2017 [BBP+17] the restriction $\#\mathbb{K} > d+1$ is never an issue, as they are defined over large fields; however, this condition means that one cannot directly apply Corollary 10 to prove the security of a scheme over a small field such as $\mathbb{F}_2$.

We now sketch a proof of the second statement of Theorem 9 as a preparation to extending it to any field.

Proof (Theorem 9 right to left, sketch). Let $\mathcal{P} = \{p_1, \ldots, p_\ell\}$ be a set of bilinear probes that is not d-simulatable. We call $\boldsymbol{R}$ the block matrix $\begin{pmatrix}\boldsymbol{\sigma}_1 \cdots \boldsymbol{\sigma}_\ell\end{pmatrix}$, where $\boldsymbol{\sigma}_i$ denotes as in Definition 6 the vector of random scalars on which p_i depends. Up to a permutation of its rows and columns, the reduced column echelon form $\boldsymbol{R}'$ of $\boldsymbol{R}$ is of the shape $\begin{pmatrix}\boldsymbol{I}_t & \boldsymbol{0}_{t,\ell-t} \\ \boldsymbol{N} & \boldsymbol{0}_t\end{pmatrix}$, where $t < \ell$ is the rank of $\boldsymbol{R}$ and $\boldsymbol{N}$ is arbitrary. If we now consider the formal matrix $\boldsymbol{P} = \begin{pmatrix}p_1 \cdots p_\ell\end{pmatrix}^t$ and multiply it by the change-of-basis matrix from $\boldsymbol{R}$ to $\boldsymbol{R}'$, we obtain the matrix $\boldsymbol{P}' = \begin{pmatrix}\boldsymbol{P}'_r\ \boldsymbol{P}'_d\end{pmatrix}$ where $\boldsymbol{P}'_r$ represents t linear combinations $\{p'_1, \ldots, p'_t\}$ of probes that each depend on at least one random scalar which does not appear across any of the other linear combinations, and $\boldsymbol{P}'_d$ represents $\ell - t$ linearly independent linear combinations $\mathcal{P}' = \{p'_{t+1}, \ldots, p'_\ell\}$ of probes that do not depend on any random scalar. All of the $\{p'_1, \ldots, p'_t\}$ can then be simulated by independent uniform distributions without requiring the knowledge of any share, and as $\mathcal{P}$ is not d-simulatable, $\mathcal{P}'$ cannot be d-simulatable either. W.l.o.g., this means that for every share $\boldsymbol{a}_i$, there is at least one linear combination of probe in $\mathcal{P}'$ that depends on it. In other words, the matrix $\boldsymbol{D} = \begin{pmatrix}\boldsymbol{M}'_{t+1}\ \boldsymbol{\mu}_{t+1} \cdots \boldsymbol{M}'_\ell\ \boldsymbol{\mu}_\ell\end{pmatrix}$ that records

[1] Results for unary functions can then easily be obtained by *e.g.* fixing one input.

this dependence has no zero row. We now finally want to show that there is a linear combination $\left(\boldsymbol{\lambda}_{t+1} \cdots \boldsymbol{\lambda}_\ell\right)^t$ of elements of $\mathcal{P}'$ that satisfies Condition 8. This can be done by showing that $\exists\, \boldsymbol{\Lambda} = \left(\boldsymbol{\Lambda}_{t+1} \cdots \boldsymbol{\Lambda}_\ell\right)^t$ s.t. $\boldsymbol{D\Lambda}$ has no zero row, where the $\boldsymbol{\Lambda}_i$'s are the $(d+2)\times(d+2)$ scalar matrices of multiplication by the $\boldsymbol{\lambda}_i$'s. By the Schwartz-Zippel-DeMillo-Lipton lemma this is always the case as soon as $\#\mathbb{K} > d+1$ [Sch80], and this last step is the only one that depends on $\mathbb{K}$. $\square$

We now wish to extend Theorem 9 and its corollary to any finite field $\mathbb{K}$. We do this using the TNI notion rather than NI, and so first state an appropriate straightforward adaptation of Condition 8:

Condition 11. *A set of bilinear probes $\mathcal{P} = \{p_1, \ldots, p_\ell\}$ on a $(d+1, v)$-gadget C for a function $f : \mathbb{K}^2 \to \mathbb{K}$ satisfies Condition 11 iff. $\exists\, \boldsymbol{\lambda} \in \mathbb{K}^\ell$, $\boldsymbol{M} \in \mathbb{K}^{(d+1)\times(d+1)}$, $\boldsymbol{\mu}, \boldsymbol{\nu} \in \mathbb{K}^{d+1}$, and $\tau \in \mathbb{K}$ s.t. $\sum_{i=1}^{\ell} \boldsymbol{\lambda}_i p_i = \boldsymbol{a}^t \boldsymbol{M} \boldsymbol{b} + \boldsymbol{a}^t \boldsymbol{\mu} + \boldsymbol{b}^t \boldsymbol{\nu} + \tau$ and the block matrix $\begin{pmatrix}\boldsymbol{M} & \boldsymbol{\mu}\end{pmatrix}$ (resp. the block matrix $\begin{pmatrix}\boldsymbol{M} \\ \boldsymbol{\nu}^t\end{pmatrix}$) has at least $\ell+1$ non-zero rows (resp. columns).*

In other words, Condition 11 states that the expression $\sum_{i=1}^{\ell} \boldsymbol{\lambda}_i p_i$, which involves ℓ probes, functionally depends on no random scalar and on at least $\ell+1$ shares of a or $\ell+1$ shares of b, and hence is a TNI attack. We will then show the following:

Theorem 12. *Let $\mathcal{P}$ be a set of at most d bilinear probes on a $(d+1, v)$-gadget C for a function $f : \mathbb{K}^2 \to \mathbb{K}$. If $\mathcal{P}$, is not d-simulatable then $\exists\, \mathcal{P}' \subseteq \mathcal{P}$ s.t. $\mathcal{P}'$ satisfies Condition 11.*

Corollary 13 (Corollary of Theorems 9 and 12). *Let C be a $(d+1, v)$-gadget C for a function $f : \mathbb{K}^2 \to \mathbb{K}$ for which all probes are bilinear. If C is d-NI, then there is no set of d probes on C satisfying Condition 8. Furthermore, if there is no set of $t \leq d$ probes on C satisfying Condition 11, then C is d-NI.*[2]

The proof of Theorem 12 essentially relies on the following lemmas, conveniently formulated with linear codes[3]:

Lemma 14. *Let $\mathcal{C}_1$ (resp. $\mathcal{C}_2$) be an $[n_1, k]$ (resp. $[n_2, k]$, $n_2 > n_1$) linear code over a finite field $\mathbb{K}$. Let $\boldsymbol{G}_1 \in \mathbb{K}^{k\times n_1}$ and $\boldsymbol{G}_2 \in \mathbb{K}^{k\times n_2}$ be two generator matrices for $\mathcal{C}_1$ and $\mathcal{C}_2$ that have no zero column. Then the concatenated code $\mathcal{C}_{1,2}$ of $\mathcal{C}_1$ and $\mathcal{C}_2$ generated by $\boldsymbol{G}_{1,2} := \begin{pmatrix}\boldsymbol{G}_1 & \boldsymbol{G}_2\end{pmatrix}$ has the following property: $\exists\, \boldsymbol{c} \in \mathcal{C}_{1,2}$ s.t. $\mathrm{wt}_1(\boldsymbol{c}) < \mathrm{wt}_2(\boldsymbol{c})$, where $\mathrm{wt}_1(\cdot)$ (resp. $\mathrm{wt}_2(\cdot)$) denotes the Hamming weight function restricted to the first n_1 (resp. last n_2) coordinates of $\mathcal{C}_{1,2}$.*

One may remark that if $\#\mathbb{K}$ is sufficiently large w.r.t. the parameters of the codes, then by the Schwartz-Zippel-DeMillo-Lipton lemma there exists a word in $\mathcal{C}_{1,2}$ of maximal wt_2 weight, and the conclusion immediately follows; yet this argument does not hold over any field.

Lemma 15. *The statement of Lemma 14 still holds if $\mathbb{K}$ is replaced by a matrix ring ${\mathbb{K}'}^{d\times d}$ and if $\boldsymbol{G}_1$ is defined over the subfield of the scalar matrices of ${\mathbb{K}'}^{d\times d}$.*

We first recall the following:

Definition 16 (Shortening of a linear code). *Let $\mathcal{C}$ be an $[n, k]$ linear code over $\mathbb{K}$ generated by $\boldsymbol{G} \in \mathbb{K}^{k\times n}$. The* shortened code *$\mathcal{C}'$ w.r.t. coordinate $i \in [\![1, n]\!]$ is the subcode made of all codewords of $\mathcal{C}$ that are zero at coordinate i, with this coordinate then being deleted.*

We also give:

Definition 17 (Isolated coordinate). *Let $\boldsymbol{M} \in \mathbb{K}^{m\times n}$. A coordinate $i \in [\![1, n]\!]$ is called* isolated *for the row $\boldsymbol{M}_j$ of $\boldsymbol{M}$, $j \in [\![1, m]\!]$, iff. $\boldsymbol{M}_{j,i} \neq 0$ and $\forall j' \neq j \in [\![1, m]\!]$, $\boldsymbol{M}_{j',i} = 0$.*

And:

Procedure 18. We reuse the notation of the statement of Lemma 14. We *apply Procedure 18 on a row of $\boldsymbol{G}_{1,2}$* by doing the following: denote $\mathcal{I}_1$ (resp. $\mathcal{I}_2$) the (possibly empty) set of isolated coordinates on its first n_1 (resp. last n_2) columns; then if $\#\mathcal{I}_1 \geq \#\mathcal{I}_2$, shorten $\mathcal{C}_{1,2}$ w.r.t. all the coordinates in $\mathcal{I}_1 \cup \mathcal{I}_2$. Practically, this means deleting from $\boldsymbol{G}_{1,2}$ the row being processed and all the columns in $\mathcal{I}_1 \cup \mathcal{I}_2$. This results in a code $\mathcal{C}'_{1,2}$ generated by $\begin{pmatrix}\boldsymbol{G}'_1 & \boldsymbol{G}'_2\end{pmatrix}$ where $\boldsymbol{G}'_1 \in \mathbb{K}^{(k-1)\times n'_1}$ (resp. $\boldsymbol{G}'_2 \in \mathbb{K}^{(k-1)\times n'_2}$) is a submatrix of $\boldsymbol{G}_1$ (resp. $\boldsymbol{G}_2$) and $n'_1 < n_1$, $n'_2 < n_2$, $n'_1 < n'_2$, and none of the columns of $\boldsymbol{G}'_{1,2}$ is zero. One may also remark that since $\boldsymbol{G}'_1$ is of rank $k-1$, we have $k-1 \leq n'_1$.

[2] As Condition 11 directly implies an attack, one could also formulate this corollary solely in terms of this condition.

[3] Recall that an $[n, k]$ linear code over a field $\mathbb{K}$ is a k-dimensional linear subspace of $\mathbb{K}^n$.

We are now ready to prove Lemmas 14 and 15.

Proof (Lemma 14). We prove this lemma by induction using Procedure 18.

In a first step one applies Procedure 18 to every row of $\boldsymbol{G}_{1,2}$ one at a time and repeats this process again until either there is no row for which applying the procedure results in a shortening, or the dimension of the shortened code reaches 1.

In the latter case, this means that the only codeword in $\boldsymbol{G}'_{1,2} \in \mathbb{K}^{1\times(n'_1+n'_2)}$ is of full weight $n'_1 + n'_2$ with $n'_1 < n'_2$ (since $\boldsymbol{G}'_{1,2}$ only has a single row and none of its columns is zero). This induces a codeword c of $\mathcal{C}$ s.t. $\mathrm{wt}_1(c) = n'_1$ and $\mathrm{wt}_2(c) = n'_2$, so we are done.

In the former case, one is left with a matrix $\boldsymbol{G}'_{1,2} \in \mathbb{K}^{k'\times(n'_1+n'_2)}$, $k' > 1$. One then computes the reduced row echelon form of $\boldsymbol{G}'_{1,2}$ (this does not introduce any zero column since the elementary row operations are invertible) and again iteratively applies Procedure 18 on the resulting matrix as done in the first step. Now either the application of Procedure 18 leads to a shortened code of dimension 1 and then we are done as above, or we are left with a matrix $\boldsymbol{G}''_{1,2} \in \mathbb{K}^{k''\times(n''_1+n''_2)}$ which can be of two forms:

1. $k'' = n''_1$. Up to permutation of its columns, $\boldsymbol{G}''_{1,2}$ can be written as:

$$\left(\boldsymbol{I}_{n''_1} \,\middle|\, \boldsymbol{I}_{n''_1}\ \boldsymbol{I}_{n''_1}\ *\right),$$

   where $*$ is arbitrary. The left $k'' \times n''_1$ block is justified from $\boldsymbol{G}''_{1,2}$ being in reduced row echelon form and having full rank. The right $k'' \times n''_2$ block is justified from the fact that every row of the left block has exactly one isolated coordinate; since no simplification can be done anymore to $\boldsymbol{G}''_{1,2}$ by applying Procedure 18, this means that those rows have at least two isolated coordinates on the right block. This is enough to conclude on the existence of a codeword of $\mathcal{C}$ satisfying the desired property.
   Recall that it is not possible to have $k'' > n''_1$ from the last remark in Procedure 18. The only remaining case is then:
2. $k'' < n''_1$. Up to a permutation of its columns, the rank-k'' matrix $\boldsymbol{G}''_{1,2}$ can be written as:

$$\left(\boldsymbol{I}_{k''}\ *_L \,\middle|\, \boldsymbol{I}_{k''}\ \boldsymbol{I}_{k''}\ *_R\right),$$

   and it has no zero column. One then applies Lemma 14 inductively on the code generated by the submatrix $\boldsymbol{G}'''_{1,2} := \left(*_L \,\middle|\, \boldsymbol{I}_{k''}\ *_R\right)$ which is of strictly smaller length. Let $c''' = \boldsymbol{\lambda}\boldsymbol{G}'''_{1,2}$ be a codeword of this latter code that satisfies the desired property, then $\boldsymbol{\lambda}\boldsymbol{G}''_{1,2}$ also satisfies it for $\mathcal{C}_{1,2}$, which concludes the proof.

□

Proof (Lemma 15). The proof simply consists in remarking that all the steps of the proof of Lemma 14 can be carried out in the modified setting of Lemma 15. Mainly:

— Definitions 16 and 17 and Procedure 18 naturally generalise to matrices over rings, and the application of Procedure 18 is unchanged.

— Recall that by induction the left $k' \times n'_1$ submatrix is always of full rank k', which is also the rank of $\boldsymbol{G}'_{1,2}$. Since $\boldsymbol{G}_1$ is defined over scalar matrices, Gauß-Jordan elimination can be computed as if over a field.

□

The proof of Theorem 12 then follows.

Proof (Theorem 12). We start similarly from the proof of Theorem 9, and use the same notation: let $\mathcal{P}'$ be a set of $\ell - t$ linearly independent linear combinations of probes of $\mathcal{P}$ that do not depend on any random scalar, and let $\boldsymbol{D} = \left(\boldsymbol{M}'_{t+1}\ \boldsymbol{\mu}_{t+1} \cdots \boldsymbol{M}'_{\ell}\ \boldsymbol{\mu}_{\ell}\right)$ be the matrix that records the dependence of these probes on every share $\boldsymbol{a}_i$.We will show that $\exists \mathcal{P}'' \subseteq \mathcal{P}$ that satisfies Condition 11. To do this, we introduce two new indicator matrices:

— Let $\boldsymbol{\Pi} \in \mathbb{K}^{(d+2)\times(d+2)^{(\ell-t)\times\ell}}$ be s.t. for every $p' \in \mathcal{P}'$ it records in its rows its dependence on the probes of $\mathcal{P}$ as scalar matrices;[4] that is, $\boldsymbol{\Pi}$ is s.t. $p'_i = \sum_{j=1}^{\ell} \boldsymbol{\pi}_{i,j} p_j$ where $\boldsymbol{\pi}_{i,j}$ is the scalar on the diagonal of the scalar matrix $\boldsymbol{\Pi}_{i,j}$. W.l.o.g., we may assume that every probe of $\mathcal{P}$ appears at least once in a linear combination of $\mathcal{P}'$, otherwise it is simply discarded, so $\boldsymbol{\Pi}$ has no zero column.

[4] This use of scalar matrices is only so that $\boldsymbol{\Pi}$ is defined on the same base structure as $\boldsymbol{\Delta}$ below. As an example, taking $\ell = d = 2$ and considering two probes in $\mathcal{P}'$ as $p'_1 = p_1 + p_2$; $p'_2 = p_2$, then $\boldsymbol{\Pi} = \begin{pmatrix} \boldsymbol{I}_4 & \boldsymbol{I}_4 \\ \boldsymbol{0}_4 & \boldsymbol{I}_4 \end{pmatrix}$.

— Let $\boldsymbol{\Delta} \in \mathbb{K}^{(d+2)\times(d+2)^{(\ell-t)\times(d+1)}}$ be the matrix that for every $p' \in \mathcal{P}'$ records in its rows its dependence on the shares $\boldsymbol{a}_i$s; that is if the bilinear probe p'_i can be written as $p'_i = \boldsymbol{a}^t\boldsymbol{M}'\boldsymbol{b} + \boldsymbol{a}^t\boldsymbol{\mu}' + \boldsymbol{b}^t\boldsymbol{\nu}' + \tau'$, then $\boldsymbol{\Delta}_{i,j}$ is set to the diagonal matrix of the j^{th} row of $\begin{pmatrix}\boldsymbol{M}' & \boldsymbol{\mu}'\end{pmatrix}$.[5] Note that since by assumption $\boldsymbol{D}$ has no zero row, $\boldsymbol{\Delta}$ has no zero *column*.

Now we invoke Lemma 15 with $\boldsymbol{\Pi}$ as $\boldsymbol{G}_1$ and $\boldsymbol{\Delta}$ as $\boldsymbol{G}_2$ the generator matrices for the concatenated code $\mathcal{C}_{1,2}$. Let $\boldsymbol{c} \in \mathcal{C}_{1,2}$ be a codeword that satisfies $\mathrm{wt}_1(\boldsymbol{c}) < \mathrm{wt}_2(\boldsymbol{c})$; this translates to a linear combination of $\ell'' := \mathrm{wt}_1(\boldsymbol{c})$ probes of $\mathcal{P}'' \subseteq \mathcal{P}$ that (as linear combinations of elements of $\mathcal{P}'$) does not depend on any randomness and s.t. the associated matrix $\begin{pmatrix}\boldsymbol{M}'' & \boldsymbol{\mu}''\end{pmatrix}$ has $\mathrm{wt}_2(\boldsymbol{c}) \geq \ell'' + 1$ non-zero rows (by applying the inverse transformation from $\boldsymbol{\Delta}$ to $\boldsymbol{D}$), hence $\mathcal{P}''$ satisfies Condition 11. □

Finally, the proof of Corollary 13 is immediate from Theorems 9 and 12.

2.3 Matrix model for strong non-interference

We now wish to adapt the approach of Theorems 9 and 12 to be able to prove that a scheme is SNI. This is in fact quite straightforward, and it mostly consists in defining a suitable variant of Condition 11 and in applying Lemma 15 to well-chosen matrices, to show again that there is a subset of probes that satisfies the condition whenever there is an attack.

Condition 19. *A set of $\ell = \ell_1 + \ell_2$ bilinear probes $\mathcal{P} = \{p_1, \ldots, p_\ell\}$ on a $(d+1, v)$-gadget C for a function $f : \mathbb{K}^2 \to \mathbb{K}$, of which ℓ_1 are internal, satisfies Condition 19 iff. $\exists \boldsymbol{\lambda} \in \mathbb{K}^\ell$, $\boldsymbol{M} \in \mathbb{K}^{(d+1)\times(d+1)}$, $\boldsymbol{\mu}, \boldsymbol{\nu} \in \mathbb{K}^{d+1}$, and $\tau \in \mathbb{K}$ s.t. $\sum_{i=1}^{\ell} \boldsymbol{\lambda}_i p_i = \boldsymbol{a}^t\boldsymbol{M}\boldsymbol{b} + \boldsymbol{a}^t\boldsymbol{\mu} + \boldsymbol{b}^t\boldsymbol{\nu} + \tau$ and the block matrix $\begin{pmatrix}\boldsymbol{M} & \boldsymbol{\mu}\end{pmatrix}$ (resp. the block matrix $\begin{pmatrix}\boldsymbol{M} \\ \boldsymbol{\nu}^t\end{pmatrix}$) has at least $\ell_1 + 1$ non-zero rows (resp. columns).*

Theorem 20. *Let $\mathcal{P}$ be a set of at most d bilinear probes on a $(d+1, v)$-gadget C for a function $f : \mathbb{K}^2 \to \mathbb{K}$, of which ℓ_1 are internal. If $\mathcal{P}$ is not ℓ_1-simulatable then $\exists \mathcal{P}' \subseteq \mathcal{P}$ s.t. $\mathcal{P}'$ satisfies Condition 19.*

Proof. We reuse the notation of Theorems 9 and 12. The proof is essentially the same as the one of Theorem 12, except that we only account for internal probes in $\boldsymbol{\Pi}$. Let $\mathcal{P}'$ be a set of $\ell - t$ linearly independent linear combinations of probes of $\mathcal{P}$ that do not depend on any random scalar, and let $\boldsymbol{D} = \begin{pmatrix}\boldsymbol{M}'_{t+1} & \boldsymbol{\mu}_{t+1} & \cdots & \boldsymbol{M}'_\ell & \boldsymbol{\mu}_\ell\end{pmatrix}$ be the matrix that records the dependence of these probes on every share $\boldsymbol{a}_i$. From the assumption that $\mathcal{P}$ is not ℓ_1-simulatable, we have that w.l.o.g., $\boldsymbol{D}$ has at least $\ell_1 + 1$ non-zero rows.

— Let $\boldsymbol{\Pi} \in \mathbb{K}^{(d+2)\times(d+2)^{(\ell-t)\times\ell_1}}$ be s.t. for every $p' \in \mathcal{P}'$ it records in its rows its dependence on the ℓ_1 *internal* probes (w.l.o.g. $\{p_1, \ldots, p_{\ell_1}\}$) of $\mathcal{P}$ as scalar matrices; that is, $\boldsymbol{\Pi}$ is s.t. $p'_i = \sum_{j=1}^{\ell_1} \boldsymbol{\pi}_{i,j} p_j + \sum_{j=\ell_1+1}^{\ell} \alpha_j p_j$, where $\boldsymbol{\pi}_{i,j}$ is the scalar on the diagonal of the scalar matrix $\boldsymbol{\Pi}_{i,j}$ and the α_js are unimportant. W.l.o.g., we may assume that every internal probe of $\mathcal{P}$ appears at least once in a linear combination of $\mathcal{P}'$, otherwise it is simply discarded, so $\boldsymbol{\Pi}$ has no zero column.

— Let $\boldsymbol{\Delta} \in \mathbb{K}^{(d+2)\times(d+2)^{(\ell-t)\times d'}}$ be the matrix that for every $p' \in \mathcal{P}'$ records in its rows its dependence on the shares $\boldsymbol{a}_i$s. If a row of $\boldsymbol{D}$ is all zero, the corresponding column is not included in $\boldsymbol{\Delta}$, and since $\boldsymbol{D}$ has at least $\ell_1 + 1$ non-zero rows, $\boldsymbol{\Delta}$ has at least $d' \geq \ell_1 + 1$ columns none of which are zero.

Now we invoke Lemma 15 with $\boldsymbol{\Pi}$ as $\boldsymbol{G}_1$ and $\boldsymbol{\Delta}$ as $\boldsymbol{G}_2$ the generator matrices for the concatenated code $\mathcal{C}_{1,2}$. Let $\boldsymbol{c} \in \mathcal{C}_{1,2}$ be a codeword that satisfies $\mathrm{wt}_1(\boldsymbol{c}) < \mathrm{wt}_2(\boldsymbol{c})$; this translates to a linear combination of $\ell'' := \mathrm{wt}_1(\boldsymbol{c})$ internal probes to which one can add a linear combination of up to ℓ_2 external probes s.t. it does not depend on any randomness and the associated matrix $\begin{pmatrix}\boldsymbol{M}'' & \boldsymbol{\mu}''\end{pmatrix}$ has $\mathrm{wt}_2(\boldsymbol{c}) \geq \ell'' + 1$ non-zero rows. The set $\mathcal{P}'' \subseteq \mathcal{P}$ of these internal and external probes thus satisfies Condition 19. □

And we then have the immediate corollary:

Corollary 21. *Let C be a $(d+1, v)$-gadget for a function $f : \mathbb{K}^2 \to \mathbb{K}$ for which all probes are bilinear. If there is no set of $t \leq d$ probes on C satisfying Condition 19, then C is d-SNI.*

[5] This use of diagonal matrices allows to keep track of (the lack of) simplifications when combining several probes; for instance, if two probes depend on the same $\boldsymbol{a}_i$ as $\boldsymbol{a}_i\boldsymbol{b}_j$ and $\boldsymbol{a}_i\boldsymbol{b}_{j'}$ with $j \neq j'$, then the sum of those probes still depends on $\boldsymbol{a}_i$. Continuing the previous example and taking $p'_1 = \boldsymbol{a}_0\boldsymbol{b}_0 + \boldsymbol{a}_0\boldsymbol{b}_1 + \boldsymbol{a}_1\boldsymbol{b}_2 + \boldsymbol{a}_2$, then the first row of $\boldsymbol{\Delta}$ (whose entries are 4×4 matrices) is $\begin{pmatrix} 1 & & & & 0 & & & & 0 & & & \\ & 1 & & & & 0 & & & & 0 & & \\ & & 0 & & & & 1 & & & & 0 & \\ & & & 0 & & & & 0 & & & & 1 \end{pmatrix}$.

2.4 Security of binary schemes over finite fields of characteristic two

Let C be a d-NI or SNI gadget for a function defined over $\mathbb{F}_2$; a natural question is whether its security is preserved if it is lifted to an extension $\mathbb{F}_{2^n}$. Indeed, the probes available to the adversary are the same in the two cases, but the latter offers more possible linear combinations $\sum_{i=1}^{\ell} \boldsymbol{\lambda}_i p_i$, since the $\boldsymbol{\lambda}_i$s are no longer restricted to $\{0,1\}$. We answer this question positively, and give a simple proof based on Theorems 12 and 20.

Theorem 22. *Let C be a d-NI (resp. d-SNI) gadget for a function $f : \mathbb{F}_2^2 \to \mathbb{F}_2$, then for any n, the natural lifting $\widehat{C}$ of C to $\widehat{f} : \mathbb{F}_{2^n}^2 \to \mathbb{F}_{2^n}$ is also d-NI (resp. d-SNI).*

Proof. We only prove the d-NI case, the d-SNI one being similar. From Corollary 13, it is sufficient to show that if $\nexists \mathcal{P}$ for C that satisfies Condition 11, then the same holds for $\widehat{C}$. We do this by showing the following contrapositive: if a set of probes $\mathcal{P}$ is not d-simulatable for $\widehat{C}$, then it is not d-simulatable either for C.

From the proofs of Theorems 9 and 12, if $\mathcal{P}$ is not d-simulatable for $\widehat{C}$, then there is a matrix $\widehat{\boldsymbol{D}}$ that leads to the existence of $\mathcal{P}'$ s.t. Condition 11 is satisfied. All we need to do is showing that a similar matrix $\boldsymbol{D}$ can also be found for C. Since C is defined over $\mathbb{F}_2$, the matrices $\boldsymbol{R}$ and $\boldsymbol{P}$, and thence $\widehat{\boldsymbol{R}}$ and $\widehat{\boldsymbol{P}}$ have all their coefficients in $\{0,1\}$. As 1 is its own inverse, the change-of-basis matrix from $\widehat{\boldsymbol{R}}$ to $\widehat{\boldsymbol{R}}'$ is also binary; equivalently, this means that the Gauß-Jordan elimination of $\widehat{\boldsymbol{R}}$ can be done in the subfield $\mathbb{F}_2$. Thus one only has to take $\boldsymbol{D} = \widehat{\boldsymbol{D}}$ to satisfy Condition 11 on C. $\square$

This result is quite useful as it means that the security of a binary scheme only needs to be proven once in $\mathbb{F}_2$, even if it is eventually used in one or several extension fields. Proceeding thusly is in particular beneficial in terms of verification performance, since working over $\mathbb{F}_2$ limits the number of linear combinations to consider and may lead to some specific optimisations (cf. *e.g.* Sections 3 and 4).

Remark. This result was in fact already implicitly used (in a slight variant) by Barthe *et al.* in their masking compiler [BBD+15] and in maskVerif [BBC+19], since they use gadgets defined over an arbitrary structure $(\mathbb{K}, 0, 1, \oplus, \ominus, \odot)$. However we could not find a proof, which actually seems necessary to justify the correctness of this approach and of our algorithms of the next section.

3 An algorithm for checking non-interference

In this section, we present a new efficient algorithm to check if a scheme is (strong) non-interfering. This algorithm is a modification of the one presented by Belaïd *et al.* at EUROCRYPT 2016 [BBP+16, Section 8], and its correctness crucially relies on Theorems 12 and 20; it thus only applies to schemes for which all probes are bilinear, but this is not a hard restriction in practice.

In all of the following we assume that the field $\mathbb{K}$ over which the scheme is defined is equal to $\mathbb{F}_2$, which means that we will simultaneously assess its security in that field and all its extensions (cf. Section 2.4). Some discussion of implementation in the NI case for schemes natively defined over larger fields (meaning that shares or random masks may be multiplied by constants not in $\{0,1\}$) for which the new Theorem 12 is not needed can be found in [KR18].

We start by introducing some vocabulary and by recalling the algorithm from Belaïd *et al.*.

Definition 23 (Elementary probes). *A probe p is called* elementary *if it is of the form $p = \boldsymbol{a}_i \boldsymbol{b}_j$ (*elementary *deterministic* probe*) or $p = \boldsymbol{r}_i$ (*elementary *random* probe*).*

Definition 24 (Shares indicator matrix). *Let p be a bilinear probe. We call* shares indicator matrix *and write $\boldsymbol{M}_p$ the matrix $\boldsymbol{M}$ from Definition 6.*

Definition 25 (Randomness indicator matrix). *Let p be a bilinear probe. We call* randomness indicator matrix *and write $\boldsymbol{\sigma}_p$ the column matrix $\boldsymbol{\sigma}$ from Definition 6.*

3.1 The algorithm from EUROCRYPT 2016

At EUROCRYPT 2016, Belaïd *et al.* presented an efficient probabilistic algorithm to find potential attacks against the d-privacy notion[6] for masking schemes for the multiplication over $\mathbb{F}_2$. By running the algorithm many times and not detecting any attack, one can also establish the security

[6] It can also be trivially modified to check attacks against NI security.

of a scheme up to some probability, but deriving a deterministic counterpart is less trivial. This algorithm works as follows.

Consider a scheme on which all possible probes $\mathcal{P}$ are bilinear, and let $\boldsymbol{H}_{\mathcal{P}} := \left(\boldsymbol{\sigma}_p\right), p \in \mathcal{P}$ be the block matrix constructed from the all the corresponding randomness indicator matrices. The algorithm of [BBP+16, Section 8] starts by finding a set of fewer than d probes whose sum[7] does not depend on any randomness. That is to say, it is looking for a vector $\boldsymbol{x}$ such that $\boldsymbol{H}_{\mathcal{P}} \cdot \boldsymbol{x} = \boldsymbol{0}$ and $\mathrm{wt}(\boldsymbol{x}) \leq d$. This can be immediately reformulated as a coding problem, as one is in fact searching for a codeword of weight less than d in the dual code of $\boldsymbol{H}_{\mathcal{P}}$. This search can then be performed using any information set decoding algorithm, and Belaïd *et al.* used the original one of Prange [Pra62].[8] Once such a set has been found, it is tested against [BBP+16, Condition 2] (which is similar to Condition 8) to determine if it is a valid attack against the d-NI notion, and [BBP+16, Condition 1] to determine if it is an attack for d-privacy. This procedure is then repeated until an attack is found or one has gained sufficient confidence in the security of the scheme.

Removing elementary *deterministic* probes. To make the above procedure more efficient, an important observation made by Belaïd *et al.* is that if the sum of every probe of a given set does not functionally depend on some $\boldsymbol{a}_i$ or $\boldsymbol{b}_j$, it is always possible to make it so by adding a corresponding elementary probe $\boldsymbol{a}_i\boldsymbol{b}_j$. This can be used to check, say, d-NI security by simply comparing the number of missing $\boldsymbol{a}_i$ or $\boldsymbol{b}_j$ to $d - \mathrm{wt}(\boldsymbol{x})$. This allows to reduce the number of probes that one has to include in $\mathcal{P}$ (and thus the dimension of $\boldsymbol{H}_{\mathcal{P}}$), making the algorithm more efficient.

3.2 A new algorithm based on enumeration

We now describe a new algorithm based on a partial enumeration of the power set $\wp(\mathcal{P})$ of $\mathcal{P}$. The idea is to simply consider every sum of fewer than d probes and to check if it depends on all shares and no random masks, relying on Corollaries 13 and 21 for correctness. Since the cost of such an enumeration quickly grows with the size of $\mathcal{P}$, we then follow and extend the above observation by Belaïd *et al.* and only perform the enumeration on a reduced set. We first describe a simple extension of this "dimension reduction" strategy, before detailing the algorithms themselves. A more elaborate dimension reduction process is then described in Section 3.3, and we discuss implementation aspects in Section 4.

Removing elementary *random* probes. It is easy to adapt a deterministic enumeration so that one can completely remove elementary random probes; it suffices to remark that if the sum of every probe of a given set functionally depends on some $\boldsymbol{r}_i$, it is always possible to make it not so by adding the corresponding elementary probes.

Combining the two above observations, we may remove every elementary probe from the set $\mathcal{P}$.[9] This can be summarized by saying that in the enumeration, one is not restricted anymore to finding exactly a combination of fewer than d probes that depends on all shares and no random masks, as it is enough to find a combination of $\ell \leq d$ probes that depends on u shares and v masks as long as $d - \ell \geq (d + 1 - u) + v$, since the missing shares and extra masks can be dealt with elementary probes in a predictable way. This is in fact exactly the check that is performed in our implementation in the case of NI security, as is detailed and justified below.

Checking a scheme for non-interference. We now state the following:

Proposition 26. *Let C be a $(d+1, v)$-gadget for a function $f : \mathbb{F}_2^2 \to \mathbb{F}_2$ for which all probes are bilinear, and $\mathcal{Q}_0$ be a set of n_0 non-elementary probes on C that functionally depends on n_a shares $\boldsymbol{a}_i$s, n_b shares $\boldsymbol{b}_j$s, and n_r random scalars $\boldsymbol{r}_i$s. Let $\mathcal{Q}_1$ be one of the smallest sets of elementary probes needed to complete $\mathcal{Q}_0$ such that $\mathcal{Q}_0 \cup \mathcal{Q}_1$ satisfies Condition 11 and functionally depends on all the $\boldsymbol{a}_i$s or all the $\boldsymbol{b}_i$s.*[10] *Then $n_1 := \#\mathcal{Q}_1 = n_r + (d + 1 - \max(n_a, n_b))$.*

[7] That is, the only non-trivial linear combination over $\mathbb{F}_2$ that depends on all the elements of the set.

[8] One may remark that since information set decoding relies on Gaussian elimination, the cost of one step of this algorithm increases more than linearly in the size of $\mathcal{P}$.

[9] Note that this means that one would not detect the existence of an attack that would use *only* elementary probes. However, it is easy to see from their definitions that ℓ such probes functionally depend on at most ℓ shares, and so can never lead to a non-trivial attack.

[10] This additional constraint is not in itself necessary, but it simplifies the overall algorithm.

Proof. An elementary probe functionally depends on either one $\boldsymbol{r}_i$ or one $\boldsymbol{a}_i$ and one $\boldsymbol{b}_j$, but not both. Thus, the minimum number of elementary probes needed to cancel every $\boldsymbol{r}_i$ and to add the $d+1-n_a$ (resp. $d+1-n_b$) missing $\boldsymbol{a}_i$s (resp. $\boldsymbol{b}_j$s) in $\mathcal{Q}_0$ is $n_r + (d+1-n_a)$ (resp. $n_r + (d+1-n_b)$). Thus, $\#\mathcal{Q}_1 = \min(n_r + d + 1 - n_a, n_r + d + 1 - n_b) = n_r + d + 1 - \max(n_a, n_b)$. ◻

This proposition can then be used in a straightforward way to check if a scheme is d-NI. To do so, one simply has to enumerate every set $\mathcal{Q}_0 \in \wp(\mathcal{P})$ of d non-elementary probes or fewer and to check if $n_0 + n_1 \leq d$. By Corollary 13, if no such set $\mathcal{Q}_0$ can be completed as in Proposition 26 and still contain fewer than d probes, then the scheme is d-NI.

Checking a scheme for strong non-interference. We only need to adapt Proposition 26 to distinguish between internal and external probes:

Proposition 27. *Let C be a $(d+1, v)$-gadget for a function $f : \mathbb{F}_2^2 \to \mathbb{F}_2$ for which all probes are bilinear, and $\mathcal{Q}_0$ be a set of n_0 non-elementary probes on C that functionally depends on n_a shares $\boldsymbol{a}_i$s, n_b shares $\boldsymbol{b}_j$s, and n_r random scalars $\boldsymbol{r}_i$s. Let n_I denote the number of internal probes in $\mathcal{Q}_0$. Then there is a set $\mathcal{Q}_1$ of n_r elementary random probes such that $\mathcal{Q}_0 \cup \mathcal{Q}_1$ satisfies Condition 19 iff.* $\max(n_a, n_b) > n_I + n_r$.

Proof. Recall that all elementary probes are internal. If $\mathcal{Q}_0$ does not satisfy Condition 19, then adding an elementary deterministic probe increases by at most one the number of non-zero rows, while increasing by one the total number of probes, so this completed set does not satisfy $\mathcal{Q}_0$ either. It is thus enough to only consider random probes in $\mathcal{Q}_1$.

For $\mathcal{Q} = \mathcal{Q}_0 \cup \mathcal{Q}_1$ to satisfy Condition 19, it is necessary to cancel all the potential randomness $\boldsymbol{r}_i$s on which $\mathcal{Q}_0$ depends; so $\mathcal{Q}_1$ must be the (possibly empty) set of the n_r corresponding elementary random probes. Now $\mathcal{Q}$ contains $n_I + n_r$ internal probes and it functionally depends on n_a $\boldsymbol{a}_i$s and n_b $\boldsymbol{b}_j$s. Thus it satisfies Condition 19 iff. $\max(n_a, n_b) > n_I + n_r$. ◻

This proposition can then be used in a straightforward way to check if a scheme is d-SNI. To do so, one simply has to enumerate every set $\mathcal{Q}_0 \in \wp(\mathcal{P})$ of d non-elementary probes or fewer and to check if $\max(n_a, n_b) > n_I + n_r$ and $n_0 + n_r \leq d$. If no such set satisfying this condition is found, then the scheme is d-SNI by Corollary 21.

3.3 Dimension reduction

To further reduce the size of the space to explore during the verification, it may be possible to filter additional non-elementary probes from the set $\mathcal{P}$, in the case where they can be replaced by "better" ones. To do this while ensuring the correctness of our verification algorithm, we first define the following:

Definition 28 (Equipotent sets). *Let $\mathcal{P} := \cup_{k=0}^{v} \mathcal{P}_k$ and $\mathcal{P}' := \cup_{k=0}^{v} \mathcal{P}'_k$ be two sets of probes on a $(d+1, v)$-gadget C for a function $f : \mathbb{F}_2^2 \to \mathbb{F}_2$ for which all probes are bilinear, where $\mathcal{P}_k$ (resp. $\mathcal{P}'_k$) denotes the probes on the wires of C that are connected to the output share $\boldsymbol{c}_k$. Then $\mathcal{P}'$ is said to be* equipotent *to $\mathcal{P}$ iff.:*

- $\#\mathcal{P}' \leq \#\mathcal{P}$
- $\forall k, \forall \boldsymbol{\lambda} \in \mathbb{K}^{\#\mathcal{P}_k}, \exists \boldsymbol{\lambda}' \in \mathbb{K}^{\#\mathcal{P}'_k}$, $\mathrm{wt}(\boldsymbol{\lambda}') \leq \mathrm{wt}(\boldsymbol{\lambda})$ *for which, reusing the notation of Definition 6,* $\sum_{p_i \in \mathcal{P}} \boldsymbol{\lambda}_i p_i$ *and* $\sum_{p'_i \in \mathcal{P}'} \boldsymbol{\lambda}'_i p'_i$ *are such that:*
  - $\boldsymbol{\sigma} = \boldsymbol{\sigma}'$
  - $\mathrm{supp}(\boldsymbol{M}) \subseteq \mathrm{supp}(\boldsymbol{M}')$, $\mathrm{supp}(\boldsymbol{\mu}) \subseteq \mathrm{supp}(\boldsymbol{\mu}')$, $\mathrm{supp}(\boldsymbol{\nu}) \subseteq \mathrm{supp}(\boldsymbol{\nu}')$, *where by* $\mathrm{supp}(\boldsymbol{A})$ *we denote the set of non-zero coefficients of* $\boldsymbol{A}$.

In other words a set is equipotent to another iff. the shares and randomness dependence of any linear combination of elements of the latter can be "covered" by a linear combination of equal or lower weight of the former. We then have:

Lemma 29. *If two linear combinations of probes $\sum \boldsymbol{\lambda}_i p_i$ and $\sum \boldsymbol{\lambda}'_i p'_i$ functionally depend on disjoint sets of elementary probes and shares $\boldsymbol{a}_i \boldsymbol{b}_j$, $\boldsymbol{a}_i$ and $\boldsymbol{b}_j$, then their sum functionally depends on the union of those sets.*

Proof. Immediate, since using the notation of Definition 6, the supports of $\boldsymbol{M}$, $\boldsymbol{\mu}$, $\boldsymbol{\nu}$ are disjoint from the ones of $\boldsymbol{M}'$, $\boldsymbol{\mu}'$, $\boldsymbol{\nu}'$. ◻

Finally, we conclude with the following:

Proposition 30. *Let $\mathcal{P}$ and $\mathcal{P}'$ be two equipotent sets of probes on a $(d+1, v)$-gadget C for a function $f : \mathbb{F}_2^2 \to \mathbb{F}_2$ for which all probes are bilinear and for which all output shares functionally depend on pairwise disjoint sets of elementary probes and shares $\boldsymbol{a}_i\boldsymbol{b}_j$, $\boldsymbol{a}_i$ and $\boldsymbol{b}_j$. Then if $\mathcal{Q} \subseteq \mathcal{P}$ satisfies Condition 11, $\exists\, \mathcal{Q}' \subseteq \mathcal{P}'$, $\#\mathcal{Q}' \leq \#\mathcal{Q}$ that also satisfies Condition 11.*

Proof. Let us write $\mathcal{Q}$ as $\cup_{k=0}^{v} \mathcal{Q}_k$ (resp. $\mathcal{Q}'$ as $\cup_{k=0}^{v} \mathcal{Q}'_k$) where $\mathcal{Q}_k$ (resp. $\mathcal{Q}'_k$) denotes the probes on the wires of C that are connected to the output share $\boldsymbol{c}_k$. Let $\sum_{p_i \in \mathcal{Q}} \boldsymbol{\lambda}_i p_i$ denote one linear combination of elements of $\mathcal{Q}$ whose existence is guaranteed by its satisfying Condition 11, which we rewrite as: $\sum_k \sum_{p_i^{(k)} \in \mathcal{Q}_k} \boldsymbol{\lambda}_i^{(k)} p_i^{(k)}$. For each $\boldsymbol{\lambda}^{(k)}$, let $\boldsymbol{\lambda}'^{(k)}$ be the coefficients for one of the linear combination of elements of $\mathcal{Q}'_k$ whose existence is guaranteed by $\mathcal{P}'$ being equipotent to $\mathcal{P}$. Then by applying Lemma 29 to each of $\sum_k \sum_{p_i^{(k)} \in \mathcal{Q}_k} \boldsymbol{\lambda}_i^{(k)} p_i^{(k)}$ and $\sum_k \sum_{p_i^{(k)} \in \mathcal{Q}'_k} \boldsymbol{\lambda}'^{(k)}_i p_i^{(k)}$, it follows that the latter does not functionally depend on any elementary random probe $\boldsymbol{r}_i$, and the elementary deterministic probes and shares on which it functionally depends is a superset of the ones on which depends the former; thus $\mathcal{Q}'$ satisfies Condition 11. ◻

Examples. Consider a set $\mathcal{P}$ of two probes $\boldsymbol{a}_0\boldsymbol{b}_0 + \boldsymbol{r}_0 + \boldsymbol{a}_0\boldsymbol{b}_1$ and $\boldsymbol{a}_0\boldsymbol{b}_0 + \boldsymbol{r}_0 + \boldsymbol{a}_0\boldsymbol{b}_1 + \boldsymbol{a}_1\boldsymbol{b}_0$ on the same output share. Then provided that none of the $\boldsymbol{a}_i\boldsymbol{b}_j$ appear in other output shares, this set can be simplified by keeping only the second probe, since it covers all the shares of the first one.

On the other hand, a set containing two probes $\boldsymbol{a}_0\boldsymbol{b}_0 + \boldsymbol{r}_0 + \boldsymbol{a}_0\boldsymbol{b}_1 + \boldsymbol{a}_1\boldsymbol{b}_0$ and $\boldsymbol{a}_0\boldsymbol{b}_0 + \boldsymbol{r}_0 + \boldsymbol{a}_0\boldsymbol{b}_1 + \boldsymbol{a}_1\boldsymbol{b}_0 + \boldsymbol{r}_1$ cannot be simplified since the two probes do not include exactly the same random masks.

We will see in Section 5 how Proposition 30 can be used in practice to significantly improve verification performance. The nature of the probes that can be removed of course depends on the scheme under consideration, and we will later detail how to do this for some concrete gadgets.

3.4 Adaptation to the robust probing model

From a security perspective, a major difference between software and hardware masked implementations of cryptographic algorithms is that in the latter case, physical defects may render the countermeasure ineffective. In the case of one such phenomenon known as *glitches*, a probe at an arithmetic gate (*i.e.* an addition or a multiplication) can leak more to the adversary than its sole output — something that is not taken into account in the usual probing model. In an effort to remedy this situation, Faust *et al.* recently proposed to extend probing security into a *robust probing model* [FGP+18], able to take several types of hardware defects into account. In the case of glitches, this is done by assuming that a probe at an arithmetic gate leaks the union of what is leaked by its two inputs. One consequence is that if two arithmetic gates are connected together, leakage at the first one also propagates to the second. To stop this propagation, one must then use a memory gate (a register), which only leaks its output value.

Concretely, the robust probing model defines a leakage set $\mathfrak{L}(p)$ of possibly more than one value for every probe p at an arbitrary gate. This is more complex than, and not directly compatible with the usual probing security model and how we exploit it in our algorithm, where a probe leaks a single expression and verification implies enumerating and summing all subsets of size up to some order d. Nevertheless, one can opt for the following simple two-step strategy: 1) iterate over all subsets $\mathcal{P}$ of d probes or fewer; 2) then compute and check every possible full-weight linear combination of values leaked by this set of probes. In a non-robust model and for schemes over $\mathbb{F}_2$, step 2) only involves a single expression (*viz.* the sum of all the single values leaked by each probe), but in a robust model there are in general $\prod_{p \in \mathcal{P}} \left(2^{\#\,\mathfrak{L}(p)} - 1\right)$ expressions to consider (since for each probe one must now consider all the non-trivial binary linear combinations of the values it leaked).

Related work. The maskVerif tool [BBC+19] is also able to check security in presence of glitches, and similarly relies on the robust probing model for that purpose.

Another approach to analyse hardware schemes has been proposed by Bloem *et al.* [BGI+18]. Even if the underlying model is not the same as the one of Faust *et al.*, one of the steps in the verification also involves computing and considering leakage sets formed by every linear combination of the inputs to arithmetic gates.

4 Implementation

We now describe an efficient `C` implementation of the algorithm of the previous section for $\mathbb{K} = \mathbb{F}_2$. Our software is publicly available at: https://github.com/NicsTr/binary_masking.

4.1 Data structures and vectorisation

To evaluate if a set of probes $\mathcal{P}$ may lead to an attack, it is convenient to define the following:

Definition 31 (Attack matrix). *The* attack matrix $\boldsymbol{A}_{\mathcal{P}}$ *of a set of probes* $\mathcal{P}$ *is defined as the sum of the share indicator matrices of the probes in* $\mathcal{P}$*:*

$$\boldsymbol{A}_{\mathcal{P}} = \sum_{p \in \mathcal{P}} \boldsymbol{M}_p.$$

Definition 32 (Noise matrix). *The* noise matrix $\boldsymbol{B}_{\mathcal{P}}$ *of a set of probes* $\mathcal{P}$ *is defined as the sum of the randomness indicator matrices of the probes in* $\mathcal{P}$*:*

$$\boldsymbol{B}_{\mathcal{P}} = \sum_{p \in \mathcal{P}} \boldsymbol{\sigma}_p.$$

One can then simply compute the quantities n_a, n_b and n_r needed in Propositions 26 and 27 as the number of non-zero rows or columns of these two matrices. To analyse a given scheme, one then just has to provide a full description of $\boldsymbol{M}_p$ and $\boldsymbol{\sigma}_p$ for every non-elementary probe. Additionally, since Proposition 27 requires to compute the number of internal probes n_I in a set, those have to be labelled as such.

We inline all data structures and store them in either standard or vector registers. $\boldsymbol{A}_{\mathcal{P}}$ is stored twice, once row-wise and once column-wise, in order to avoid the otherwise costly transposition needed to compute both its row and its column "Hamming weight". For schemes at order $d \leq 15$, each row or column fits within a 16-bit words leading to a quite efficient vectorised Hamming weight computation, as shown in Listing 1. We also provide a slower implementation for schemes at higher order; in this case actually proving the security with our algorithm is likely to be intractable due to the combinatorial explosion of the number of sets to consider, yet a partial run may still be able to detect attacks, in the fashion of the original algorithm from EUROCRYPT 2016.

```c
int popcount256_16(__m256i v)
{
	return __builtin_popcountl(_mm256_cmpgt_epi16_mask(v, _mm256_setzero_si256()));
}
```

Listing 1: Hamming weight computation of a vector of dimension 16 over 16-bit words using AVX512VL and AVX512BW; a variant with only a few more instructions can be used with only AVX2.

4.2 Amortised enumeration & parallelisation

Recall that to prove the security of a scheme at order d, the algorithm of Section 3 requires to enumerate all the $\sum_{i=1}^{d} \binom{n}{i}$ subsets of a (possibly filtered) set of probes $\mathcal{P}$ of size n. For a subset $\mathcal{P}' \subseteq \mathcal{P}$ of size ℓ, a naïve approach in computing $\boldsymbol{A}_{\mathcal{P}'}$ would use $\ell - 1$ additions, and this for every such $\mathcal{P}'$. However, a well-known optimisation for this kind of enumeration is instead to go through all the subsets of a fixed weight in a way that ensures that two consecutive sets $\mathcal{P}'$ and $\mathcal{P}''$ only differ by two elements. One can then compute, say, $\boldsymbol{A}_{\mathcal{P}''}$ efficiently by updating $\boldsymbol{A}_{\mathcal{P}'}$ with one addition and one subtraction. We do this in our implementation by using a so-called "revolving-door algorithm" (cf. *e.g.* [Knu11, Algorithm R]) for the Nijenhuis-Wilf-Liu-Tang "combination Gray code"[NW78,LT73].

In the robust probing model setting, one may also need to enumerate more than one expression for a given set of probes. This can still be done efficiently with Gray codes: an outer mixed-radix code can be used to go through the $\prod_{p \in \mathcal{P}'}(2^{\#\mathfrak{L}(p)} - 1)$ considered linear combinations by only "incrementing" the leaked value of a single probe at a time, and an inner binary code can be used to efficiently implement this incrementation for every probe.

The enumeration can also be easily parallelised, and the main challenge is to couple it with the above amortised approach. This can in fact be done quite efficiently, as the combination Gray code that we use possesses an efficient *unranking* map from the integers to arbitrary configurations [Wal]. One can then easily divide a full enumeration of a total of n combinations into j jobs by starting each of them independently at one of the configurations given by the unranking of $i \times n/j$, $i \in [\![0, j[\![$.

4.3 From high-level representation to `C` description

We use a custom parser to convert a readable description of a masking scheme into a `C` description of its probes' indicator matrices.

Each line of the high-level description corresponds to an output share. The available symbols are:

— `sij` which represents a product $\boldsymbol{a}_i\boldsymbol{b}_j$;
— `ri` which represents a random mask $\boldsymbol{r}_i$;
— a space '␣', a binary operator which represents an addition (*i.e.* XOR) gate;
— parentheses, which allow explicit scheduling of the operations;
— `|`, a postfixed unary operator which represents the use of a register to store the expression that is *before* the symbol. This is only needed for an analysis in presence of glitches.

Additionally, the user needs to specify the order d of the scheme as well as the list of random masks used.

The scheduling of the operations needed to compute the output shares is important, as it determines the probes available to the adversary. In that respect, the parser uses by default an implicit left-to-right scheduling and addition gates have precedence over registers. As an example the scheme whose output shares are defined as:

$$\begin{aligned}
\boldsymbol{c}_0 &= ((((\boldsymbol{a}_0\boldsymbol{b}_0 \oplus \boldsymbol{r}_0) \oplus \boldsymbol{a}_0\boldsymbol{b}_1) \oplus \boldsymbol{a}_1\boldsymbol{b}_0) \oplus \boldsymbol{r}_1)\\
\boldsymbol{c}_1 &= ((((\boldsymbol{a}_1\boldsymbol{b}_1 \oplus \boldsymbol{r}_1) \oplus \boldsymbol{a}_1\boldsymbol{b}_2) \oplus \boldsymbol{a}_2\boldsymbol{b}_1) \oplus \boldsymbol{r}_2)\\
\boldsymbol{c}_2 &= ((((\boldsymbol{a}_2\boldsymbol{b}_2 \oplus \boldsymbol{r}_2) \oplus \boldsymbol{a}_2\boldsymbol{b}_0) \oplus \boldsymbol{a}_0\boldsymbol{b}_2) \oplus \boldsymbol{r}_0)
\end{aligned}$$

is described by the file:

```
ORDER = 2
MASKS = [r0, r1, r2]
s00 r0 s01 s10 r1
s11 r1 s12 s21 r2
s22 r2 s20 s02 r0
```

Another example is the following *DOM-indep* multiplication by Groß *et al.*, which is NI at order two even in the presence of glitches:

```
ORDER = 2
MASKS = [r0, r1, r2]
 s00      (s01 r0|) (s02 r1|)
(s10 r0|)  s11      (s12 r2|)
(s20 r1|) (s21 r2|)  s22
```

Fig. 1: High-level representation of the glitch-resistant DOM-indep multiplication at order 2 [GMK16].

5 Applications

In this section we apply our fast implementation of the verification algorithm of Section 3 to various state-of-the-art masking gadgets and also propose new improved instances in medium order, including better SNI multiplication and refreshing gadgets for the practically-relevant case of 8 shares.

We analyse:

— In Section 5.1: NI and SNI multiplication gadgets originally from [BDF+17,GPSS18].
— In Section 5.2: SNI refreshing gadgets originally from [BDF+17,BBD+18].
— In Section 5.3: Glitch-resistant NI multiplication from [GMK16].

5.1 NI and SNI multiplication gadgets

We first study a family of multiplication gadgets that were introduced by Barthe *et al.* at EUROCRYPT 2017 [BDF+17] and used in the efficient masked AES implementation of Grégoire *et al.* [GPSS18] (who also propose improvements in the 4-share setting) and in the very high order implementations of Journault and Standaert [JS17].

Our motivations in doing so are the following: since there is no known security proof at arbitrary order for these schemes, it is natural to try to prove them computationally at the highest possible order. Barthe *et al.* originally did this up to order 7,[11] and we manage to reach order 11 both for NI and SNI security, which represents a significant improvement.[12] A second motivation is that the verification of multiplication gadgets quickly becomes intractable with increasing order, and such a task allows us to clearly demonstrate our performance gain over maskVerif. Finally, this improved verification efficiency is exploited in trying to find *ad hoc* gadget variants with lower cost.

On the negative side our verification shows that a conjecture from Barthe *et al.* on the security of a natural strategy to convert NI multiplication into SNI fails at order 10. More positively, we were able to find *ad hoc* conversions tuned to every NI multiplication we considered, which sometimes also bring a significant improvement in randomness cost over Barthe *et al.*'s strategy. For instance we are able to gain 17% for an 8-share, 7-SNI gadget similar to the one used in [GPSS18]. Finally using a slight variant of Barthe *et al.*'s gadget generation algorithm, we occasionally obtain some improvements also in the NI case, notably at order 5.

We give details of our improvements in Table 5 in Appendix A, and the descriptions of all the gadgets at `https://github.com/NicsTr/binary_masking`.

The NI multiplication gadget family of [BDF+17, Algorithm 3]. We give in Algorithm 1 a description of a slightly modified variant of [BDF+17, Algorithm 3], which occasionally gives better gadgets than the original. We also provide a small script to automatically generate a scheme at a given order at `https://github.com/NicsTr/binary_masking`.

This description relies on the following convenient definition:

Definition 33 (Pair of shares).
Let $(\boldsymbol{a}_i\boldsymbol{b}_j)$, $i,j \in [\![0,d]\!]$ be the input shares of a $(d+1,v)$ gadget. We define $\hat{\boldsymbol{\alpha}}_{i,j}$ as:

$$\hat{\boldsymbol{\alpha}}_{i,j} = \begin{cases} \boldsymbol{a}_i\boldsymbol{b}_j & \text{if } i = j \\ \boldsymbol{a}_i\boldsymbol{b}_j + \boldsymbol{a}_j\boldsymbol{b}_i & \text{otherwise} \end{cases}$$

Extension to SNI security. One can derive an SNI multiplication gadget from Algorithm 1 by doing the following: 1) proving NI security at some order d; 2) proving SNI security at the same order for a *refreshing gadget*; 3) composing the two gadgets.

This strategy can for instance be implemented with the refreshing gadgets also introduced in [BDF+17] that we discuss in the next Section 5.2, but Barthe *et al.* already remarked that it was in fact apparently not necessary to use full refreshing gadgets and that one could do better by using a degraded variant thereof: in a nutshell, one starts from a secure NI multiplication and simply masks every output share with a fresh random mask and then again with the mask of the following share in a circular fashion. This is illustrated for 7-share gadgets in Figure 2.

Barthe *et al.* then conjecture in [BDF+17] that this transformation is always enough to convert an NI scheme into an SNI one. However we could check that this is not true for 11- and 12-share gadgets: the respective instantiations of Algorithm 1 are NI, but the transformation fails to provide SNI multiplications. Yet it is in fact still possible to derive an 11-share, 10-SNI multiplication gadget at no additional cost by simply rotating the last repeated masks by two positions instead of one, for a total cost of 44 random masks.

We explored several other transformation strategies, trying to exploit the special shape of the NI multiplication gadgets as much as possible. This almost always improved on the use of a new

[11] We ourselves used the latest version of maskVerif to do so up to order 8.
[12] This however still cannot theoretically justify the use of this masked multiplication at order 31 as is done in [JS17].

Algorithm 1: A conjectured d-NI $(d+1, d+1)$-gadget for multiplication over fields of characteristic two.

Input : $\mathcal{S} = \{\hat{\boldsymbol{\alpha}}_{i,j}, 0 \le i \le j \le d\}$
Input : $\mathcal{R} = \{\boldsymbol{r}_i\}$, $i \in \mathbb{N}$
Output: $(\boldsymbol{c}_i)_{0 \le i \le d}$, such that $\sum_{i=0}^{d} \boldsymbol{c}_i = \sum_{i=0}^{d} \boldsymbol{a}_i \sum_{i=0}^{d} \boldsymbol{b}_i$

for $i \leftarrow 0$ **to** d **do**
  $\boldsymbol{c}_i \leftarrow \hat{\boldsymbol{\alpha}}_{i,i}$
  $\mathcal{S} \leftarrow \mathcal{S} \setminus \{\hat{\boldsymbol{\alpha}}_{i,i}\}$
end
$\mathcal{R}' \leftarrow \{\}$
$j \leftarrow 1$
while $\mathcal{S} \neq \emptyset$ **do**
  for $i \leftarrow 0$ **to** d **do**
    if $j \equiv 1 \mod 2$ **then**
      $\boldsymbol{c}_i \leftarrow \boldsymbol{c}_i + \boldsymbol{r}_{\frac{(j-1)}{2}.(d+1)+i}$
      $\mathcal{R}' \leftarrow \mathcal{R}' \cup \left\{\boldsymbol{r}_{\frac{(j-1)}{2}.(d+1)+i}\right\}$
    else
      $\boldsymbol{c}_i \leftarrow \boldsymbol{c}_i + \boldsymbol{r}_{\frac{(j-2)}{2}.(d+1)+(i+1 \mod (d+1))}$
      $\mathcal{R}' \leftarrow \mathcal{R}' \setminus \left\{\boldsymbol{r}_{\frac{(j-2)}{2}.(d+1)+(i+1 \mod (d+1))}\right\}$
    end
    if $\mathcal{S} \neq \emptyset$ **then**
      $\boldsymbol{c}_i \leftarrow \boldsymbol{c}_i + \hat{\boldsymbol{\alpha}}_{i,((i+j) \mod (d+1))}$
      $\mathcal{S} \leftarrow \mathcal{S} \setminus \{\hat{\boldsymbol{\alpha}}_{i,((i+j) \mod (d+1))}\}$
    else
      break
    end
  end
  $j \leftarrow j+1$
end
$k \leftarrow \#\mathcal{R}'$
for $i \leftarrow 0$ **to** d **do**
  $\boldsymbol{c}_i \leftarrow \boldsymbol{c}_i + \boldsymbol{r}_{\frac{(j-1)}{2}(d+1)+(i+1 \mod k)}$
end

```
s00 r00 s01 s10 r01 s02 s20 r07 s03 s30 r08
s11 r01 s12 s21 r02 s13 s31 r08 s14 s41 r09
s22 r02 s23 s32 r03 s24 s42 r09 s25 s52 r10
s33 r03 s34 s43 r04 s35 s53 r10 s36 s63 r11
s44 r04 s45 s54 r05 s46 s64 r11 s40 s04 r12
s55 r05 s56 s65 r06 s50 s05 r12 s51 s15 r13
s66 r06 s60 s06 r00 s61 s16 r13 s62 s26 r07
```

(a) 6-NI multiplication, 14 random masks.

```
s00 r00 s01 s10 r01 s02 s20 r07 s03 s30 r08 r14 r20
s11 r01 s12 s21 r02 s13 s31 r08 s14 s41 r09 r15 r14
s22 r02 s23 s32 r03 s24 s42 r09 s25 s52 r10 r16 r15
s33 r03 s34 s43 r04 s35 s53 r10 s36 s63 r11 r17 r16
s44 r04 s45 s54 r05 s46 s64 r11 s40 s04 r12 r18 r17
s55 r05 s56 s65 r06 s50 s05 r12 s51 s15 r13 r19 r18
s66 r06 s60 s06 r00 s61 s16 r13 s62 s26 r07 r20 r19
```

(b) 6-SNI multiplication, 21 random masks (not optimal).

Fig. 2: 6-NI and SNI gadgets for multiplication.

mask for every share (the current exception being the order-8 gadget), usually requiring only about half. For instance our best 11-share gadget in fact only requires 39 masks instead of the above 44 as shown in Figure 3, and we found a 7-SNI multiplication with only 20 masks shown in Figure 4, which is 4 less than [BDF+17]. While this latter improvement is somewhat moderate at about 17%, this 8-share case is quite relevant due to its use in the efficient vectorised masked AES implementation of Grégoire *et al.* [GPSS18]; using our new variant should then result in a noticeable decrease in randomness usage.

We provide a summary of the cost of the multiplication gadgets that we have verified and their improvement over the previously best known ones in Appendix A, and we give their full description at https://github.com/NicsTr/binary_masking.

```
s00 r00 s01 s10 r01 s02 s20 r11 s03 s30 r12 s04 s40 r22 s05 s50 r23 r40
s11 r01 s12 s21 r02 s13 s31 r12 s14 s41 r13 s15 s51 r23 s16 s61 r24 r41
s22 r02 s23 s32 r03 s24 s42 r13 s25 s52 r14 s26 s62 r24 s27 s72 r25 r42
s33 r03 s34 s43 r04 s35 s53 r14 s36 s63 r15 s37 s73 r25 s38 s83 r26 r43
s44 r04 s45 s54 r05 s46 s64 r15 s47 s74 r16 s48 s84 r26 s49 s94 r27 r44
s55 r05 s56 s65 r06 s57 s75 r16 s58 s85 r17 s59 s95 r27 s5a sa5 r28 r45
s66 r06 s67 s76 r07 s68 s86 r17 s69 s96 r18 s6a sa6 r28 s60 s06 r29 r40
s77 r07 s78 s87 r08 s79 s97 r18 s7a sa7 r19 s70 s07 r29 s71 s17 r30 r41
s88 r08 s89 s98 r09 s8a sa8 r19 s80 s08 r20 s81 s18 r30 s82 s28 r31 r42
s99 r09 s9a sa9 r10 s90 s09 r20 s91 s19 r21 s92 s29 r31 s93 s39 r32 r43
saa r45 sa0 s0a r00 sa1 s1a r21 sa2 s2a r11 sa3 s3a r32 sa4 s4a r22 r44 r10
```

Fig. 3: 10-SNI gadget for multiplication, using 39 random masks.

```
s00 r00 s01 s10 r01 s02 s20 r08 s03 s30 r09 s04 r20
s11 r01 s12 s21 r02 s13 s31 r09 s14 s41 r10 s15 r21
s22 r02 s23 s32 r03 s24 s42 r10 s25 s52 r11 s26 r22
s33 r03 s34 s43 r04 s35 s53 r11 s36 s63 r12 s37 r23
s44 r04 s45 s54 r05 s46 s64 r12 s47 s74 r13 s40 r20
s55 r05 s56 s65 r06 s57 s75 r13 s50 s05 r14 s51 r21
s66 r06 s67 s76 r07 s60 s06 r14 s61 s16 r15 s62 r22
s77 r07 s70 s07 r00 s71 s17 r15 s72 s27 r08 s73 r23
```

Fig. 4: 7-SNI gadget for multiplication, using 20 random masks.

Verification performance. We now analyse the performance of our verification software on these multiplication schemes, and compare it with the one of the latest version of maskVerif [BBC⁺19].[13]

PROBES FILTERING. Following the results of Section 3.3, we use a filtering process to reduce the initial set of probes that one has to enumerate prove security. For the gadgets of Algorithm 1 and their SNI counterparts, this means removing probes of the form: $\hat{\boldsymbol{\alpha}}_{*,*} + \sum(\boldsymbol{r}_* + \hat{\boldsymbol{\alpha}}_{*,*}) + \boldsymbol{r}_* + \boldsymbol{a}_*\boldsymbol{b}_*$,[14] and the equipotency of the filtered set and the original one is verified by an exhaustive check on the subsets corresponding to every output share. Intuitively, the idea is that one can always replace in an attack a probe of the above form with one that includes one extra $\boldsymbol{a}_j\boldsymbol{b}_i$ term, *i.e.* one of the form $\hat{\boldsymbol{\alpha}}_{*,*} + \sum(\boldsymbol{r}_* + \hat{\boldsymbol{\alpha}}_{*,*}) + \boldsymbol{r}_* + \hat{\boldsymbol{\alpha}}_{*,*}$, since the latter only adds an additional functional dependence on the input shares “for free”.

The concrete impact of filtering on the verification performance of our schemes can be seen in Table 1, where we give the size of the attack sets to enumerate before and after this filtering.

PERFORMANCE. For order $d \leq 10$ (except the 10-SNI case) we have run our software on a single core of the `retourdest` server, which features a single Intel Xeon Gold 6126 at 2.60 GHz. The corresponding timings are given in Table 1. At peak performance, we are able to enumerate $\approx 2^{27.5}$ candidate attack sets per second for NI verification, while SNI performance is slightly worse.

Using filtered sets significantly improves verification time, especially at high order. For instance, the running times of 2 and 6 hours for NI and SNI multiplication at order 9 are an order of magnitude faster than the 3 and 6 days initially spent before we implemented filtering. This optimisation was also essential in allowing to check the security of 10-NI multiplication in less than one calendar day on a single machine (using parallelisation); it would otherwise have taken a rather costly 1 core-year.

[13] Available at https://gitlab.com/benjgregoire/maskverif.

[14] This corresponds exactly to the probes made of an even number of $\boldsymbol{a}_*\boldsymbol{b}_*$ terms.

We also tested a multi-threaded implementation of our software on schemes at order $8 \sim 10$, using all 12 physical cores of the same Xeon Gold 6126; the results are shown in Table 2. While we do not have many data points, the speed-up offered by the parallelisation seems to be close to linear, albeit slightly less for NI verification: the 9-SNI multi-threaded wall time is ≈ 11.7 times less than the single-threaded one, and multi-threading for 9- and 10-NI saves a factor ≈ 9.7.

The largest instances that we verified are NI (resp. SNI) multiplication at order $d = 11$, which represents a significant computation. We relied heavily on parallelisation to enumerate the $\approx 2^{52.72}$ (resp. $\approx 2^{54.48}$) possible attack sets,[15] using up to 40 nodes of the *Dahu* cluster.[16] Each node has two 16-core Intel Xeon Gold 6130 at 2.10 GHz, and when using hyperthreading allows to enumerate $\approx 2^{31.38}$ sets per second. This cluster was also used to verify the best version of our 10-SNI gadget.

Comparison with maskVerif. We used the maskVerif tool from Barthe *et al.* [BBC+19] to check the security of the gadgets at order 6 to 8. Due to system constraints, we could not run the verification on `retourdest`, and instead defaulted to the older `hpac`, which features an Intel Xeon E5-4620 at 2.20 GHz. Yet since the maskVerif implementation is parallel and may use up to four threads, we believe that comparison with our single-threaded performance is nonetheless meaningful.

The running times are summarised in Table 3. Comparing with Table 1, it is notable that our own software is faster by three orders of magnitude, for instance taking one minute to check 8-NI multiplication *versus* two days for maskVerif.

Table 1: Running time of our verification software (sequential).

Order d		$\log_2$(number of sets) Before/After filtering	Wall time Best (after filtering)
1	NI	2.6/2.6	$<$ 0.01 sec.
	SNI	2.6/2.6	$<$ 0.01 sec.
2	NI	6.3/5.5	$<$ 0.01 sec.
	SNI	6.3/5.5	$<$ 0.01 sec.
3	NI	10.4/8.9	$<$ 0.01 sec.
	SNI	11.2/9.96	$<$ 0.01 sec.
4	NI	15.0/12.6	$<$ 0.01 sec.
	SNI	16.4/14.6	$<$ 0.01 sec.
5	NI	21.2/18.6	$<$ 0.01 sec.
	SNI	21.7/19.3	$<$ 0.01 sec.
6	NI	27.1/23.9	0.09 sec.
	SNI	28.0/25.3	0.28 sec.
7	NI	32.7/28.7	2.43 sec.
	SNI	33.6/30.6	11.70 sec.
8	NI	38.5/33.7	1 min. 17 sec.
	SNI	40.3/36.3	9 min. 28 sec.
9	NI	45.6/40.5	2 h. 18 min.
	SNI	46.3/41.6	6 h. 30 min.
10	NI	52.6/47.1	9 days 3h.
	SNI	53.5/48.4	—

5.2 SNI refreshing gadgets

We used our software to verify the SNI security of some (variations of) refreshing gadgets introduced in [BDF+17], and subsequently improved in [GPSS18,BBD+18]. Such schemes are useful when

[15] This is after filtering of the initial $\approx 2^{59}$ (resp. $\approx 2^{59.76}$) sets.
[16] https://ciment.univ-grenoble-alpes.fr/wiki-pub/index.php/Hardware:Dahu

Table 2: Running time of our verification software (parallel, 12 threads).

Order d		Wall time
8	NI	7.43 sec.
	SNI	47.0 sec.
9	NI	14 min. 20 sec.
	SNI	33 min. 20 sec.
10	NI	22 h. 30 min.
	SNI	—

Table 3: Running time of maskVerif (parallel, up to 4 threads) [BBC+19].

Order d		Wall time
6	NI	2 min. 44 sec.
	SNI	8 min. 11 sec.
7	NI	1 h. 39 min.
	SNI	5 h. 54 min.
8	NI	2 days 10h.
	SNI	13 days 6h.

designing large circuits based on gadgets satisfying composable security notions since they help in providing strong security for the overall design. However, refreshing also comes with a significant cost in terms of randomness while not performing any sort of useful computation, leading several prior work to try finding new low-complexity gadgets.

The best current results come from [BBD+18] who prove the SNI security at any order of a "block" refreshing gadget introduced in [BDF+17], when iterated enough times. Yet together with [GPSS18], they also remark that it is possible to make significant improvements in practice at the cost of losing generic proofs, and they give cheaper alternatives verified secure up to order 16.

Our contribution here is an 8-share, 7-SNI refreshing gadget shown in Figure 5 that only needs 13 masks, which improves slightly on the best gadget from [BBD+18], which requires 16. Since such gadgets are used in the implementation of [GPSS18], it could again lead to actual practical gains.

We also compared the verification time of our tool with the one of maskVerif on the largest "RefreshZero" instances of [BBD+18], and actually have worse performance. For instance, even using 24 threads on the 12-core `retourdest`, verifying $\mathsf{RefreshZero}^{14}_{[1,3]}$ took us about 3 hours 40 minutes, while [BBD+18] reports an "Order of Magnitude" of 1 hour 30 minutes. We suspect this to be caused by the fact that there is no obvious probe filtering to be done on this sort of gadget, whereas maskVerif is likely able to successfully exploit their structure to reduce the number of attack sets to consider.

```
s00 r00 r01 r10 r20
s11 r01 r02 r11 r20
s22 r02 r03 r12 r20
s33 r03 r04 r13 r20
s44 r04 r05 r10
s55 r05 r06 r11
s66 r06 r07 r12
s77 r07 r00 r13
```

Fig. 5: 7-SNI refreshing gadget, using 13 random masks..

5.3 Glitch-resistant NI multiplication

We conclude with a brief application to the *DOM-indep* family of multiplication gadgets introduced by Groß *et al.* [GMK16]. While those schemes are not more efficient than the state-of-the-art in terms of randomness complexity, their main advantage is their resistance to glitches. A description of an instantiation at order 2 can be found in Figure 1, and at any order less than 7 in the supporting material.

These gadgets can be instantiated at an arbitrary order d but do not come with a generic security proof guaranteeing the security of the result. We then have used our implementation to verify that instantiations up to order 7 are NI in the robust probing model. The verification at order 7 takes less than 2 hours on a single core of `retourdest`. The running times are summarised in Table 4.

Table 4: Running time of our verification software on the *DOM-indep* schemes (sequential).

Order d	Wall time
1	$<$ 0.01 sec.
2	$<$ 0.01 sec.
3	$<$ 0.01 sec.
4	$<$ 0.01 sec.
5	0.75 sec.
6	1 min. 04 sec.
7	1 h. 57 min.

Acknowledgments

We thank Clément Pernet for his part in the proof of Lemma 14, Yann Rotella for an early discussion on the possibility of further filtering, and the authors of [BBC+19] for providing us access to an up-to-date version of maskVerif.

This work is partially supported by the French National Research Agency in the framework of the *Investissements d'avenir* programme (ANR-15-IDEX-02).

Some of the computations presented in this paper were performed using the GRICAD infrastructure (`https://gricad.univ-grenoble-alpes.fr`), which is partially supported by the Equip@Meso project (ANR-10-EQPX-29-01) of the *Investissements d'Avenir* programme.

References

BBC+19. Gilles Barthe, Sonia Belaïd, Gaëtan Cassiers, Pierre-Alain Fouque, Benjamin Grégoire, and François-Xavier Standaert. maskVerif: Automated Verification of Higher-Order Masking in Presence of Physical Defaults. In Kazue Sako, Steve Schneider, and Peter Y. A. Ryan, editors, *ESORICS 2019*, volume 11735 of *Lecture Notes in Computer Science*, pages 300–318. Springer, 2019.

BBD+15. Gilles Barthe, Sonia Belaïd, François Dupressoir, Pierre-Alain Fouque, and Benjamin Grégoire. Compositional Verification of Higher-Order Masking: Application to a Verifying Masking Compiler. *IACR Cryptology ePrint Archive*, 2015:506, 2015.

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, *ACM CCS 2016*, pages 116–129. ACM, 2016.

BBD+18. Gilles Barthe, Sonia Belaïd, François Dupressoir, Pierre-Alain Fouque, Benjamin Grégoire, François-Xavier Standaert, and Pierre-Yves Strub. Improved Parallel Mask Refreshing Algorithms: Generic Solutions with Parametrized Non-Interference & Automated Optimizations. *IACR Cryptology ePrint Archive*, 2018:505, 2018.

BBP+16. Sonia Belaïd, Fabrice Benhamouda, Alain Passelègue, Emmanuel Prouff, Adrian Thillard, and Damien Vergnaud. Randomness Complexity of Private Circuits for Multiplication. In Marc Fischlin and Jean-Sébastien Coron, editors, *EUROCRYPT 2016*, volume 9666 of *Lecture Notes in Computer Science*, pages 616–648. Springer, 2016.

BBP+17. Sonia Belaïd, Fabrice Benhamouda, Alain Passelègue, Emmanuel Prouff, Adrian Thillard, and Damien Vergnaud. Private Multiplication over Finite Fields. In Jonathan Katz and Hovav Shacham, editors, *CRYPTO 2017*, volume 10403 of *Lecture Notes in Computer Science*, pages 397–426. Springer, 2017.

BDF+17. Gilles Barthe, François Dupressoir, Sebastian Faust, Benjamin Grégoire, François-Xavier Standaert, and Pierre-Yves Strub. Parallel Implementations of Masking Schemes and the Bounded Moment Leakage Model. In Coron and Nielsen [CN17], pages 535–566.

BGI+18. Roderick Bloem, Hannes Groß, Rinat Iusupov, Bettina Könighofer, Stefan Mangard, and Johannes Winter. Formal Verification of Masked Hardware Implementations in the Presence of Glitches. In Jesper Buus Nielsen and Vincent Rijmen, editors, *EUROCRYPT 2018*, volume 10821 of *Lecture Notes in Computer Science*, pages 321–353. Springer, 2018.

BS19. Olivier Bronchain and François-Xavier Standaert. Side-Channel Countermeasures' Dissection and the Limits of Closed Source Security Evaluations. *IACR Cryptology ePrint Archive*, 2019:1008, 2019.

CN17. Jean-Sébastien Coron and Jesper Buus Nielsen, editors. *Advances in Cryptology — EUROCRYPT 2017*, volume 10210 of *Lecture Notes in Computer Science*, 2017.

DFS15. Alexandre Duc, Sebastian Faust, and François-Xavier Standaert. Making Masking Security Proofs Concrete - Or How to Evaluate the Security of Any Leaking Device. In Elisabeth Oswald and Marc Fischlin, editors, *EUROCRYPT 2015*, volume 9056 of *Lecture Notes in Computer Science*, pages 401–429. Springer, 2015.

FG18. Junfeng Fan and Benedikt Gierlichs, editors. *Constructive Side-Channel Analysis and Secure Design — COSADE 2018*, volume 10815 of *Lecture Notes in Computer Science*. Springer, 2018.

FGP+18. Sebastian Faust, Vincent Grosso, Santos Merino Del Pozo, Clara Paglialonga, and François-Xavier Standaert. Composable Masking Schemes in the Presence of Physical Defaults & the Robust Probing Model. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2018(3):89–120, 2018.

GJRS18. Dahmun Goudarzi, Anthony Journault, Matthieu Rivain, and François-Xavier Standaert. Secure Multiplication for Bitslice Higher-Order Masking: Optimisation and Comparison. In Fan and Gierlichs [FG18], pages 3–22.

GMK16. Hannes Groß, Stefan Mangard, and Thomas Korak. Domain-Oriented Masking: Compact Masked Hardware Implementations with Arbitrary Protection Order. In Begül Bilgin, Svetla Nikova, and Vincent Rijmen, editors, *ACM TIS@CCS 2016*, page 3. ACM, 2016.

GPSS18. Benjamin Grégoire, Kostas Papagiannopoulos, Peter Schwabe, and Ko Stoffelen. Vectorizing Higher-Order Masking. In Fan and Gierlichs [FG18], pages 23–43.

GR17. Dahmun Goudarzi and Matthieu Rivain. How Fast Can Higher-Order Masking Be in Software? In Coron and Nielsen [CN17], pages 567–597.

ISW03. Yuval Ishai, Amit Sahai, and David A. Wagner. Private Circuits: Securing Hardware against Probing Attacks. In Dan Boneh, editor, *CRYPTO 2003*, volume 2729 of *Lecture Notes in Computer Science*, pages 463–481. Springer, 2003.

JS17. Anthony Journault and François-Xavier Standaert. Very High Order Masking: Efficient Implementation and Security Evaluation. In Wieland Fischer and Naofumi Homma, editors, *CHES 2017*, volume 10529 of *Lecture Notes in Computer Science*, pages 623–643. Springer, 2017.

KJJ99. Paul C. Kocher, Joshua Jaffe, and Benjamin Jun. Differential Power Analysis. In Michael J. Wiener, editor, *CRYPTO '99*, volume 1666 of *Lecture Notes in Computer Science*, pages 388–397. Springer, 1999.

Knu11. Donald E. Knuth. *Combinatorial Algorithms, Part 1*, volume 4A of *The Art of Computer Programming*. Addison Wesley, 2011.

KR18. Pierre Karpman and Daniel S. Roche. New Instantiations of the CRYPTO 2017 Masking Schemes. In Thomas Peyrin and Steven D. Galbraith, editors, *ASIACRYPT 2018*, volume 11273 of *Lecture Notes in Computer Science*, pages 285–314. Springer, 2018.

LT73. C. N. Liu and Donald T. Tang. Enumerating Combinations of m Out of n Objects [G6] (Algorithm 452). *Commun. ACM*, 16(8):485, 1973.

MMSS19. Thorben Moos, Amir Moradi, Tobias Schneider, and François-Xavier Standaert. Glitch-Resistant Masking Revisited or Why Proofs in the Robust Probing Model are Needed. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2019(2):256–292, 2019.

NW78. Albert Nijenhuis and Herbert S. Wilf. *Combinatorial algorithms for computers and calculators*. Academic Press, second edition, 1978.

Pra62. Eugene Prange. The use of information sets in decoding cyclic codes. *IRE Trans. Information Theory*, 8(5):5–9, 1962.

Sch80. Jacob T. Schwartz. Fast Probabilistic Algorithms for Verification of Polynomial Identities. *J. ACM*, 27(4):701–717, 1980.

Wal. Timothy R. Walsh. A simple sequencing and ranking method that works on almost all gray codes. Unpublished research report. Available at: https://www.labunix.uqam.ca/~walsh_t/papers/sequencing_and_ranking.pdf.

A Explicit randomness cost of multiplication gadgets

We give here the randomness cost for multiplication gadgets that were verified to be NI or SNI by Barthe *et al.* in [BDF+17], and by us in Section 5. Note that Belaïd *et al.* also propose optimized gadgets in [BBP+16] up to order 4, that ISW is also better than [BDF+17] at order 3 and that Grégoire *et al.* already proposed improvements at this same order in [GPSS18]. The main range of interest of this table is thus at order 5 and beyond.

Table 5: Explicit randomness cost of multiplication gadgets

Order d		Defined *and* verified in [BDF+17]	Defined *or* verified in §5
		Random masks	Random masks
2	SNI	3	=
3	NI	4	=
	SNI	8	5
4	NI	5	=
	SNI	10	9
5	NI	12	10
	SNI	18	12
6	NI	14	=
	SNI	21	18
7	NI	—	16
	SNI	24	20
8	NI	—	18
	SNI	—	27
9	NI	—	26
	SNI	—	30
10	NI	—	33
	SNI	—	39
11	NI	—	36
	SNI	—	42