Faster Evaluation of SBoxes via Common Shares*

Jean-Sébastien Coron[1], Aurélien Greuet[2], Emmanuel Prouff[3]**, and Rina Zeitoun[2]

[1] University of Luxembourg
jean-sebastien.coron@uni.lu
[2] Oberthur Technologies, France
{r.zeitoun,a.greuet}@oberthur.com
[3] UPMC University Paris 06, Team POLSYS, France

Abstract. We describe a new technique for improving the efficiency of the masking countermeasure against side-channel attacks. Our technique is based on using common shares between secret variables, in order to reduce the number of finite field multiplications. Our algorithms are proven secure in the ISW probing model with $n \geqslant t+1$ shares against t probes. For AES, we get an equivalent of 2.8 non-linear multiplications for every SBox evaluation, instead of 4 in the Rivain-Prouff countermeasure. We obtain similar improvements for other block-ciphers. Our technique is easy to implement and performs relatively well in practice, with roughly a 20% speed-up compared to existing algorithms.

1 Introduction

Side-Channel Attacks. Side-channel analysis is a class of cryptanalytic attacks that exploit the physical environment of a cryptosystem to recover some *leakage* about its secrets. It is often more efficient than a cryptanalysis mounted in the so-called *black-box model* where no leakage occurs. In particular, *continuous side-channel attacks* in which the adversary gets information at each invocation of the cryptosystem are especially threatening. Common attacks as those exploiting the running-time, the power consumption or the electromagnetic radiations of a cryptographic computation fall into this class. Many implementations of block ciphers have been practically broken by continuous side-channel analysis — see for instance [KJJ99, BCO04, Mes00, MPO05] —and securing them has been a longstanding issue for the embedded systems industry.

The Masking Countermeasure. A sound approach to counteract side-channel attacks is to use *secret sharing* [Bla79, Sha79], often called *masking* in the context of side-channel attacks. This approach consists in splitting each sensitive variable x of the implementation into n shares such that $x = x_1 \oplus \cdots \oplus x_n$, where n is called the *sharing order*, such that x can be recovered from these shares but no information can be recovered from fewer than n shares. It has been shown that the complexity of mounting a successful side-channel attack against a masked implementation increases exponentially with the order [CJRR99, PR13, DDF14]. Starting from this observation, the design of efficient *secure schemes* for different ciphers has become a foreground issue. When specified *at order* n, such a scheme aims at specifying how to update the sharing of the internal state throughout the processing while ensuring that (1) the final sharing corresponds to the expected ciphertext, and (2) the n-th order security property is satisfied.

* An extended abstract will appear at CHES 2016; this is the full version.
** Part of this work has been done at Safran Identity and Security, and while the author was at ANSSI, France.

The ISW Probing Model. Ishai, Sahai and Wagner [ISW03] initiated the theoretical study of securing circuits against an adversary who can probe a fraction of its wires. They showed how to transform any circuit of size $|C|$ into a circuit of size $\mathcal{O}(|C| \cdot t^2)$ secure against any adversary who can probe at most t wires. The ISW constructions consists in secret-sharing every variable x into $x = x_1 \oplus x_2 \oplus \cdots \oplus x_n$ where $x_2, \ldots, x_n$ are uniformly and independently distributed bits, with $n \geqslant 2t+1$ to get security against t probes. Processing a XOR gate is straightforward as the shares can be xored separately. The processing of an AND gate $z = xy$ is based on writing:

$$z = xy = \left(\bigoplus_{i=1}^{n} x_i\right) \cdot \left(\bigoplus_{i=1}^{n} y_i\right) = \bigoplus_{1 \leqslant i,j \leqslant n} x_i y_j \tag{1}$$

where the cross-products $x_i y_j$ are first computed and then randomly recombined to get an n-sharing of the output z. This construction, called *ISW gadget* in the rest of this paper, enables, in its general form, to securely evaluate a multiplication at the cost of n^2 multiplications, $2n(n-1)$ additions and $n(n-1)/2$ random values. Its complexity is therefore $\mathcal{O}(n^2)$, which implies that the new circuit with security against t probes has $\mathcal{O}(|C| \cdot t^2)$ gates.

A proof of security in the ISW framework is usually simulation based: one must show that any set of t probes can be perfectly simulated without the knowledge of the original variables of the circuit. In [ISW03] and subsequent work this is done by progressively generating a subset I of input shares such that the knowledge of those input shares is sufficient to simulate all the t probes. For example, in the above AND gate, if the adversary would probe $x_i \cdot y_j$, one would put both indices i and j in I, so that the simulator would get the input shares x_i and y_j, and therefore could simulate the product $x_i \cdot y_j$. More generally in the ISW construction every probe adds at most two indices in I, which implies $|I| \leqslant 2t$. Therefore if the number of shares n is such that $n \geqslant 2t+1$, then $|I| < n$, which implies that only a proper subset of the input shares is required for the simulation; those input shares can in turn be generated as independently uniformly distributed bits. Therefore, the knowledge of the original circuit variables is not required to generate a perfect simulation of the t probes, hence these probes do not bring any additional information to the attacker (since he could perform that simulation by himself).

Existing work. In the last decade, several masking countermeasures have been proposed for block-ciphers together with security proofs in the ISW probing model (see *e.g.* [CGP+12a, Cor14, CPRR13, GPQ11, ISW03, RP10, RV13]), based on the original notion of private circuits introduced in [ISW03]. Except [Cor14] which extends the original idea of [KJJ99] to any order, the other proposals are based on the ISW gadget recalled above. The core idea of the latter works is to split the processing into a short sequence of field multiplications and $\mathbb{F}_2$-linear operations, and then to secure these operations independently, while ensuring that the local security proofs can be combined to prove the security of the entire processing. When parametrized at order n, as recalled above the complexity of the ISW gadget for the field multiplication is $\mathcal{O}(n^2)$, but only $\mathcal{O}(n)$ for $\mathbb{F}_2$-linear operations.[1] Therefore, an interesting problem is to minimize the number of field multiplications required to evaluate an SBox.

In the Rivain-Prouff countermeasure [RP10], the authors showed how to adapt the ISW circuit construction to a software implementation of AES, by working in $\mathbb{F}_{2^8}$ instead of $\mathbb{F}_2$. Namely as

[1] A function f is $\mathbb{F}_2$-linear if it satisfies $f(x \oplus y) = f(x) \oplus f(y)$ for any pair (x, y) of elements in its domain. This property must not be confused with $\mathbb{F}_{2^m}$-linearity of a function, where m divides n and is larger than 1, which is defined such that $f(ax \oplus by) = af(x) \oplus bf(y)$, for every $a, b \in \mathbb{F}_{2^m}$. An $\mathbb{F}_{2^m}$-linear function is $\mathbb{F}_2$-linear but the converse is false in general.

illustrated in Fig. 1, the non-linear part $S(x) = x^{254}$ of the AES SBox can be evaluated with only 4 non-linear multiplications over $\mathbb{F}_{2^8}$, and a few linear squarings. Each of those 4 multiplications can in turn be evaluated with the previous ISW gadget based on Equation (1), by working over $\mathbb{F}_{2^8}$ instead of $\mathbb{F}_2$.

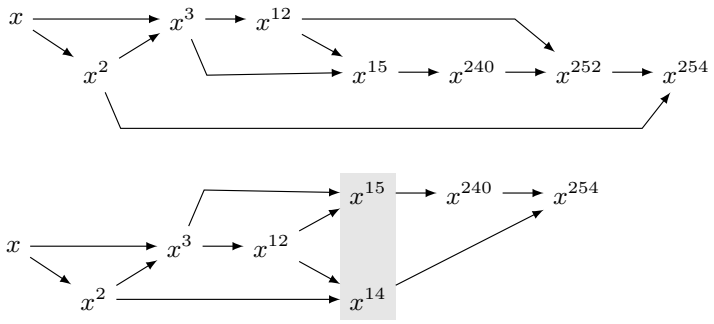

Fig. 1. a) Sequential computation of x^{254} as used in [RP10, BBD+15a]. b) Alternative computation of x^{254}; the multiplications $x^{14} = x^{12} \cdot x^2$ and $x^{15} = x^{12} \cdot x^3$ can be computed in parallel [GHS12].

The Rivain-Prouff countermeasure was later extended by Carlet *et al.* to any look-up table [CGP+12a]. Namely any given k-bit SBox can be represented by a polynomial $\sum_{i=0}^{2^k-1} a_i\, x^i$ over $\mathbb{F}_{2^k}$ using Lagrange's interpolation theorem. Therefore one can mask any SBox by securely evaluating this polynomial using n-shared multiplications as in the Rivain-Prouff countermeasure. To improve efficiency, one must look for operations sequences (*e.g.* SBox representations) that minimize the number of field multiplications which are not $\mathbb{F}_2$-linear [2] (this kind of multiplication shall be called *non-linear* in this paper). This problematic has been tackled out in [CGP+12a], [RV13] and [CRV14] and led to significantly reduce the number of multiplications needed to evaluate any function defined over $\mathbb{F}_{2^k}$ for $k \leqslant 10$ (*e.g.* the AES SBox can be evaluated with only 4 multiplications, and only 4 multiplications are needed for the DES SBoxes).

Recently, a sequence of works continued to improve the original work [ISW03] and led, in particular, to exhibit a new scheme enabling to securely evaluate any function of algebraic degree 2 at the cost of a single multiplication (with the ISW gadget). The application of this work to the AES SBox led the authors of [GPS14] to describe a scheme which can be secure at any order n and is a valuable alternative to the scheme proposed in [RP10]. In parallel, some schemes [BGN+14, NRS11, PR11] have been proposed which remain secure in the probing model even in presence of so-called glitches [MS06] and the recent work [RBN+15] has investigated relations between these schemes and the ISW construction.

Refined Security Model: *t*-SNI Security. Since in this paper we are interested in efficiency improvements, we would like to use the optimal $n = t + 1$ number of shares instead of $n = 2t + 1$ as in the original ISW countermeasure. For $n \geqslant 2t + 1$ shares the security proof for the single ISW multiplication gadget easily extends to the full circuit [ISW03]; however for $n \geqslant t + 1$ shares only one must be extra careful. For example, for the Rivain-Prouff countermeasure, it was originally

[2] A multiplication over a field of characteristic 2 is $\mathbb{F}_2$-linear if it corresponds to a Frobenius automorphism, *i.e.* to a series of squarings.

claimed in [RP10] that only $n \geqslant t+1$ shares were required, but an attack of order $\lceil (n-1)/2 \rceil + 1$ was later described in [CPRR13]; the security proof in [RP10] with $n \geqslant t+1$ shares actually applies only when the ISW multiplication is used in isolation, but not for the full block-cipher.

To prove security with $n \geqslant t+1$ shares only for the full block-cipher, a refined security model against probing attacks was recently introduced in [BBD+15a], called t-SNI security. As shown in [BBD+15a], this stronger definition of t-SNI security enables to prove that a gadget can be used in a full construction with $n \geqslant t+1$ shares, instead of $n \geqslant 2t+1$ for the weaker definition of t-NI security (corresponding to the original ISW security proof). The authors show that the ISW multiplication gadget does satisfy this stronger t-SNI security definition. They also show that with some additional mask refreshing, the Rivain-Prouff countermeasure for the full AES can be made secure with $n \geqslant t+1$ shares. Due to its power and simplicity, the t-SNI notion appears to be the "right" security definition against probing attacks. Therefore, in this paper, we always prove the security of our algorithms under this stronger t-SNI notion, so that our algorithms can be used within a larger construction (typically a full block-cipher) with $n \geqslant t+1$ shares only.

Our Contribution. Our goal in this paper is to further improve the efficiency of the masking countermeasure. As recalled above, until now the strategy followed by the community has been to reduce the number of calls to the ISW multiplication gadget. In this paper, we follow a complementary approach consisting in reducing the complexity of the ISW multiplication gadget itself. Our core idea is to use common shares between the inputs of multiple ISW multiplication gadgets, up to the first $n/2$ shares; in that case, a given processing performed in the first ISW gadget can be re-used in subsequent gadgets.

Consider for example the alternative evaluation circuit for x^{254} in AES used in [GHS12], as illustrated in Fig. 1. It still has 4 non-linear multiplications as in the original circuit [RP10], but now the two multiplications $x^{14} \leftarrow x^{12} \cdot x^2$ and $x^{15} \leftarrow x^{12} \cdot x^3$ can be evaluated in parallel, moreover with a common operand x^{12}. Let denote by $d \leftarrow c \cdot a$ and $e \leftarrow c \cdot b$ those two multiplications with common operand c. In the ISW multiplication gadget, one must compute all cross-products $c_i \cdot a_j$ and $c_i \cdot b_j$ for all $1 \leqslant i, j \leqslant n$. Now if we can ensure that half of the shares of a and b are the same, that is $a_j = b_j$ for all $1 \leqslant j \leqslant n/2$, then the products $c_i \cdot a_j$ and $c_i \cdot b_j$ for $1 \leqslant j \leqslant n/2$ are the same and can be computed only once; see Fig. 2 for an illustration. This implies that when processing the second multiplication gadget for $e \leftarrow c \cdot b$, we only have to compute $n^2/2$ finite field multiplications instead of n^2. For two multiplications as above, this saves the equivalent of 0.5 multiplication.

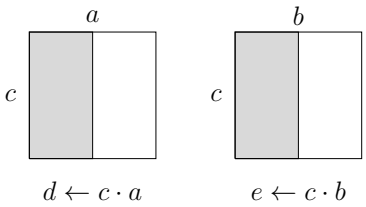

Fig. 2. When half of the shares in a and b are the same, the multiplications corresponding to the left-hand blocks are the same. This saves the equivalent of 0.5 multiplications out of 2.

To ensure that the two inputs have half of their shares in common, we introduce a new gadget called CommonShares with complexity $\mathcal{O}(n)$, taking as input two independent n-sharings of data and outputting two new n-sharings, but with their first $n/2$ shares in common. Obviously this must

be achieved without degrading the security level; we show that this is indeed the case by proving the security of the full SBox evaluation in the previous t-SNI model, with $n \geqslant t+1$ shares. Note that we cannot have more than $n/2$ shares in common between two variables a and b, since otherwise there would be a straightforward attack with fewer than n probes: namely if $a_i = b_i$ for all $1 \leqslant i \leqslant k$, then we can probe the $2(n-k)$ remaining shares a_i and b_i for $k+1 \leqslant i \leqslant n$; if $k > n/2$ this gives strictly less than n shares, whose xor gives the secret variable $a \oplus b$. Hence having half of the shares in common is optimal.

More generally, the 16 SBoxes of AES can be processed in parallel, and therefore each of the 4 non-linear multiplications in x^{254} can be processed in parallel. As opposed to the previous case those multiplications do not share any operand, but we show that by using a generalization of the CommonShares algorithm between m operands instead of 2, for every multiplication in the original circuit one can still save the equivalent of roughly $1/4$ multiplication. This also applies to other block-ciphers as well, since in most block-ciphers the SBoxes are applied in parallel. One can therefore apply the technique from [CRV14] based on fast polynomial evaluation, and using our CommonShares algorithm between the inputs of the evaluated polynomials, we again save roughly $1/4$ of the number of finite field multiplications. Our results for various block-ciphers are summarized in Table 1, in which we give the equivalent number of non-linear multiplications for a single SBox evaluation, for various block-ciphers; we refer to Section 5 for a detailed description. Finally, we show in Appendix G how to apply our common shares technique to the Threshold Implementations (TI) approach for securing implementation against side channel attacks, even in the presence of glitches.

Methods	SBox					
	AES	DES	PRESENT	SERPENT	CAMELLIA	CLEFIA
Parity-Split [CGP+12a]	4	10	3	3	22	22
Roy-Vivek [RV13]	4	7	3	3	15	15,16
[CRV14]	4	4	2	2	10	10
Our Method	**2.8**	**3.1**	**1.5**	**1.5**	**7.8**	**7.8**

Table 1. Equivalent number of non-linear multiplications for a single SBox evaluation, for various block-ciphers.

Practical Implementation. A practical implementation of our common shares technique is described in Section 7, for the n-shared evaluation of x^{254} in AES, on ATmega1284P (8-bit AVR microcontroller) and ARM Cortex M0 (32-bit CPU). We obtain that our technique is relatively practical: for a large number of shares, we get roughly a 20% speed improvement compared to the Rivain-Prouff countermeasure (but only roughly 5% compared to the quadratic evaluation technique in [GPS14]).

2 Security Definitions

Given a variable $x \in \mathbb{F}_{2^k}$ and an integer n, we say that the vector $(x_1, \ldots, x_n) \in (\mathbb{F}_{2^k})^n$ is an n-sharing of x if $x = \bigoplus_{i=1}^{n} x_i$. We recall the security definitions from [BBD+15a], which we make slightly more explicit. For simplicity we only provide the definitions for a simple gadget taking as input a single variable x (given by n shares x_i) and outputting a single variable y (given by n shares y_i). We provide the generalization to multiple inputs and outputs in Appendix A. Given a vector $(x_i)_{1 \leqslant i \leqslant n}$, we denote by $x_{|I} := (x_i)_{i \in I}$ the sub-vector of shares x_i with $i \in I$.

Definition 1 (t-NI security). *Let G be a gadget taking as input $(x_i)_{1\leqslant i\leqslant n}$ and outputting $(y_i)_{1\leqslant i\leqslant n}$. The gadget G is said t-NI secure if for any set of t_1 intermediate variables and any subset $\mathcal{O}$ of output indices, there exists a subset I of input indices with $|I| \leqslant t_1 + |\mathcal{O}|$, such that the t_1 intermediate variables and the output variables $y_{|\mathcal{O}}$ can be perfectly simulated from $x_{|I}$.*

Definition 2 (t-SNI security). *Let G be a gadget taking as input $(x_i)_{1\leqslant i\leqslant n}$ and outputting $(y_i)_{1\leqslant i\leqslant n}$. The gadget G is said t-SNI secure if for any set of t_1 intermediate variables and any subset $\mathcal{O}$ of output indices such that $t_1 + |\mathcal{O}| \leqslant t$, there exists a subset I of input indices with $|I| \leqslant t_1$, such that the t_1 intermediate variables and the output variables $y_{|\mathcal{O}}$ can be perfectly simulated from $x_{|I}$.*

The t-NI security notion corresponds to the original security definition in the ISW probing model; it allows to prove the security of a full construction with $n \geqslant 2t+1$ shares. The stronger t-SNI notion allows to prove the security of a full construction with $n \geqslant t+1$ shares only [BBD+15a]. The difference is that in the stronger t-SNI notion, the size of the input shares subset I can only depend on the number of internal probes t_1, and must be independent of the number of output variables $|\mathcal{O}|$ that must be simulated (as long as the condition $t_1 + |\mathcal{O}| \leqslant t$ is satisfied). Intuitively, this provides an "isolation" between the output shares and the input shares of a given gadget, and for composed constructions this enables to easily prove that a full construction is t-SNI secure, based on the t-SNI security of its components.

3 The Rivain-Prouff Countermeasure

In this section we recall the Rivain-Prouff countermeasure [RP10] for securing AES against high-order attacks. It can be seen as an extension to $\mathbb{F}_{2^k}$ of the original ISW countermeasure [ISW03] described in $\mathbb{F}_2$. The Rivain-Prouff countermeasure is proved t-SNI secure in [BBD+15a]; therefore it can be used to protect a full block-cipher against t probes with $n \geqslant t+1$ shares, instead of $n \geqslant 2t+1$ shares in the original ISW probing model.

3.1 The Rivain-Prouff Multiplication

The Rivain-Prouff countermeasure is based on the SecMult operation below, which is similar to the ISW multiplication gadget but over $\mathbb{F}_{2^k}$ instead of $\mathbb{F}_2$. The SecMult algorithm enables to securely compute a product $c = a \cdot b$ over $\mathbb{F}_{2^k}$, from an n-sharing of a and b, and outputs an n-sharing of c. Here we use the linear memory version from [Cor14], using similar notations as in [BBD+15a].

It is shown in [BBD+15a] that the SecMult algorithm is t-SNI secure with $n \geqslant t+1$ shares. For completeness we provide a proof of Lemma 1 in Appendix B.1; our proof is essentially the same as in [BBD+15a]. In Appendix B.2, we also provide a slightly different, more modular proof in which we separate the computation of the matrix elements $v_{ij} = a_i \cdot b_j$ from the derivation of the output shares c_i.

Lemma 1 (t-SNI of SecMult). *Let $(a_i)_{1\leqslant i\leqslant n}$ and $(b_i)_{1\leqslant i\leqslant n}$ be the input shares of the SecMult operation, and let $(c_i)_{1\leqslant i<n}$ be the output shares. For any set of t_1 intermediate variables and any subset $|\mathcal{O}| \leqslant t_2$ of output shares such that $t_1 + t_2 < n$, there exist two subsets I and J of indices with $|I| \leqslant t_1$ and $|J| \leqslant t_1$, such that those t_1 intermediate variables as well as the output shares $c_{|\mathcal{O}}$ can be perfectly simulated from $a_{|I}$ and $b_{|J}$.*

Algorithm 1 SecMult

Require: shares a_i satisfying $\bigoplus_{i=1}^{n} a_i = a$, shares b_i satisfying $\bigoplus_{i=1}^{n} b_i = b$
Ensure: shares c_i satisfying $\bigoplus_{i=1}^{n} c_i = a \cdot b$

for $i = 1$ **to** n **do**
  $c_i \leftarrow a_i \cdot b_i$
end for
for $i = 1$ **to** n **do**
  for $j = i + 1$ **to** n **do**
    $r \leftarrow \mathbb{F}_{2^k}$ ▷ referred by $r_{i,j}$
    $c_i \leftarrow c_i \oplus r$ ▷ referred by $c_{i,j}$
    $r \leftarrow (a_i \cdot b_j \oplus r) \oplus a_j \cdot b_i$ ▷ referred by $r_{j,i}$
    $c_j \leftarrow c_j \oplus r$ ▷ referred by $c_{j,i}$
  end for
end for
return $(c_1, \ldots, c_n)$

3.2 Mask Refreshings

To obtain security against t probes with $n \geqslant t+1$ shares instead of $n \geqslant 2t+1$, the previous SecMult algorithm is usually not sufficient; one must also use a mask refreshing algorithm. The following RefreshMask operation is used in [BBD+15a] to get the t-SNI security of a full construction.

Algorithm 2 RefreshMask

Input: $a_1, \ldots, a_n$
Output: $c_1, \ldots, c_n$ such that $\bigoplus_{i=1}^{n} c_i = \bigoplus_{i=1}^{n} a_i$

For $i = 1$ **to** n **do** $c_i \leftarrow a_i$
for $i = 1$ to n **do**
  for $j = i + 1$ to n **do**
    $r \leftarrow \{0,1\}^k$
    $c_i \leftarrow c_i \oplus r$
    $c_j \leftarrow c_j \oplus r$
  end for
end for
return $c_1, \ldots, c_n$

The following lemma is proven in [BBD+15a], showing the t-SNI security of RefreshMask. In Appendix B.3 we also provide a modular proof, using the same approach as in Lemma 1; namely the above RefreshMask algorithm can be viewed as a SecMult with multiplication by 1, with shares $(1, 0, \ldots, 0)$; therefore the same proof technique applies.

Lemma 2 (t-SNI of RefreshMask). *Let $(a_i)_{1 \leqslant i \leqslant n}$ be the input shares of the* RefreshMask *operation, and let $(c_i)_{1 \leqslant i \leqslant n}$ be the output shares. For any set of t_1 intermediate variables and any subset $|\mathcal{O}| \leqslant t_2$ of output shares such that $t_1 + t_2 < n$, there exists a subset I of indices with $|I| \leqslant t_1$, such that the t_1 intermediate variables as well as the output shares $c_{|\mathcal{O}}$ can be perfectly simulated from $a_{|I}$.*

3.3 Application to the Computation of x^{254} in $\mathbb{F}_{2^8}$

To compute $y = x^{254}$ over $\mathbb{F}_{2^8}$ with 4 multiplications, the following sequence of operation is used in [RP10], including two RefreshMask operations.

Algorithm 3 SecExp254

Input: shares $x_1, \ldots, x_n$ satisfying $x = \bigoplus_{i=1}^{n} x_i$
Output: shares $y_1, \ldots, y_n$ such that $\bigoplus_{i=1}^{n} y_i = x^{254}$

1: **For** $i = 1$ to n **do** $z_i \leftarrow x_i^2$ ▷ $\bigoplus_i z_i = x^2$
2: $(z_i)_{1 \leqslant i \leqslant n} \leftarrow \mathsf{RefreshMask}((z_i)_{1 \leqslant i \leqslant n})$
3: $(y_i)_{1 \leqslant i \leqslant n} \leftarrow \mathsf{SecMult}((z_i)_{1 \leqslant i \leqslant n}, (x_i)_{1 \leqslant i \leqslant n})$ ▷ $\bigoplus_i y_i = x^3$
4: **For** $i = 1$ to n **do** $w_i \leftarrow y_i^4$ ▷ $\bigoplus_i w_i = x^{12}$
5: $(w_i)_{1 \leqslant i \leqslant n} \leftarrow \mathsf{RefreshMask}((w_i)_{1 \leqslant i \leqslant n})$
6: $(y_i)_{1 \leqslant i \leqslant n} \leftarrow \mathsf{SecMult}((y_i)_{1 \leqslant i \leqslant n}, (w_i)_{1 \leqslant i \leqslant n})$ ▷ $\bigoplus_i y_i = x^{15}$
7: **For** $i = 1$ to n **do** $y_i \leftarrow y_i^{16}$ ▷ $\bigoplus_i y_i = x^{240}$
8: $(y_i)_{1 \leqslant i \leqslant n} \leftarrow \mathsf{SecMult}((y_i)_{1 \leqslant i \leqslant n}, (w_i)_{1 \leqslant i \leqslant n})$ ▷ $\bigoplus_i y_i = x^{252}$
9: $(y_i)_{1 \leqslant i \leqslant n} \leftarrow \mathsf{SecMult}((y_i)_{1 \leqslant i \leqslant n}, (z_i)_{1 \leqslant i \leqslant n})$ ▷ $\bigoplus_i y_i = x^{254}$
10: **return** $y_1, \ldots, y_n$

Using the two previous lemmas, one can prove the t-SNI security of SecExp254; we refer to [BBD+15a] for the proof.

Lemma 3 (t-SNI of x^{254}). *Let $(x_i)_{1 \leqslant i \leqslant n}$ be the input shares of* ExpSec254*, and let $(y_i)_{1 \leqslant i \leqslant n}$ be the output shares. For any set of t_1 intermediate variables and any subset $|\mathcal{O}| \leqslant t_2$ of output shares such that $t_1 + t_2 < n$, there exists a subset I of indices with $|I| \leqslant t_1$, such that those t_1 intermediate variables as well as the output shares $y_{|\mathcal{O}}$ can be perfectly simulated from $x_{|I}$.*

As explained in [BBD+15a], since the SecExp254 operation has the t-SNI property, it can be used to secure a full AES against t probes with $n \geqslant t + 1$ shares.

4 Secure Computation of 2 Parallel Multiplications with Common Operand, and Application to AES

In this section we show a first efficiency improvement of the Rivain-Prouff countermeasure for AES recalled in the previous section. Namely, we show that when two finite-field multiplications $d \leftarrow c \cdot a$ and $e \leftarrow c \cdot b$ have the same operand c, we can save $n^2/2$ field multiplications in SecMult by making sure that the inputs a and b have half of their shares in common; we then show how to apply this technique to the evaluation of the AES SBox, by using an alternative evaluation circuit for x^{254}.

Arithmetic Circuit with Depth 3 for x^{254}. The original arithmetic circuit for computing $y = x^{254}$ over $\mathbb{F}_{2^8}$ from [RP10] and recalled in Section 3.3 has 4 multiplicative levels, with a total of 4 non-linear multiplications. Below we use an alternative circuit with only 3 multiplicative levels, still with 4 multiplications, as described in [GHS12]; see Fig. 1 for an illustration.

- Level 1: compute $x^3 = x \cdot x^2$ (1 mult) and then $x^{12} = (x^3)^4$.
- Level 2: compute $x^{14} = x^{12} \cdot x^2$ (1 mult) and $x^{15} = x^{12} \cdot x^3$ (1 mult), and then $x^{240} = (x^{15})^{16}$.
- Level 3: compute $x^{254} = x^{240} \cdot x^{14}$ (1 mult).

Multiplications with Common Shares. In the arithmetic circuit above, the multiplications $x^{14} \leftarrow x^{12} \cdot x^2$ and $x^{15} \leftarrow x^{12} \cdot x^3$ can be computed in parallel; moreover they have one operand x^{12} in common. More generally, assume that we must compute two multiplications with a common

operand c:

$$d \leftarrow c \cdot a$$
$$e \leftarrow c \cdot b$$

The SecMult algorithm will compute the cross-products $c_i \cdot a_j$ and $c_i \cdot b_j$ for all $1 \leqslant i, j \leqslant n$. Now assume that half of the shares of a and b are the same, that is $a_j = b_j$ for all $1 \leqslant j \leqslant n/2$. In that case the products $c_i \cdot a_j$ for $1 \leqslant j \leqslant n/2$ have to be computed only once, and therefore when processing $e \leftarrow c \cdot b$, we only have to compute $n^2/2$ multiplications instead of n^2; see Fig. 2 for an illustration. For an arithmetic circuit with 4 multiplications as above, this saves the equivalent of 0.5 multiplication.

4.1 The CommonShares Algorithm

The CommonShares algorithm below ensures that the output shares a'_i and b'_i corresponding to a and b are the same on the first half, that is $a'_i = b'_i$ for all $1 \leqslant i \leqslant n/2$. In the rest of the paper, for simplicity we assume that n is even.

Algorithm 4 CommonShares

Require: shares a_i satisfying $\bigoplus_{i=1}^n a_i = a$, shares b_i satisfying $\bigoplus_{i=1}^n b_i = b$
Ensure: shares a'_i and b'_i satisfying $\bigoplus_{i=1}^n a'_i = a$ and $\bigoplus_{i=1}^n b'_i = b$, with $a'_i = b'_i$ for all $1 \leqslant i \leqslant n/2$
1: **for** $i = 1$ **to** $n/2$ **do**
2: $\quad r_i \leftarrow^{\$} \mathbb{F}_{2^k}$
3: $\quad a'_i \leftarrow r_i, \;\; a'_{n/2+i} \leftarrow (a_{n/2+i} \oplus r_i) \oplus a_i$ ▷ $a'_i \oplus a'_{n/2+i} = a_i \oplus a_{n/2+i}$
4: $\quad b'_i \leftarrow r_i, \;\; b'_{n/2+i} \leftarrow (b_{n/2+i} \oplus r_i) \oplus b_i$ ▷ $b'_i \oplus b'_{n/2+i} = b_i \oplus b_{n/2+i}$
5: **end for**
6: **return** $(a'_i)_{1 \leqslant i \leqslant n}$ and $(b'_i)_{1 \leqslant i \leqslant n}$

It is easy to see that we still get as output an n-sharing of the same variables a and b, since for each $1 \leqslant i \leqslant n/2$ we have $a'_i \oplus a'_{n/2+i} = a_i \oplus a_{n/2+i}$, and similarly for b. As explained previously, we cannot have more than $n/2$ shares in common between a and b, since otherwise there would be a straightforward attack with fewer than n probes: namely if $a_i = b_i$ for all $1 \leqslant i \leqslant k$, then we can probe the $2(n-k)$ remaining shares a_i and b_i for $k+1 \leqslant i \leqslant n$; if $k > n/2$ this gives strictly less than n shares, whose xor gives the secret variable $a \oplus b$. Hence having half of the shares in common is optimal.

The following Lemma shows the security of the CommonShares algorithm; as will be shown later, for this algorithm we only need the weaker t-NI security property (instead of t-SNI).

Lemma 4 (t-NI of CommonShares). *Let $(a_i)_{1 \leqslant i \leqslant n}$ and $(b_i)_{1 \leqslant i \leqslant n}$ be the input shares of the algorithm* CommonShares*, and let $(a'_i)_{1 \leqslant i \leqslant n}$ and $(b'_i)_{1 \leqslant i \leqslant n}$ be the output shares. For any set of t_1 intermediate variables and any subsets of indices $I, J \subset [1, n]$, there exists a subset $\mathcal{S} \subset [1, n]$ with $|\mathcal{S}| \leqslant |I| + |J| + t_1$, such that those t_1 variables as well as the output shares $a'_{|I}$ and $b'_{|J}$ can be perfectly simulated from $a_{|\mathcal{S}}$ and $b_{|\mathcal{S}}$.*

Proof. The proof intuition is as follows. If for a given i with $1 \leqslant i \leqslant n/2$ the adversary requests only one of the variables r_i, $a_{n/2+i} \oplus r_i$, $b_{n/2+i} \oplus r_i$, $a'_{n/2+i}$ or $b'_{n/2+i}$, then such variable can be perfectly

simulated without knowing any of the input shares a_i, b_i, $a_{n/2+i}$ and $b_{n/2+i}$, thanks to the mask r_i. On the other hand, if two such variables (or more) are requested, then we can provide a perfect simulation from the 4 previous input shares, whose knowledge is obtained by adding the two indices i and $n/2+i$ in $\mathcal{S}$. Therefore we never add more than one index in $\mathcal{S}$ per probe (or per output index in I or J), which implies that the size of the subset $\mathcal{S}$ of input shares is upper-bounded by $|I|+|J|+t_1$, as required.[3]

More precisely, we describe hereafter the construction of the set $\mathcal{S} \subset [1,n]$ of input shares, initially empty. For every probed input variable a_i and b_i (for any i), we add i to $\mathcal{S}$. For all $1 \leqslant i \leqslant n/2$, we let t_i be the number of probed variables among $a_{n/2+i} \oplus r_i$ and $b_{n/2+i} \oplus r_i$. We let:

$$\lambda_i := t_i \; + \; |\{i, n/2+i\} \cap I| \; + \; |\{i, n/2+i\} \cap J| \; ,$$

We then add $\{i, n/2+i\}$ to $\mathcal{S}$ if $\lambda_i \geqslant 2$. This terminates the construction of $\mathcal{S}$. By construction of $\mathcal{S}$, we must have $|\mathcal{S}| \leqslant |I|+|J|+t$ as required.

We now show that the output shares $a'_{|I}$ and $b'_{|J}$ and the t_1 intermediate variables of Algorithm CommonShares can be perfectly simulated from $a_{|\mathcal{S}}$ and $b_{|\mathcal{S}}$. This is clear for the probed input variables a_i and b_i. For all $1 \leqslant i \leqslant n/2$, we distinguish two cases. If $\lambda_i \geqslant 2$, then $\{i, n/2+i\} \in \mathcal{S}$, so we can let $r_i \leftarrow \mathbb{F}_{2^k}$ as in the real algorithm and simulate all output and intermediate variables from the knowledge of a_i, $a_{n/2+i}$, b_i and $b_{n/2+i}$. If $\lambda_i = 1$, then if $t_i = 0$, then only a single output variable among a'_i, b'_i, $a'_{n/2+i}$ and $b'_{n/2+i}$ must be simulated. Since each of those variables is masked by r_i, we can simulate this single output variable by generating a random value in $\mathbb{F}_{2^k}$. Similarly, if $t_i = 1$, then only one of the two intermediate variables among $a_{n/2+i} \oplus r_i$ and $b_{n/2+i} \oplus r_i$ is probed (while no output variable must be simulated), and therefore we can also simulate such variable by generating a random value in $\mathbb{F}_{2^k}$. This terminates the proof of Lemma 4. □

4.2 The CommonMult Algorithm

To perform the two multiplications with the same operand $d \leftarrow c \cdot a$ and $e \leftarrow c \cdot b$, instead of doing two independent SecMult, we define the following CommonMult algorithm below.

Algorithm 5 CommonMult

Input: shares satisfying $c = \bigoplus_{i=1}^{n} c_i$, $a = \bigoplus_{i=1}^{n} a_i$ and $b = \bigoplus_{i=1}^{n} b_i$.
Output: d_i such that $\bigoplus_{i=1}^{n} d_i = c \cdot a$, and e_i such that $\bigoplus_{i=1}^{n} e_i = c \cdot b$
1: $(a'_i)_{1 \leqslant i \leqslant n}, (b'_i)_{1 \leqslant i \leqslant n} \leftarrow \mathsf{CommonShares}((a_i)_{1 \leqslant i \leqslant n}, (b_i)_{1 \leqslant i \leqslant n})$
2: $(d_i)_{1 \leqslant i \leqslant n} \leftarrow \mathsf{SecMult}((c_i)_{1 \leqslant i \leqslant n}, (a'_i)_{1 \leqslant i \leqslant n})$
3: $(e_i)_{1 \leqslant i \leqslant n} \leftarrow \mathsf{SecMult}((c_i)_{1 \leqslant i \leqslant n}, (b'_i)_{1 \leqslant i \leqslant n})$
4: **return** $(d_i)_{1 \leqslant i \leqslant n}$ and $(e_i)_{1 \leqslant i \leqslant n}$.

The algorithm first calls the previous CommonShares subroutine, to ensure that half of the shares of a and b are the same. It then applies the previous SecMult algorithm twice to securely compute the two multiplications. Then the multiplications $c_i \cdot a_j$ for $1 \leqslant j \leqslant n/2$ performed in the first SecMult can be re-used in the second SecMult, so this saves $n^2/2$ multiplications. More precisely, for the SecMult computation performed at Line 3, we don't have to compute again the products

[3] Note that the proof would not work without the masks r_i; namely with $r_i = 0$ we would need to know both a_i and $a_{n/2+i}$ to simulate $a'_{n/2+i}$; hence with t probes we would need at least $n \geqslant 2t+1$ shares, which would make CommonShares useless.

$c_i \cdot b'_j$ for $1 \leqslant j \leqslant n/2$, since those products have already been computed at Line 2 with $c_i \cdot a'_j$, since $a'_j = b'_j$ for all $1 \leqslant j \leqslant n/2$. However reusing at Line 3 the products already computed at Line 2 requires to store $\mathcal{O}(n^2)$ values. In Appendix C we describe a different version of the CommonMult algorithm above, where the matrix elements $c_i \cdot a_j$ are computed on the fly and then used in both SecMult, with memory complexity $\mathcal{O}(n)$ instead of $\mathcal{O}(n^2)$.

The following Lemma shows that the CommonMult algorithm is t-SNI secure in the ISW model, with $n \geqslant t + 1$ shares.

Lemma 5 (t-SNI of CommonMult). *Let $(a_i)_{1\leqslant i\leqslant n}$, $(b_i)_{1\leqslant i\leqslant n}$ and $(c_i)_{1\leqslant i\leqslant n}$ be the input shares of the CommonMult operation, and let $(d_i)_{1\leqslant i\leqslant n}$ and $(e_i)_{1\leqslant i\leqslant n}$ be the output shares. For any set of t_1 intermediate variables and any subsets $|\mathcal{O}_1| \leqslant t_2$ and $|\mathcal{O}_2| \leqslant t_2$ of output shares such that $t_1 + t_2 < n$, there exist two subsets I and J of indices such that $|I| \leqslant t_1$ and $|J| \leqslant t_1$, and those t_1 intermediate variables as well as the output shares $d_{|\mathcal{O}_1}$ and $e_{|\mathcal{O}_2}$ can be perfectly simulated from $a_{|J}$, $b_{|J}$ and $c_{|I}$.*

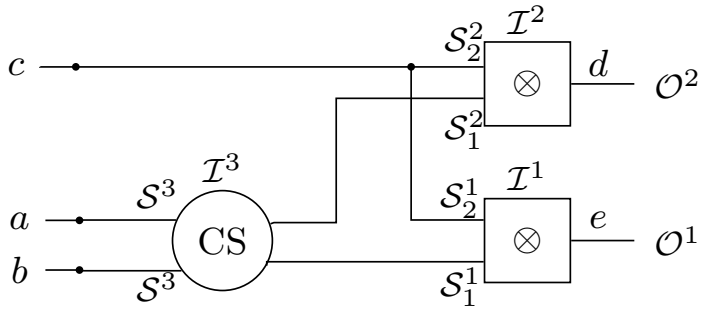

Fig. 3. The CommonMult algorithm as composition of gadgets. Each variable a, b and c contains actually n shares a_i, b_i and c_i.

Proof. The proof is relatively straightforward, following the process in [BBD+15a, Sect. 4.1]. We use Lemmas 1 and 4 to prove that the composition of the CommonShares gadget with both SecMult $\otimes$ allows the entire circuit to be t-SNI. We label the gadgets from 1 to 3 starting from right to left (see Figure 3).

Let $\mathcal{I} = \mathcal{I}^1 \cup \mathcal{I}^2 \cup \mathcal{I}^3$ be a set of indices such that $|\mathcal{I}| \leqslant t_1$, corresponding to observations of intermediate variables done by the attacker in the three gadgets, and let $\mathcal{O}^1$ and $\mathcal{O}^2$ be two sets of indices such that $\left|\mathcal{O}^1\right| \leqslant t_2$ and $\left|\mathcal{O}^2\right| \leqslant t_2$, corresponding to observations on the outputs made by the attacker.

Gadget 1 By assumption, $\left|\mathcal{I}^1\right| + \left|\mathcal{O}^1\right| \leqslant t_1 + t_2 < n$. Since from Lemma 1, the multiplication $\otimes$ is t-SNI, this means that there exist two sets of indices $\mathcal{S}^1_1, \mathcal{S}^1_2$ such that $\left|\mathcal{S}^1_1\right| \leqslant \left|\mathcal{I}^1\right|$, $\left|\mathcal{S}^1_2\right| \leqslant \left|\mathcal{I}^1\right|$ and the gadget can be perfectly simulated from its input shares corresponding to indices in $\mathcal{S}^1_1$ and $\mathcal{S}^1_2$.

Gadget 2 Similarly, there exist two sets of indices $\mathcal{S}^2_1, \mathcal{S}^2_2$ such that $\left|\mathcal{S}^2_1\right| \leqslant \left|\mathcal{I}^2\right|$ and $\left|\mathcal{S}^2_2\right| \leqslant \left|\mathcal{I}^2\right|$, and the gadget can be perfectly simulated from its input shares corresponding to indices in $\mathcal{S}^2_1$ and $\mathcal{S}^2_2$.

Gadget 3 From Lemma 4 there exists a set of indices $\mathcal{S}^3$ such that $\left|\mathcal{S}^3\right| \leqslant \left|\mathcal{I}^3\right| + \left|\mathcal{S}_1^1\right| + \left|\mathcal{S}_1^2\right|$ and Gadget 3 can be perfectly simulated from its input shares corresponding to indices in $\mathcal{S}^3$. From gadgets 1 and 2, it follows that $\left|\mathcal{S}^3\right| \leqslant \left|\mathcal{I}^3\right| + \left|\mathcal{I}^1\right| + \left|\mathcal{I}^2\right|$.

Each of the previous steps ensures the existence of a simulator for each gadget. Let $I = \mathcal{S}_2^1 \cup \mathcal{S}_2^2$ and $J = \mathcal{S}^3$. We can then compose these simulators to perfectly simulate the computation of CommonMult from $c_{|I}$, $a_{|J}$ and $b_{|J}$. Furthermore, from gadgets 1 and 2, we have $|I| = \left|\mathcal{S}_2^1\right| + \left|\mathcal{S}_2^2\right| \leqslant \left|\mathcal{I}^1\right| + \left|\mathcal{I}^2\right| \leqslant t_1$ and from Gadget 3 we have $|J| = \left|\mathcal{S}^3\right| \leqslant \left|\mathcal{I}^3\right| + \left|\mathcal{I}^2\right| + \left|\mathcal{I}^1\right| \leqslant t_1$. This concludes the proof. □

4.3 Application to AES SBoxes

We are now ready to describe the full computation of $y = x^{254}$ based on the CommonShares algorithm; the algorithm SecExp254' is described below; it is a variant of Algorithm 3.

Algorithm 6 SecExp254'

Input: shares $x_1, \ldots, x_n$ satisfying $x = \bigoplus_{i=1}^n x_i$
Output: shares $y_1, \ldots, y_n$ such that $\bigoplus_{i=1}^n y_i = x^{254}$

1: **For** $i = 1$ to n **do** $z_i \leftarrow x_i^2$ ▷ $\bigoplus_i z_i = x^2$
2: $(x_i)_{1 \leqslant i \leqslant n} \leftarrow \mathsf{RefreshMask}((x_i)_{1 \leqslant i \leqslant n})$
3: $(y_i)_{1 \leqslant i \leqslant n} \leftarrow \mathsf{SecMult}((z_i)_{1 \leqslant i \leqslant n}, (x_i)_{1 \leqslant i \leqslant n})$ ▷ $\bigoplus_i y_i = x^3$
4: **For** $i = 1$ to n **do** $w_i \leftarrow y_i^4$ ▷ $\bigoplus_i w_i = x^{12}$
5: $(w_i)_{1 \leqslant i \leqslant n} \leftarrow \mathsf{RefreshMask}((w_i)_{1 \leqslant i \leqslant n})$
6: $(z_i)_{1 \leqslant i \leqslant n}, (y_i)_{1 \leqslant i \leqslant n} \leftarrow \mathsf{CommonMult}((w_i)_{1 \leqslant i \leqslant n}, (z_i)_{1 \leqslant i \leqslant n}, (y_i)_{1 \leqslant i \leqslant n})$ ▷ $\bigoplus_i z_i = x^{14}$, $\bigoplus_i y_i = x^{15}$
7: **For** $i = 1$ to n **do** $y_i \leftarrow y_i^{16}$ ▷ $\bigoplus_i y_i = x^{240}$
8: $(y_i)_{1 \leqslant i \leqslant n} \leftarrow \mathsf{SecMult}((y_i)_{1 \leqslant i \leqslant n}, (z_i)_{1 \leqslant i \leqslant n})$ ▷ $\bigoplus_i y_i = x^{254}$
9: **return** $y_1, \ldots, y_n$

The following Lemma proves the t-SNI security of our new algorithm; therefore our new algorithm achieves exactly the same security level as Algorithm 3. That is, it can be used in the computation of a full block-cipher, with $n \geqslant t+1$ shares against t probes. We provide the proof in Appendix D.

Lemma 6 (t-**SNI of** x^{254})**.** *Let $(x_i)_{1 \leqslant i \leqslant n}$ be the input shares of the x^{254} operation, and let $(y_i)_{1 \leqslant i \leqslant n}$ be the output shares. For any set of t_1 intermediate variables and any subset $|\mathcal{O}| \leqslant t_2$ of output shares such that $t_1 + t_2 < n$, there exists a subset I of indices with $|I| \leqslant t_1$, such that those t_1 intermediate variables as well as the output shares $y_{|\mathcal{O}}$ can be perfectly simulated from $x_{|I}$.*

Finally, we summarize in Table 2 the complexities of the above algorithms. Table 2 shows that our new algorithm for x^{254} saves $n^2/2$ multiplications, with the same security level as in the original algorithm.

5 Parallel Multiplications with Common Shares

5.1 Secure Evaluation of Parallel Multiplications.

In the previous section, we have shown that by using a different arithmetic circuit for x^{254}, two multiplications in $\mathbb{F}_{2^8}$ could be processed in parallel, moreover with a common operand, and then

	# add	# mult	# rand
SecMult (Alg. 1)	$2n^2$	n^2	$n^2/2$
RefreshMask (Alg. 2)	n^2	-	$n^2/2$
SecMult $\times$ 2	$4n^2$	$2n^2$	n^2
CommonMult (Alg. 5)	$4n^2$	$3n^2/2$	n^2
SecExp254 (Alg. 3)	$10n^2$	$4n^2$	$3n^2$
SecExp254' (Alg. 6)	$10n^2$	$7n^2/2$	$3n^2$

Table 2. Complexity of CommonMult and SecExp254'; for simplicity we omit the $\mathcal{O}(n)$ terms.

by using half common shares we could save the equivalent of 1/2 multiplication out of 4 in the evaluation of an AES SBox.

We now consider the case of parallel multiplications that do not necessarily share an operand. Previously we have focused on a single evaluation of an AES SBox, but in AES the 16 SBoxes can actually be processed in parallel, and therefore each of the 4 multiplications in x^{254} can be processed in parallel. As opposed to the previous case those multiplications do not share any operand, but we show that by using a generalization of the CommonShares algorithm between m operands instead of 2, for every multiplication one can still save the equivalent of roughly 1/4 multiplication.

Namely as illustrated in Fig. 4, when performing a sequence of m parallel SecMult with $z^{(\ell)} \leftarrow x^{(\ell)} \cdot y^{(\ell)}$ for $1 \leqslant \ell \leqslant m$, if the variables $x^{(1)}, \ldots x^{(m)}$ have the same first half shares in common, and similarly for the $y^{(1)}, \ldots y^{(m)}$ variables, then all the left-upper blocks of the corresponding multiplication matrices will be the same. This implies that such block has to be processed only once (instead of m times), and therefore asymptotically for large m for each multiplication we can save the equivalent of 1/4 multiplication.

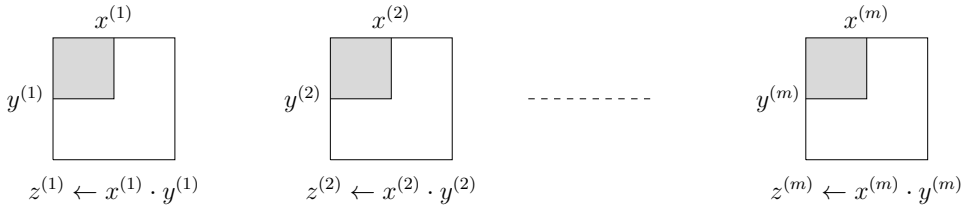

Fig. 4. A sequence of m parallel SecMult, where the operands $x^{(i)}$ have the same first half shares in common, and similarly for $y^{(i)}$.

More precisely, using a generalization of the CommonShares algorithm we can ensure that all operands $x^{(\ell)}$ for $1 \leqslant \ell \leqslant m$ have the same first half shares, that is $x_i^{(1)} = x_i^{(2)} = \ldots = x_i^{(\ell)}$ for all $1 \leqslant i \leqslant n/2$; similarly we can ensure that all operands $y^{(\ell)}$ for $1 \leqslant \ell \leqslant m$ have the same first half shares, that is $y_j^{(1)} = y_j^{(2)} = \ldots = y_j^{(\ell)}$ for all $1 \leqslant j \leqslant n/2$. This implies that for the m SecMult operations, we have to compute each product $x_i^{(\ell)} \cdot y_j^{(\ell)}$ for $1 \leqslant i, j \leqslant n/2$ only once, since it is the same for all ℓ; as illustrated in Fig. 4, this corresponds to the same upper-left 1/4-block of the multiplication matrices. Therefore for m parallel multiplications, we must process $3m + 1$ blocks instead of $4m$, which gives a relative saving of $(m-1)/(4m)$, hence asymptotically 1/4.

For Two Parallel Multiplications. For simplicity we first illustrate our approach with only $m = 2$ parallel multiplications over $\mathbb{F}_{2^k}$:

$$e \leftarrow a \cdot b$$
$$f \leftarrow c \cdot d$$

Instead of processing those two multiplications with two separate SecMult, we first ensure that the operands a and c, respectively the operands b and d, have their first half shares in common, using the previous CommonShares algorithm from Section 4.2. We obtain the following TwoParaMult algorithm.

Algorithm 7 TwoParaMult

Require: four n-sharings $(a_i)_{1\leqslant i\leqslant n}$, $(b_i)_{1\leqslant i\leqslant n}$, $(c_i)_{1\leqslant i\leqslant n}$ and $(d_i)_{1\leqslant i\leqslant n}$ of a, b, c and d respectively.
Ensure: two n-sharings $(e_i)_i$ and $(f_i)_i$ of $e = a \cdot b$ and $f = c \cdot d$ respectively.
1: $(a'_i)_{1\leqslant i\leqslant n}, (c'_i)_{1\leqslant i\leqslant n} \leftarrow \mathsf{CommonShares}((a_i)_{1\leqslant i\leqslant n}, (c_i)_{1\leqslant i\leqslant n})$
2: $(b'_i)_{1\leqslant i\leqslant n}, (d'_i)_{1\leqslant i\leqslant n} \leftarrow \mathsf{CommonShares}((b_i)_{1\leqslant i\leqslant n}, (d_i)_{1\leqslant i\leqslant n})$
3: $(e_i)_{1\leqslant i\leqslant n} \leftarrow \mathsf{SecMult}((a'_i)_{1\leqslant i\leqslant n}, (b'_i)_{1\leqslant i\leqslant n})$
4: $(f_i)_{1\leqslant i\leqslant n} \leftarrow \mathsf{SecMult}((c'_i)_{1\leqslant i\leqslant n}, (d'_i)_{1\leqslant i\leqslant n})$
5: **return** $(e_i)_{1\leqslant i\leqslant n}$ and $(f_i)_{1\leqslant i\leqslant n}$

For $m = 2$ parallel multiplications, as illustrated in Fig. 4 we must process 7 blocks instead of 8, so the relative saving is 1/8. Namely after the CommonShares algorithm we have $a'_i = c'_i$ and $b'_j = d'_j$ for all $1 \leqslant i, j \leqslant n/2$, and therefore the products $a'_i \cdot c'_j$ corresponding to the upper-left block of the multiplication matrix can be computed only once between the two SecMult. The following Lemma proves the security of the TwoParaMult algorithm.

Lemma 7 (t-SNI of TwoParaMult). *Let $(a_i)_{1\leqslant i\leqslant n}$, $(b_i)_{1\leqslant i\leqslant n}$, $(c_i)_{1\leqslant i\leqslant n}$ and $(d_i)_{1\leqslant i\leqslant n}$ be the input shares of the* TwoParaMult *operation, and let $(e_i)_{1\leqslant i\leqslant n}$ and $(f_i)_{1\leqslant i\leqslant n}$ be the output shares. For any set of t_1 intermediate variables and any subsets $|\mathcal{O}| \leqslant t_2$ and $|\mathcal{O}'| \leqslant t_2$ of output shares such that $t_1 + t_2 < n$, there exist two subsets $|I| \leqslant t_1$ and $|J| \leqslant t_1$ of indices such that the t_1 intermediate variables as well as the output shares $e_{|\mathcal{O}}$ and $f_{|\mathcal{O}'}$ can be perfectly simulated from $a_{|I}$, $b_{|J}$, $c_{|I}$ and $d_{|J}$.*

Proof. The proof is relatively straightforward. As illustrated in Fig. 5, any probe within the two SecMult gadgets can generate an index i in $\mathcal{S}_2^3$ or $\mathcal{S}_2^4$, and an index j in $\mathcal{S}_1^3$ or $\mathcal{S}_1^4$, which in turn will generate a single index $i' \in I = \mathcal{S}^1$ and a single index $j' \in J = \mathcal{S}^2$, thanks to the CommonShares Lemma. This implies that the size of the input subsets I and J is upper bounded by the number of internal probes t_1 within the TwoParaMult algorithm, as required. □

Generalization to m Parallel Multiplications. We refer to Appendix E.1 for a formal description of the algorithm (including a generalization of the CommonShares algorithm to m operands instead of 2), as well as the proof of its t-SNI security.

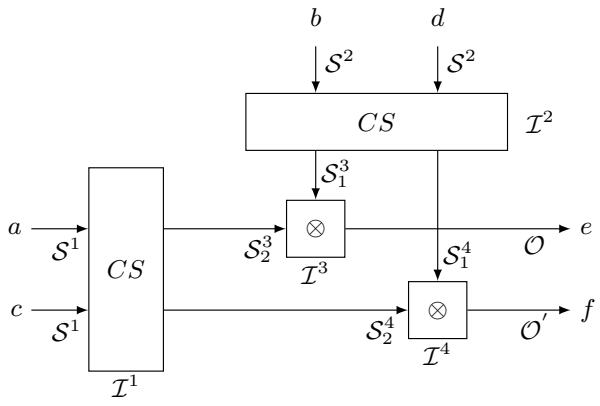

Fig. 5. The TwoParaMult algorithm as composition of gadgets.

5.2 Parallel Multiplications with Common Operand

When evaluating $y = x^{254}$ the previous technique can be combined with the technique from Section 4, in which within the evaluation of x^{254} two parallel multiplications have the same operand; in that case, for m SBoxes we get m such pairs of multiplications (with $m = 16$ for AES). As illustrated in Fig. 6, if the common operands $x^{(\ell)}$ all have the same first half shares in common, and if additionally $y^{(\ell)}$ and $z^{(\ell)}$ all have the same first half shares, then not only the $2m$ SecMult have their upper-left block in common (as previously), but for a given ℓ the two SecMult operations $x^{(\ell)} \cdot y^{(\ell)}$ and $x^{(\ell)} \cdot z^{(\ell)}$ also have the same upper-right block (since they have the same operand $x^{(\ell)}$, and $y^{(\ell)}$ and $z^{(\ell)}$ have the same first half shares). Therefore we must process only 1 upper-left block (instead of $2m$), and m upper-right blocks (instead of $2m$). In total, the number of 1/4-blocks to be processed is therefore $1 + m + 2 \cdot 2m = 5m + 1$, instead of $8m$.

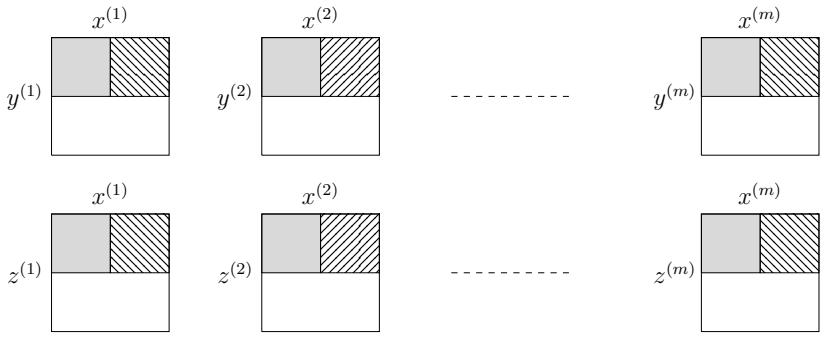

Fig. 6. A sequence of m parallel pairs of SecMult, where the operands $x^{(i)}$ have the same first half shares in common, and similarly for $y^{(i)}$ and $z^{(i)}$.

We summarize our results in Table 3; asymptotically for large m the relative cost per multiplication is 3/4 for a single multiplication, and 5/8 for two multiplications with a common operand. We refer to Appendix E.2 for a formal description of the algorithm, as well as the proof of its t-SNI security.

	Cost per mult.
Single multiplication	$\frac{3m+1}{4m}$
Two multiplications, common operand	$\frac{5m+1}{8m}$

Table 3. Relative cost for the total number of finite field multiplications, for a single multiplication and for two multiplications with a common operand, when m multiplications can be performed in parallel ($m = 16$ for AES).

5.3 Application to AES

To compute an AES SBox, the previous combined technique illustrated in Fig. 6 only applies to 2 out of the 4 multiplications required to compute x^{254}; for the other 2 multiplications we can still apply the technique illustrated in Fig. 4. Therefore the total number of 1/4-block computation for the full AES SBox becomes $2 \cdot (3m + 1) + (5m + 1) = 11m + 3$, instead of $16m$. Therefore the speed-up ratio becomes

$$\frac{11m+3}{16m} \quad ,$$

which gives 70% for $m = 16$. Therefore instead of 4 multiplications for x^{254} we have an equivalent number of multiplications of

$$4 \cdot \frac{11m+3}{16m} \quad ,$$

which gives an equivalent of 2.8 multiplications for $m = 16$, instead of 4 in the original Rivain-Prouff countermeasure; see Table 1 for a summary.

5.4 Application to other Block-Ciphers

Fast Polynomial Evaluation. In [CRV14] a generic technique for fast polynomial evaluation in $\mathbb{F}_{2^k}$ is described, with heuristic complexity $\mathcal{O}(2^{k/2}/\sqrt{k})$ for any arbitrary k-bit SBox, compared to the $\mathcal{O}(2^{k/2})$ proven complexity for the Parity-Split method from [CGP$^+$12a].

As in [CGP$^+$12a], the technique consists in first interpolating the SBox $S(x)$ with a polynomial $P(x)$ over $\mathbb{F}_{2^k}$; since the corresponding Vandermonde matrix is invertible, this is done by solving a linear system with 2^k equations and 2^k unknowns. To evaluate the polynomial $P(x)$ efficiently, the technique consists in first generating a set L of monomials x^α, including all the monomials from a given cyclotomic class. Secondly a fixed set of polynomials $q_i(x)$ is randomly generated, whose monomials are all in the precomputed set L. Then one tries to write $P(x)$ as:

$$P(x) = \sum_{i=1}^{t-1} p_i(x) \cdot q_i(x) + p_t(x) \pmod{x^{2^k} + x} \quad , \tag{2}$$

where the $p_i(x)$ also have all their monomials in the set L, for some parameter t; since the polynomials $q_i(x)$ are fixed, this can be done by solving a linear system over the coefficients of $p_i(x)$.

For a given x, the value $P(x)$ can then be evaluated by first evaluating all the monomials in the set L; this enables to evaluate the polynomials $p_i(x)$ and $q_i(x)$ without any further non-linear multiplication. Eventually $P(x)$ is evaluated from (2) with $t-1$ additional non-linear multiplications. Denoting by ℓ the number of cyclotomic classes in L, the monomials in L can be evaluated with $\ell - 2$ non-linear multiplications; the total number of non-linear multiplications is then:

$$N_{mult} = \ell + t - 3 \quad .$$

With Common Shares. For block-cipher computation, we have that Equation (2) is evaluated independently for every SBox interpolated by $P(x)$. When m SBoxes are evaluated in parallel, we can again apply our common shares technique from Section 5 with parameter m. From Table 3 the equivalent number of multiplication becomes:

$$N'_{mult} = N_{mult} \cdot \frac{3m+1}{4m} \quad ,$$

where N_{mult} is the number of non-linear multiplications in [CRV14]. We summarize our results in Table 4 for various block-ciphers, using the same (ℓ, t) decompositions as in [CRV14]. In all cases this enables to improve the number of multiplications compared to [CRV14].

	k	m	ℓ	t	N_{mult}	N'_{mult}
DES	6	8	4	3	4	3.1
PRESENT	4	16	3	2	2	1.5
SERPENT	4	32	3	2	2	1.5
CAMELLIA	8	8	7	6	10	7.8
CLEFIA	8	8	7	6	10	7.8

Table 4. Equivalent number of multiplications N'_{mult} for various block-ciphers, with k-bit SBoxes.

6 Parallel Computation of Quadratic Functions

6.1 Secure Evaluation of Quadratic Functions

In [CPRR15], the authors propose a generalization of an idea originally published in [CPRR13] to securely process any function h of algebraic degree[4] 2, with application to the secure evaluation of SBoxes. The algorithm is based on the following equation:

$$h\left(\sum_{i=1}^{n} x_i\right) = \sum_{1 \leqslant i < j \leqslant n} \big(h(x_i + x_j + s_{ij}) + h(x_i + s_{ij}) + h(x_j + s_{ij}) + h(s_{ij})\big) + \sum_{i=1}^{n} h(x_i) + ((n+1) \bmod 2) \cdot h(0) \tag{3}$$

which holds for any $s_{ij} \in \mathbb{F}_{2^k}$. From the above equation, any function h of algebraic degree 2 can be securely processed with n-th order security. Equation (3) is a consequence of the following theorem proved in [CPRR15].

Theorem 1. *Let n and m be two positive integers such that $m \leqslant n$. Let h be a function from $\mathbb{F}_2^n$ into $\mathbb{F}_2^m$ with algebraic degree at most s. Then, for every $d \geqslant s$ we have:*

$$h\Big(\sum_{i=1}^{d} a_i\Big) = \sum_{1 \leqslant i_1 < \cdots < i_s \leqslant d} \varphi_h^{(s)}(a_{i_1}, \ldots, a_{i_s}) + \sum_{j=0}^{s-1} \eta_{d,s}(j) \sum_{\substack{I \subseteq [\![1,d]\!] \\ |I|=j}} h\Big(\sum_{i \in I} a_i\Big) \quad ,$$

[4] The *algebraic degree* of a function h is the integer value $\max_{a_i \neq 0}(\mathrm{HW}(i))$ where the a_i's are the coefficients of the polynomial representation of h and where $\mathrm{HW}(i)$ denotes the Hamming weight of i.

where $\eta_{d,s}(j) = \binom{d-j-1}{s-j-1} \bmod 2$ *for every* $j \leqslant s-1$, *and* $\varphi_h^{(s)}(a_1, \ldots, a_s) := \sum_{I \subseteq [1,t]} h(\sum_{i \in I} a_i)$.

We prove the theorem above for the particular case of quadratic functions. For $s = 2$ we must show:

$$h\left(\sum_{i=1}^{n} x_i\right) = \sum_{1 \leqslant i < j \leqslant n} \big(h(x_i + x_j) + h(x_i) + h(x_j) + h(0)\big) + \sum_{i=1}^{n} h(x_i) + ((n+1) \bmod 2) \cdot h(0) \quad (4)$$

A quadratic function $h(x)$ over $\mathbb{F}_{2^k}$ can be written as a sum of quadratic terms, linear terms and constant term:

$$h(x) = \sum_{a,b} \alpha_{ab} \cdot x^{2^a + 2^b} + \sum_{a} \beta_a \cdot x^{2^a} + c$$

for some coefficients α_{ab}, β_a and $c \in \mathbb{F}_{2^k}$. We show that Equation (4) is satisfied by the quadratic terms, the linear terms and the constant term. For the quadratic term we can take $a = 0$ without loss of generality. We have:

$$\begin{aligned}(x+y)^{1+2^b} &= (x+y) \cdot (x+y)^{2^b} = (x+y) \cdot \left(x^{2^b} + y^{2^b}\right) \\ &= x^{1+2^b} + y^{1+2^b} + x \cdot y^{2^b} + y \cdot x^{2^b}\end{aligned}$$

which gives:

$$\begin{aligned}\left(\sum_{i=1}^{n} x_i\right)^{1+2^b} &= \sum_{1 \leqslant i \leqslant n, 1 \leqslant j \leqslant n} x_i \cdot x_j^{2^b} = \sum_{1 \leqslant i < j \leqslant n} \left(x_i \cdot x_j^{2^b} + x_j \cdot x_i^{2^b}\right) + \sum_{i=1}^{n} x_i^{1+2^b} \\ &= \sum_{1 \leqslant i < j \leqslant n} \left((x_i + x_j)^{1+2^b} + x_i^{1+2^b} + x_j^{1+2^b}\right) + \sum_{i=1}^{n} x_i^{1+2^b}\end{aligned}$$

so Equation (4) holds for the quadratic terms. It clearly holds for the linear terms and the constant term, so it holds for any quadratic function.

Moreover, we obtain with $n = 3$:

$$h(r+x+y) = h(r) + h(r+x) + h(r+y) + h(x+y) + h(x) + h(y) + h(0)$$

which enables to randomize (4) to obtain (3).

6.2 The QuadraticEval Algorithm

We recall the QuadraticEval algorithm from [CPRR15] derived from Equation (3). We use a slight variant of the algorithm in [CPRR15], so that we can latter apply our common shares technique from Section 5; the only difference is that instead of letting

$$r_{ji} \leftarrow r_{ij} + h(x_i + s_{ij}) + h(x_j + s_{ij}) + h((x_i + s_{ij}) + x_j) + h(s_{ij})$$

we change the order of processing in Line 9 of Algorithm 8. We also use a variant with memory $\mathcal{O}(n)$ instead of $\mathcal{O}(n^2)$.

As explained in [CPRR15], the advantage of the $h(x)$ function is that it can be entirely tabulated. Namely for some values of k in $\mathbb{F}_{2^k}$, a lookup table of 2^k entries might be affordable while a lookup

Algorithm 8 QuadraticEval

Input: $x_1, \ldots, x_n$ and a quadratic function h
Output: $c_1, \ldots, c_n$ such that $\oplus_i c_i = h(\oplus_i x_i)$

for $i = 1$ to n **do**
  $c_i \leftarrow h(x_i)$
end for
for $i = 1$ to n **do**
  for $j = i + 1$ to n **do**
    $r \leftarrow^{\$} \mathbb{F}_{2^k}$ ▷ referred by r_{ij}
    $c_i \leftarrow c_i + r$ ▷ referred by c_{ij}
    $s \leftarrow^{\$} \mathbb{F}_{2^k}$ ▷ referred by s_{ij}
    $r \leftarrow \big[r + \big(h(x_i + s) + h((x_i + s) + x_j)\big)\big] + \big(h(x_j + s) + h(s)\big)$ ▷ referred by r_{ji}
    $c_j \leftarrow c_j + r$ ▷ referred by c_{ji}
  end for
end for
if n is even **then**
  $c_1 = c_1 + h(0)$
end if
return $(c_1, c_2, \ldots, c_n)$

table of 2^{2k} entries for the multiplication is not (typically for $k = 8$ giving 256 bytes *vs.* 64 kilobytes). In such a situation, the cost of a secure evaluation of h is expected to be significantly lower than the cost of a secure multiplication. We hence expect Algorithm 8 to be more efficient than the ISW scheme. We refer to Appendix F.1 for the proof of t-SNI security, using the same modular approach as in our proof of the original SecMult.

6.3 Application to AES

The evaluation of x^{254} based on the quadratic function $h(x) = x^5$ is illustrated in Fig. 7. The evaluation of x^{254} then takes 3 evaluations of h, and 1 regular multiplication (instead of 4 regular multiplications).

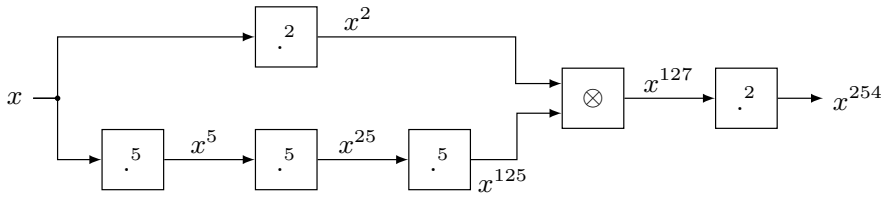

Fig. 7. Evaluation of x^{254}

6.4 Quadratic Evaluation with Common Shares

We now provide a formal description of the algorithm for two parallel evaluations of $h(x)$ with commons shares; the generalization to m parallel evaluations is straightforward. We also prove its t-SNI security.

Namely, we show that our technique from Section 5 can be applied to the previous QuadraticEval algorithm (Alg. 8). Namely Algorithm 8 has the same structure as the original ISW multiplication (Alg. 1): the only difference is that the product $a_i \cdot b_j$ is replaced by

$$v_{ij} = h(x_i + s_{ij}) + h((x_i + s_{ij}) + x_j)$$

and symmetrically the product $a_j \cdot b_i$ is replaced by

$$v_{ji} = h(x_j + s_{ij}) + h(s_{ij})$$

We can make this explicit by using a modular approach, in which we first construct the matrix elements v_{ij} (which gives an n^2-sharing of $h(x)$), and then "compress" the n^2-sharing to get a regular n-sharing of $h(x)$, as in the original ISW multiplication. The two steps can then be analyzed separately from a security standpoint, and since the second step is exactly the same as in the original ISW multiplication, only the matrix construction step must be analyzed. We refer to Appendix F.1 for the proof of t-SNI security of Alg. 8 using this approach.

Using this modular approach, it is then easy to apply our common shares technique from Section 5. Namely if two inputs x and x' have the same half common shares, that is $x_i = x'_i$ for all $1 \leqslant i \leqslant n/2$, then we can use the same matrix elements v_{ij} for both x and x', for all $1 \leqslant i, j \leqslant n/2$. As in Section 5, for a large number m of parallel evaluation this saves roughly $1/4$ of the total computation. We stress that only the randoms s_{ij} for $1 \leqslant i \leqslant j \leqslant n/2$ are shared between the two evaluations of $h(x)$ and $h(x')$; the randoms r_{ij} are not shared. Moreover the v_{ij} elements can be computed on the fly so that the memory complexity is $\mathcal{O}(n)$ instead of $\mathcal{O}(n^2)$. Below we provide a formal description of the algorithm TwoQuadraticEval for two parallel evaluations of $h(x)$; the generalization to m parallel evaluations is straightforward. We summarize the complexities in Table 5, for two parallel evaluations of QuadraticEval.

	# add	# eval_h	# mult	# rand
SecMult (Alg. 1)	$2n^2$	-	n^2	$n^2/2$
QuadraticEval (Alg. 8)	$9n^2/2$	$2n^2$	-	n^2
QuadraticEval $\times$ 2	$9n^2$	$4n^2$	-	$2n^2$
TwoQuadraticEval (Alg. 9)	$67n^2/8$	$7n^2/2$	-	$15n^2/8$

Table 5. Complexity of QuadraticEval and TwoQuadraticEval algorithms. For simplicity we omit the $\mathcal{O}(n)$ terms.

The following lemma highlights the t-SNI security of TwoQuadraticEval; we refer to Appendix F.2 for the proof.

Lemma 8 (t-SNI of TwoQuadraticEval). *Let $(a_i)_{1\leqslant i\leqslant n}$ and $(a'_i)_{1\leqslant i\leqslant n}$ be the input shares of the* TwoQuadraticEval *algorithm, and let $(c_i)_{1\leqslant i\leqslant n}$ and $(c'_i)_{1\leqslant i\leqslant n}$ be the output shares. For any set of t_1 intermediate variables and any subsets $|\mathcal{O}| \leqslant t_2$ and $|\mathcal{O}'| \leqslant t_2$ of output shares such that $t_1 + t_2 < n$, there exists a subset I of indices with $|I| \leqslant t_1$ such that the t_1 intermediate variables as well as the output shares $c_{|\mathcal{O}}$ and $c'_{|\mathcal{O}'}$ can be perfectly simulated from $a_{|I}$ and $a'_{|I}$.*

Algorithm 9 TwoQuadraticEval

Input: a quadratic function h and $(a_i)_{1\leqslant i\leqslant n}$, $(a'_i)_{1\leqslant i\leqslant n}$
Output: $(c_i)_{1\leqslant i\leqslant n}$ and $(c'_i)_{1\leqslant i\leqslant n}$ such that $\oplus_i c_i = h(\oplus_i a_i)$ and $\oplus_i c'_i = h(\oplus_i a'_i)$

$x_i, x'_i \leftarrow \mathsf{CommonShares}(a_i, a'_i)$ ▷ Ensures that $x'_i = x_i$ for all $1 \leqslant i \leqslant n/2$
for $i = 1$ to n **do**
  $c_i \leftarrow h(x_i)$, $c'_i \leftarrow h(x'_i)$
end for
for $i = 1$ to n **do**
  for $j = i + 1$ to n **do**
    $r \leftarrow^{\$} \mathbb{F}_{2^k}$, $r' \leftarrow^{\$} \mathbb{F}_{2^k}$ ▷ referred by r_{ij} and r'_{ij}
    $c_i \leftarrow c_i \oplus r$, $c'_i \leftarrow c'_i \oplus r'$ ▷ referred by c_{ij} and c'_{ij}
    $s \leftarrow^{\$} \mathbb{F}_{2^k}$, $v \leftarrow h(x_i \oplus s) \oplus h((x_i \oplus s) \oplus x_j)$, $w \leftarrow h(x_j \oplus s) \oplus h(s)$ ▷ referred by s_{ij}, v_{ij} and v_{ji}
    if $i \leqslant n/2$ and $j \leqslant n/2$ **then**
      $v' \leftarrow v$, $w' \leftarrow w$
    else
      $s' \leftarrow^{\$} \mathbb{F}_{2^k}$, $v' \leftarrow h(x'_i \oplus s') \oplus h((x'_i \oplus s') \oplus x'_j)$, $w' \leftarrow h(x'_j \oplus s') \oplus h(s')$. ▷ referred by s'_{ij}, v'_{ij} and v'_{ji}
    end if
    $r \leftarrow (r \oplus v) \oplus w$, $r' \leftarrow (r' \oplus v') \oplus w'$ ▷ referred by r_{ji} and r'_{ji}
    $c_j \leftarrow c_j \oplus r$, $c'_j \leftarrow c'_j \oplus r'$ ▷ referred by c_{ji} and c'_{ji}
  end for
end for
if n is even **then**
  $c_1 = c_1 \oplus h(0)$, $c'_1 = c'_1 \oplus h(0)$
end if
return $(c_i)_{1\leqslant i\leqslant n}$ and $(c'_i)_{1\leqslant i\leqslant n}$

7 Implementation

We have done a practical implementation of our algorithms for the AES SBox. More precisely we have implemented the n-shared evaluation of x^{254} in four different ways:

- RP10: using the Rivain-Prouff algorithm, as described in Alg. 3;
- CM: using our common shares technique, as described in Alg. 6;
- GPS14: using quadratic functions, as described in Alg. 8;
- GPS14CS: using quadratic functions and common shares, as explained in Section 6.4.

	8 shares				16 shares			
	RP10	CM	GPS14	GPS14CS	RP10	CM	GPS14	GPS14CS
ATmega	20360	18244	11076	12447	70966	57644	39554	40086
ARM	20333	18156	13796	13156	77264	65556	54133	50560

	32 shares				Ratio for 8,16 and 32 shares	
	RP10	CM	GPS14	GPS14CS	CM/RP10	GPS14CS/GPS14
ATmega	268.10^3	209.10^3	152.10^3	147.10^3	0.9, 0.81, 0.78	1.1, 1, 0.97
ARM	303.10^3	251.10^3	215.10^3	200.10^3	0.89, 0.85, 0.83	0.95, 0.93, 0.93

Table 6. Performances comparison of the RP10, CM, GPS14 and GPS14CS algorithms, on the ATmega and ARM platforms.

For portability, the code is written in `C`, except the field multiplication in $\mathbb{F}_{2^8}$ which is written in assembly for ATmega1284P (8-bit AVR microcontroller) and for ARM Cortex M0 (32-bit CPU). Performance is evaluated using simulators (AVR Studio for ATmega, Keil uVision for ARM). We assume that the random generation of one byte takes 1 cycle. This assumption is reasonable: there are at least several dozens of cycles between two 1-byte random number requests; on chips embedding hardware RNG, this is often enough to get a random value by a single memory access, without waiting. We give the average number of cycles to compute one AES SBox among 16 SBoxes in Table 6; see also Fig. 8.

Those implementation results show that our common shares technique is relatively practical: for a large number of shares, we get roughly a 20% speed improvement compared to the Rivain-Prouff countermeasure (but only roughly 5% compared to the quadratic evaluation technique in [GPS14]).

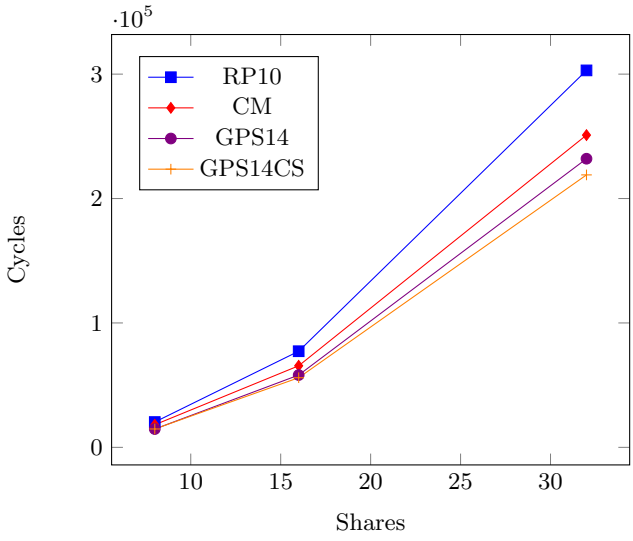

Fig. 8. AES SBox secure computation on ARM

Acknowledgments. We wish to thank Sonia Belaïd who applied the EasyCrypt verification tool [BBD+15b] on our AES SBox algorithm with common shares, at order $n = 6$.

References

[BBD+15a] Gilles Barthe, Sonia Belaïd, François Dupressoir, Pierre-Alain Fouque, and Benjamin Grégoire. Compositional verification of higher-order masking: Application to a verifying masking compiler. Cryptology ePrint Archive, Report 2015/506, 2015. `http://eprint.iacr.org/`.

[BBD+15b] Gilles Barthe, Sonia Belaïd, François Dupressoir, Pierre-Alain Fouque, Benjamin Grégoire, and Pierre-Yves Strub. Verified proofs of higher-order masking. In *Advances in Cryptology - EUROCRYPT 2015 - 34th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Sofia, Bulgaria, April 26-30, 2015, Proceedings, Part I*, pages 457–485, 2015.

[BCO04] É. Brier, C. Clavier, and F. Olivier. Correlation Power Analysis with a Leakage Model. In M. Joye and J.-J. Quisquater, editors, *Cryptographic Hardware and Embedded Systems – CHES 2004*, volume 3156 of *LNCS*, pages 16–29. SV, 2004.

[BGN+14] Begül Bilgin, Benedikt Gierlichs, Svetla Nikova, Ventzislav Nikov, and Vincent Rijmen. Higher-order threshold implementations. In Palash Sarkar and Tetsu Iwata, editors, *Advances in Cryptology - ASIACRYPT 2014 - 20th International Conference on the Theory and Application of Cryptology and Information Security, Kaoshiung, Taiwan, R.O.C., December 7-11, 2014, Proceedings, Part II*, volume 8874 of *Lecture Notes in Computer Science*, pages 326–343. Springer, 2014.

[Bla79] G.R. Blakely. Safeguarding cryptographic keys. In *National Comp. Conf.*, volume 48, pages 313–317, New York, June 1979. AFIPS Press.

[CGP+12a] Claude Carlet, Louis Goubin, Emmanuel Prouff, Michaël Quisquater, and Matthieu Rivain. Higher-order masking schemes for s-boxes. In Anne Canteaut, editor, *FSE*, volume 7549 of *Lecture Notes in Computer Science*, pages 366–384. Springer, 2012.

[CGP+12b] Jean-Sébastien Coron, Christophe Giraud, Emmanuel Prouff, Soline Renner, Matthieu Rivain, and Praveen Kumar Vadnala. Conversion of security proofs from one leakage model to another: A new issue. In Werner Schindler and Sorin A. Huss, editors, *Constructive Side-Channel Analysis and Secure Design - Third International Workshop, COSADE 2012, Darmstadt, Germany, May 3-4, 2012. Proceedings*, volume 7275 of *Lecture Notes in Computer Science*, pages 69–81. Springer, 2012.

[CJRR99] Suresh Chari, Charanjit S. Jutla, Josyula R. Rao, and Pankaj Rohatgi. Towards sound approaches to counteract power-analysis attacks. In *CRYPTO*, 1999.

[Cor14] Jean-Sébastien Coron. Higher order masking of look-up tables. In *Advances in Cryptology - EUROCRYPT 2014 - 33rd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Copenhagen, Denmark, May 11-15, 2014. Proceedings*, pages 441–458, 2014.

[CPRR13] Jean-Sébastien Coron, Emmanuel Prouff, Matthieu Rivain, and Thomas Roche. Higher-order side channel security and mask refreshing. In Shiho Moriai, editor, *Fast Software Encryption - 20th International Workshop, FSE 2013, Singapore, March 11-13, 2013. Revised Selected Papers*, volume 8424 of *Lecture Notes in Computer Science*, pages 410–424. Springer, 2013.

[CPRR15] Claude Carlet, Emmanuel Prouff, Matthieu Rivain, and Thomas Roche. Algebraic decomposition for probing security. In Rosario Gennaro and Matthew Robshaw, editors, *Advances in Cryptology - CRYPTO 2015 - 35th Annual Cryptology Conference, Santa Barbara, CA, USA, August 16-20, 2015, Proceedings, Part I*, volume 9215 of *Lecture Notes in Computer Science*, pages 742–763. Springer, 2015.

[CRV14] Jean-Sébastien Coron, Arnab Roy, and Srinivas Vivek. Fast evaluation of polynomials over binary finite fields and application to side-channel countermeasures. In *Cryptographic Hardware and Embedded Systems - CHES 2014 - 16th International Workshop, Busan, South Korea, September 23-26, 2014. Proceedings*, pages 170–187, 2014.

[DDF14] Alexandre Duc, Stefan Dziembowski, and Sebastian Faust. Unifying leakage models: From probing attacks to noisy leakage. In *Advances in Cryptology - EUROCRYPT 2014 - 33rd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Copenhagen, Denmark, May 11-15, 2014. Proceedings*, pages 423–440, 2014.

[GHS12] Craig Gentry, Shai Halevi, and Nigel P. Smart. Homomorphic evaluation of the AES circuit. In *Advances in Cryptology - CRYPTO 2012 - 32nd Annual Cryptology Conference, Santa Barbara, CA, USA, August 19-23, 2012. Proceedings*, pages 850–867, 2012.

[GPQ11] Laurie Genelle, Emmanuel Prouff, and Michaël Quisquater. Thwarting higher-order side channel analysis with additive and multiplicative maskings. In *CHES*, pages 240–255, 2011.

[GPS14] Vincent Grosso, Emmanuel Prouff, and François-Xavier Standaert. Efficient masked s-boxes processing - A step forward -. In David Pointcheval and Damien Vergnaud, editors, *Progress in Cryptology - AFRICACRYPT 2014 - 7th International Conference on Cryptology in Africa, Marrakesh, Morocco, May 28-30, 2014. Proceedings*, volume 8469 of *Lecture Notes in Computer Science*, pages 251–266. Springer, 2014.

[ISW03] Yuval Ishai, Amit Sahai, and David Wagner. Private circuits: Securing hardware against probing attacks. In *Advances in Cryptology - CRYPTO 2003, 23rd Annual International Cryptology Conference, Santa Barbara, California, USA, August 17-21, 2003, Proceedings*, pages 463–481, 2003.

[KJJ99] P. Kocher, J. Jaffe, and B. Jun. Differential Power Analysis. In M.J. Wiener, editor, *Advances in Cryptology – CRYPTO '99*, volume 1666 of *LNCS*, pages 388–397. SV, 1999.

[Mes00] T.S. Messerges. Using Second-order Power Analysis to Attack DPA Resistant software. In Ç.K. Koç and C. Paar, editors, *Cryptographic Hardware and Embedded Systems – CHES 2000*, volume 1965 of *LNCS*, pages 238–251. SV, 2000.

[MPG05] Stefan Mangard, Thomas Popp, and Berndt M. Gammel. Side-channel leakage of masked CMOS gates. In Alfred Menezes, editor, *Topics in Cryptology - CT-RSA 2005, The Cryptographers' Track at the RSA*

Conference 2005, San Francisco, CA, USA, February 14-18, 2005, Proceedings, volume 3376 of *Lecture Notes in Computer Science*, pages 351–365. Springer, 2005.

[MPO05] S. Mangard, N. Pramstaller, and E. Oswald. Successfully Attacking Masked AES Hardware Implementations. In J.R. Rao and B. Sunar, editors, *Cryptographic Hardware and Embedded Systems – CHES 2005*, volume 3659 of *LNCS*, pages 157–171. SV, 2005.

[MS06] Stefan Mangard and Kai Schramm. Pinpointing the side-channel leakage of masked AES hardware implementations. In Louis Goubin and Mitsuru Matsui, editors, *Cryptographic Hardware and Embedded Systems - CHES 2006, 8th International Workshop, Yokohama, Japan, October 10-13, 2006, Proceedings*, volume 4249 of *Lecture Notes in Computer Science*, pages 76–90. Springer, 2006.

[NRS11] Svetla Nikova, Vincent Rijmen, and Martin Schläffer. Secure hardware implementation of nonlinear functions in the presence of glitches. *J. Cryptology*, 24(2):292–321, 2011.

[PR11] Emmanuel Prouff and Thomas Roche. Higher-order glitches free implementation of the aes using secure multi-party computation protocols. In *CHES*, pages 63–78, 2011.

[PR13] Emmanuel Prouff and Matthieu Rivain. Higher-Order Side Channel Security and Mask Refreshing. In Thomas Johansson and Phong Q. Nguyen, editors, *Advances in Cryptology - EUROCRYPT 2013 - 32nd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Athens, Greece, May 26-30, 2013. Proceedings*, volume 7881 of *LNCS*, pages 142–159. SV, 2013.

[RBN+15] Oscar Reparaz, Begül Bilgin, Svetla Nikova, Benedikt Gierlichs, and Ingrid Verbauwhede. Consolidating masking schemes. In Rosario Gennaro and Matthew Robshaw, editors, *Advances in Cryptology - CRYPTO 2015 - 35th Annual Cryptology Conference, Santa Barbara, CA, USA, August 16-20, 2015, Proceedings, Part I*, volume 9215 of *Lecture Notes in Computer Science*, pages 764–783. Springer, 2015.

[RP10] Matthieu Rivain and Emmanuel Prouff. Provably secure higher-order masking of AES. In *Cryptographic Hardware and Embedded Systems, CHES 2010, 12th International Workshop, Santa Barbara, CA, USA, August 17-20, 2010. Proceedings*, pages 413–427, 2010.

[RV13] Arnab Roy and Srinivas Vivek. Analysis and improvement of the generic higher-order masking scheme of fse 2012. In Guido Bertoni and Jean-Sébastien Coron, editors, *CHES*, volume 8086 of *Lecture Notes in Computer Science*, pages 417–434. Springer, 2013.

[Sha79] Adi Shamir. How to Share a Secret. *Communications of the ACM*, 22(11):612–613, November 1979.

A Security Definitions

We recall the security definitions from [BBD+15a], which we make more explicit.

Definition 3 (t-NI Gadget). *Let G be a gadget taking m inputs $x^{(1)}, \ldots, x^{(m)}$, each input being given by n shares, and returning ℓ outputs $y^{(1)}, \ldots, y^{(\ell)}$, each output being given by n shares. Then the gadget G is t-NI if for all set of t_1 intermediate variables, and for all set of output indices $\mathcal{O}_1, \ldots, \mathcal{O}_\ell$ such that $t_1 + \sum_{i=1}^{\ell} |\mathcal{O}_i| \leqslant t$, there exists m subsets $I_1, \ldots, I_m$ with $|I_j| \leqslant t_1 + \sum_{i=1}^{\ell} |\mathcal{O}_i|$ for all j, such that the t_1 intermediate variables and the output variables $y^{(1)}_{|\mathcal{O}_1}, \ldots, y^{(\ell)}_{|\mathcal{O}_\ell}$ can be perfectly simulated from $x^{(1)}_{|I_1}, \ldots, x^{(m)}_{|I_m}$.*

Definition 4 (t-SNI Gadget). *Let G be a gadget taking m inputs $x^{(1)}, \ldots, x^{(m)}$, each input being given by n shares, and returning ℓ outputs $y^{(1)}, \ldots, y^{(\ell)}$, each output being given by n shares. Then the gadget G is t-SNI if for all set of t_1 intermediate variables, and for all set of output indices $\mathcal{O}_1, \ldots, \mathcal{O}_\ell$ such that each $|\mathcal{O}_i| \leqslant t_2$ with $t_1 + t_2 \leqslant t$, there exists m subsets $I_1, \ldots, I_m$ with $|I_j| \leqslant t_1$ for all j, such that the t_1 intermediate variables and the output variables $y^{(1)}_{|\mathcal{O}_1}, \ldots, y^{(\ell)}_{|\mathcal{O}_\ell}$ can be perfectly simulated from $x^{(1)}_{|I_1}, \ldots, x^{(m)}_{|I_m}$.*

B Security Proof of the SecMult Algorithm

B.1 A Direct Proof of Lemma 1

Our proof is essentially the same as in [BBD+15a]. We construct two sets I and J corresponding to the input shares of a and b respectively. We divide the internal probes in 4 groups. The four groups are processed separately and sequentially, that is we start with all probes in Group 1, and finish with all probes in Group 4.

- Group 1: If a_i, b_i or $a_i \cdot b_i$ is probed, add i to I and J.
- Group 2: If $r_{i,j}$ or $c_{i,j}$ is probed (for any $i \neq j$), add i to I and J.

Note that after the processing of Group 1 and 2 probes, we have $I = J$; we denote by U the common value of I and J after the processing of Group 1 and 2 probes.

- Group 3: If $a_i \cdot b_j \oplus r_{i,j}$ is probed: if $i \in U$ or $j \in U$, add $\{i, j\}$ to both I and J.
- Group 4: If $a_i \cdot b_j$ is probed (for any $i \neq j$), then add i to I and j to J.

We have $|I| \leqslant t_1$ and $|J| \leqslant t_1$, since for every probe we add at most one index in I and J. The simulation of probed variables in groups 1 and 4 is straightforward. Note that for $i < j$, the variable r_{ij} is used in all partial sums c_{ik} for $k \geqslant j$; moreover r_{ij} is used in $r_{ij} \oplus a_i b_j$, which is used in r_{ji}, which is used in all partial sums c_{jk} for $k \geqslant i$. Therefore if $i \notin U$, then r_{ij} is not probed and does not enter in the computation of any probed c_{ik}; symmetrically if $j \notin U$, then r_{ji} is not probed and does not enter in the computation of any probed c_{jk}.

For any pair $i < j$, we can now distinguish 4 cases:

- Case 1: $\{i, j\} \in U$. In that case, we can perfectly simulate all variables r_{ij}, $a_i \cdot b_j$, $a_i \cdot b_j \oplus r_{ij}$, $a_j \cdot b_i$ and r_{ji}. In particular, we let $r_{ij} \leftarrow \mathbb{F}_{2^k}$, as in the real circuit.
- Case 2: $i \in U$ and $j \notin U$. In that case we simulate $r_{ij} \leftarrow \mathbb{F}_{2^k}$, as in the real circuit. If $a_i \cdot b_j \oplus r_{i,j}$ is probed (Group 3), we can also simulate it since $i \in U$ and $j \in J$ by definition of the processing of Group 3 variables.
- Case 3: $i \notin U$ and $j \in U$. In that case r_{ij} has not been probed, nor any variable c_{ik}, since otherwise $i \in U$. Therefore r_{ij} is not used in the computation of any probed variable (except r_{ji}, and possibly $a_i \cdot b_j \oplus r_{i,j}$). Therefore we can simulate $r_{ji} \leftarrow \mathbb{F}_{2^k}$; moreover if $a_i \cdot b_j \oplus r_{ij}$ is probed, we can perfectly simulate it using $a_i \cdot b_j \oplus r_{ij} = a_j \cdot b_i \oplus r_{ji}$, since $j \in U$ and $i \in J$ by definition of the processing of Group 3 variables.
- Case 4: $i \notin U$ and $j \notin U$. If $a_i b_j \oplus r_{i,j}$ is probed, since r_{ij} is not probed and does not enter into the computation of any other probed variable, we can perfectly simulate such probe with a random value.

From cases 1, 2 and 3, we obtain that for any $i \neq j$, we can perfectly simulate any variable r_{ij} such that $i \in U$. This implies that we can also perfectly simulate all partial sums c_{ik} when $i \in U$, including the output variables c_i. Finally, all probed variables are perfectly simulated.

We now consider the simulation of the output variables c_i. We must show how to simulate c_i for all $i \in \mathcal{O}$, where $\mathcal{O}$ is an arbitrary subset of $[1, n]$ such that $t_1 + |\mathcal{O}| < n$. For $i \in U$, such variables are already perfectly simulated, as explained above. We now consider the output variables c_i with

$i \notin U$. We construct a subset of indices V as follows: for any probed Group 3 variable $a_i b_j \oplus r_{ij}$ such that $i \notin U$ and $j \notin U$ (this corresponds to Case 4), we put j in V if $i \in \mathcal{O}$, otherwise we put i in V. Since we have only considered Group 3 probes, we must have $|U| + |V| \leqslant t_1$, which implies $|U| + |V| + |\mathcal{O}| < n$. Therefore there exists an index $j^\star \in [1, n]$ such that $j^\star \notin U \cup V \cup \mathcal{O}$. For any $i \in \mathcal{O}$, we can write:

$$c_i = a_i b_i \oplus \bigoplus_{j \neq i} r_{ij} = r_{i,j^\star} \oplus \left(a_i b_i \oplus \bigoplus_{j \neq i, j^\star} r_{ij} \right)$$

We claim that neither $r_{i,j^\star}$ nor $r_{j^\star,i}$ do enter into the computation of any probed variable or other $c_{i'}$ for $i' \in \mathcal{O}$. Namely $i \notin U$ so neither $r_{i,j^\star}$ nor any partial sum c_{ik} was probed; similarly $j^\star \notin U$ so neither $r_{j^\star,i}$ nor any partial sum $c_{j^\star,k}$ was probed, and the output $c_{j^\star}$ does not have to be simulated since by definition $j^\star \notin \mathcal{O}$. Finally if $i < j^\star$ then $a_i b_{j^\star} \oplus r_{i,j^\star}$ was not probed since otherwise $j^\star \in V$ (since $i \in \mathcal{O}$); similarly if $j^\star < i$ then $a_{j^\star} b_i \oplus r_{j^\star,i}$ was not probed since otherwise we would have $j^\star \in V$ since $j^\star \notin \mathcal{O}$. Therefore, since neither $r_{i,j^\star}$ nor $r_{j^\star,i}$ are used elsewhere, we can perfectly simulate c_i by generating a random value. This proves the Lemma.

B.2 A Modular Proof of Lemma 1

In this section we provide an alternative proof of Lemma 1, based on a modular approach. Namely in SecMult (Algorithm 1) we can separate the computation of the matrix elements $a_i \cdot b_j$ (forming an n^2-sharing of $a \cdot b$), from the derivation of an n-sharing of $a \cdot b$. We obtain two separate algorithms MatrixMult and MatrixRows below, whose security properties can be analyzed separately. For the SecMult algorithm such modular approach is relatively straightforward, but it will be especially useful when analyzing the more complex QuadraticEval algorithm from Section 6.

Algorithm 10 MatrixMult

Require: shares a_i satisfying $\bigoplus_{i=1}^n a_i = a$, shares b_i satisfying $\bigoplus_{i=1}^n b_i = b$
Ensure: shares $v_{ij} = a_i \cdot b_j$ for $1 \leqslant i, j \leqslant n$

```
for i = 1 to n do
    for j = 1 to n do
        v_ij ← a_i · b_j
    end for
end for
return (v_ij)_{1≤i,j≤n}
```

The following Lemma shows the t-NI property of MatrixMult. The output shares of the algorithm are indexed by (i, j) for $1 \leqslant i, j \leqslant n$. Since there are n^2 output shares, we cannot hope to simulate any proper subset $\mathcal{O}$ of output shares with always strictly fewer than n shares. Therefore we restrict ourselves to subsets $\mathcal{O}$ of the form:

$$\mathcal{O} = (E \times E) \cup \mathcal{O}_2$$

where $|E| + |\mathcal{O}_2| \leqslant t_2$, instead of $|\mathcal{O}| \leqslant t_2$. We will see that this special structure is "compatible" with the set of input shares required for the next MatrixRows layer.

Lemma 9 (t-NI of MatrixMult). *Let $(a_i)_{1 \leqslant i < n}$ and $(b_i)_{1 \leqslant i < n}$ be the input shares of the* MatrixMult *operation. Let $\mathcal{O}$ be a subset of the output shares such that $\mathcal{O} = (E \times E) \cup \mathcal{O}_2$ where $|E| + |\mathcal{O}_2| \leqslant t_2$.*

Algorithm 11 MatrixRows

Require: shares v_{ij} for $1 \leqslant i, j \leqslant n$
Ensure: shares c_i such that $\bigoplus_{i=1}^{n} c_i = \bigoplus_{1 \leqslant i,j \leqslant n} v_{ij}$

for $i = 1$ **to** n **do**
  $c_i \leftarrow v_{ii}$
end for
for $i = 1$ **to** n **do**
  for $j = i + 1$ **to** n **do**
    $r \leftarrow \mathbb{F}_{2^k}$ ▷ referred by $r_{i,j}$
    $c_i \leftarrow c_i \oplus r$ ▷ referred by $c_{i,j}$
    $r \leftarrow (r \oplus v_{ij}) \oplus v_{ji}$ ▷ referred by $r_{j,i}$
    $c_j \leftarrow c_j \oplus r$ ▷ referred by $c_{j,i}$
  end for
end for
return $(c_1, \ldots, c_n)$

Then for any set of t_1 intermediate variables, there exist two subsets I and J of indices such that $|I| \leqslant t_1 + t_2$ and $|J| \leqslant t_1 + t_2$, and the t_1 intermediate variables as well as the output shares $(v_{ij})_{(i,j)\in\mathcal{O}}$ can be perfectly simulated from $a_{|I}$ and $b_{|J}$.

Proof. The proof is relatively straightforward. Using $\mathcal{O} = (E \times E) \cup \mathcal{O}_2$, we initially let $I = J = E$, and we then proceed as follows:

- For every probed variable a_i, add i in I, and for every b_i, add i in J.
- For every probed v_{ij}, add i to I and j to J.
- For every pair $(i, j) \in \mathcal{O}_2$, add i to I and j to J.

Since $|E| + |\mathcal{O}_2| \leqslant t_2$, we have $|I| \leqslant t_1 + t_2$ and $|J| \leqslant t_1 + t_2$ as required. It is easy to see that the t_1 intermediate variables and the output shares $(v_{ij})_{(i,j)\in\mathcal{O}}$ can then be perfectly simulated from $a_{|I}$ and $b_{|J}$. □

Lemma 10 (t-SNI of MatrixRows). *Let $(v_{ij})_{1\leqslant i,j\leqslant n}$ be the matrix of input shares of* MatrixRows*, and let $(c_i)_{1\leqslant i\leqslant n}$ be the output shares. For any set of t_1 intermediate variables and any subset $|\mathcal{O}| \leqslant t_2$ of output shares such that $t_1 + t_2 < n$, there exists a subset $I = (E \times E) \cup I_2$ of indices with $|E| + |I_2| \leqslant t_1$, and the t_1 intermediate variables as well as the output shares $c_{|\mathcal{O}}$ can be perfectly simulated from $(v_{ij})_{(i,j)\in I}$.*

Proof. The proof is very similar to the proof of Lemma 1 in Appendix B.1. We first describe the construction of the subsets E and I_2. We divide the internal probes into 3 groups. Those groups are processed separately and sequentially.

- Group 1: For every probed variable c_{ij} and r_{ij}, add i to E.

We let $U = E$ after the processing of Group 1 variables.

- Group 2: For every probed variable $r_{ij} \oplus v_{ij}$, if $i \in U$ or $j \in U$, add i and j to E.
- Group 3: For each probed input variable v_{ij}, add (i, j) to I_2.

Now that the subsets E and I_2 have been determined, we set $I = (E \times E) \cup I_2$ and we show that the t_1 intermediate variables and the t_2 output shares $(v_{ij})_{(i,j)\in\mathcal{O}}$ of Algorithm MatrixRows can be perfectly simulated from $(v_{ij})_{(i,j)\in I}$. Note that we have $|E| + |I_2| \leqslant t_1$ as required.

The simulation of variables in Group 3 is straightforward since by construction we have $(i,j) \in I_2 \subset I$. Therefore, it remains to simulate the variables from groups 1 and 2. Four cases arise depending on whether i and/or j (with $i < j$) belong to U or not.

- Case 1: $\{i,j\} \in U$. This implies $\{(i,j),(j,i)\} \in I$, and therefore we can perfectly simulate all variables from Groups 1 and 2 from v_{ij} and v_{ji}. In particular, we let $r_{ij} \leftarrow \mathbb{F}_{2^k}$ as in the real circuit.
- Case 2: $i \in U$ and $j \notin U$. In that case, we can simulate $r_{ij} \leftarrow \mathbb{F}_{2^k}$ as in the real circuit. Furthermore, if the variable $r_{ij} \oplus v_{ij}$ is probed, then since $i \in U$, by construction we must have i and j in E (from processing of Group 2 variables) and therefore $(i,j) \in E \times E \subset I$. Thus, the variable $r_{ij} \oplus v_{ij}$ can be perfectly simulated from v_{ij}.
- Case 3: $i \notin U$ and $j \in U$. In that case, the variable r_{ij} is not probed, nor any variable $c_{i\ell}$. Therefore, we can simulate $r_{ji} \leftarrow \mathbb{F}_{2^k}$. Moreover, if $r_{ij} \oplus v_{ij}$ is probed, then it can be perfectly simulated from $r_{ij} \oplus v_{ij} = r_{ji} \oplus v_{ji}$; namely by definition $j \in U$, and by construction, both indices i and j must belong to E (from processing of Group 2 variables) and therefore $(j,i) \in E \times E \subset I$.
- Case 4: $i \notin U$ and $j \notin U$. In that case, variables from Group 1 are not probed, and variables from Group 2 can perfectly be simulated with a random value since $r_{i,j}$ is not probed and does not enter in the computation of any other probed variable.

From cases 1, 2 and 3, we obtain that for any $i \neq j$, we can perfectly simulate any variable r_{ij} such that $i \in U$. This implies that we can also perfectly simulate all partial sums $c_{i\ell}$ when $i \in U$, including the output variables c_i. Finally, all probed variables are perfectly simulated. Finally, the output variables c_i for $i \in \mathcal{O}$ are simulated exactly as in the proof of Lemma 1. □

It remains to show that by combining Lemmas 9 and 10, the composition of the MatrixRows and MatrixMult algorithms allows Algorithm SecMult to be t-SNI, which proves Lemma 1.

Let $\mathcal{I} = \mathcal{I}^1 \cup \mathcal{I}^2$ be the set of intermediate variables with $|\mathcal{I}| \leqslant t_1$, such that $\mathcal{I}^1$ corresponds to the observations of intermediate variables done by the attacker in MatrixRows, and $\mathcal{I}^2$ corresponds to the observations in MatrixMult. Let $\mathcal{O}$ be a set of indices such that $|\mathcal{O}| \leqslant t_2$ corresponds to the observations on the outputs made by the attacker.

Gadget MatrixRows**.** By assumption, we have $\left|\mathcal{I}^1\right| + |\mathcal{O}| \leqslant t_1 + t_2 < n$. Therefore, Lemma 10 ensures the existence of a set of indices $\mathcal{S}^1 = (E \times E) \cup A$ such that $|E| + |A| \leqslant \left|\mathcal{I}^1\right|$, and this gadget can be perfectly simulated from its input shares corresponding to indices in $\mathcal{S}^1$.

Gadget MatrixMult**.** From Step 1, we have $|E| + |A| \leqslant \left|\mathcal{I}^1\right|$, which gives $|E| + |A| + \left|\mathcal{I}^2\right| \leqslant \left|\mathcal{I}^1\right| + \left|\mathcal{I}^2\right| \leqslant t_1 < n$. Therefore, Lemma 9 ensures the existence of two sets of indices I and J such that $|I| \leqslant \left|\mathcal{I}^2\right| + |E| + |A|$ and $|J| \leqslant \left|\mathcal{I}^2\right| + |E| + |A|$, and Gadget MatrixMult can be perfectly simulated from its input shares corresponding to indices in I and J.

From Steps 1 and 2, it follows that $|I| \leqslant \left|\mathcal{I}^2\right| + \left|\mathcal{I}^1\right| \leqslant |\mathcal{I}| \leqslant t_1$ and similarly $|J| \leqslant t_1$. This concludes the proof of Lemma 1.

B.3 Proof of Lemma 2

We can use the same modular approach of Section B.2. For RefreshMask the matrix step simply consists in letting $v_{ii} = a_i$ for all $1 \leqslant i \leqslant n$ and $v_{ij} = 0$ for all $i \neq j$. The proof of Lemma 2 then follows from Lemma 10.

C A CommonMult Algorithm with Memory $\mathcal{O}(n)$

We describe in Algorithm 12 a version of the CommonMult algorithm with memory complexity $\mathcal{O}(n)$ instead of $\mathcal{O}(n^2)$.

Algorithm 12 CommonMult

Input: shares satisfying $c = \bigoplus_{i=1}^{n} c_i$, $a = \bigoplus_{i=1}^{n} a_i$, $b = \bigoplus_{i=1}^{n} b_i$
Output: d_i such that $\bigoplus_{i=1}^{n} d_i = c \cdot a$, and e_i such that $\bigoplus_{i=1}^{n} e_i = c \cdot b$

$(a_i)_{1 \leqslant i \leqslant n}, (b_i)_{1 \leqslant i \leqslant n} \leftarrow$ CommonShares$((a_i)_{1 \leqslant i \leqslant n}, (b_i)_{1 \leqslant i \leqslant n})$ ▷ Ensures that $a_i = b_i$ for all $1 \leqslant i \leqslant n/2$
for $i = 1$ to n **do**
    $d_i \leftarrow c_i \cdot a_i$, $e_i \leftarrow c_i \cdot b_i$
end for
for $i = 1$ to n **do**
    for $j = i + 1$ to n **do**
        $r \leftarrow^{\$} \mathbb{F}_{2^k}$, $r' \leftarrow^{\$} \mathbb{F}_{2^k}$ ▷ referred by r_{ij} and r'_{ij}
        $d_i \leftarrow d_i \oplus r$, $e_i \leftarrow e_i \oplus r'$ ▷ referred by d_{ij} and e_{ij}
        $v \leftarrow c_i \cdot a_j$, $w \leftarrow c_j \cdot a_i$ ▷ referred by v_{ij} and v_{ji}
        if $j \leqslant n/2$ **then**
            $v' \leftarrow v$, $w' \leftarrow w$
        else if $i \leqslant n/2$ **then**
            $v' \leftarrow c_i \cdot b_j$, $w' \leftarrow w$
        else
            $v' \leftarrow c_i \cdot b_j$, $w' \leftarrow c_j \cdot b_i$ ▷ referred by v'_{ij} and v'_{ji}
        end if
        $r \leftarrow (r \oplus v) \oplus w$, $r' \leftarrow (r' \oplus v') \oplus w'$ ▷ referred by r_{ji} and r'_{ji}
        $c_j \leftarrow c_j \oplus r$, $c'_j \leftarrow c'_j \oplus r'$ ▷ referred by c_{ji} and c'_{ji}
    end for
end for
return $(d_i)_{1 \leqslant i \leqslant n}$ and $(e_i)_{1 \leqslant i \leqslant n}$

D Proof of Lemma 6 (t-SNI of SecExp254')

The proof is easily deduced from the security of SecMult (Lemma 1), RefreshMask (Lemma 2) and CommonShares (Lemma 4). As illustrated in Fig. 9, thanks to the t-SNI property of the SecMult and the RefreshMask algorithms, although a single probe within a SecMult can generate two different indices i and j on its two inputs, those 2 indices will never merge into the same variable, which means that we get security with $n \geqslant t+1$ shares instead of $n \geqslant 2t+1$ shares (as in the original ISW countermeasure). For example a single probe in the $\mathcal{I}^8$ SecMult gadget can generate two indices $i \in \mathcal{S}_1^8$ and $j \in \mathcal{S}_2^8$, but the index j is "blocked" by the $\mathcal{I}^{10}$ RefreshMask and therefore only i will appear in the input shares of x.

We follow the process described in [BBD+15a, Sec 4.1]. Note that except for CommonShares, all the gadgets involved in the computation of x^{254} are either t-SNI or linear. In addition, we use Lemma 4 to prove that the composition with the CommonShares gadget allows the entire circuit to be t-SNI. We label the gadgets from 1 to 10 starting from the last multiplication and progressing from right to left (see Figure 9).

Let $\mathcal{I}$ be a set of indices such that $|\mathcal{I}| \leqslant t_1$ (corresponding to observations of intermediate variables done by the attacker) and let $\mathcal{O}$ be a set of indices such that $|\mathcal{O}| \leqslant t_2$ (corresponding to observations on the outputs made by the attacker).

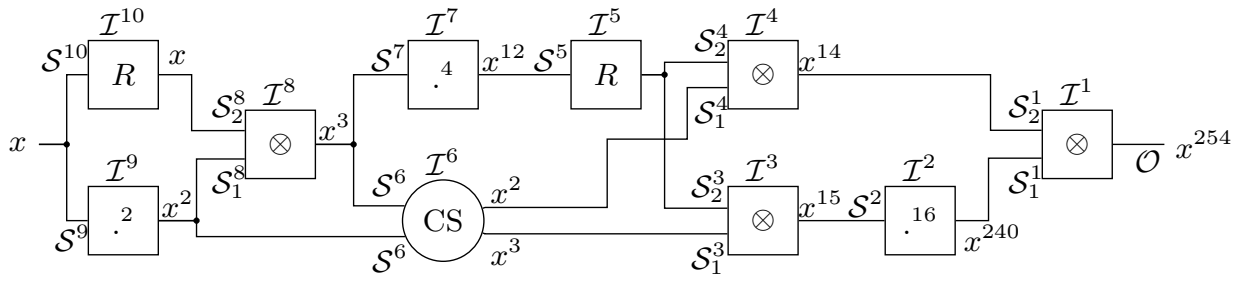

Fig. 9. x^{254} as composition of subgadgets

Consider a partition of $\mathcal{I} = \bigcup_{i=1}^{10} \mathcal{I}^i$, depending on the subgadget in which observation occurs (see Figure 9).

Gadget 1 By assumption, $\left|\mathcal{I}^1 \cup \mathcal{O}\right| \leqslant t_1 + t_2 \leqslant t$. Since from Lemma 1, the multiplication $\otimes$ is t-SNI, this means that there exist two sets of indices $\mathcal{S}_1^1, \mathcal{S}_2^1$ such that $\left|\mathcal{S}_1^1\right| \leqslant \left|\mathcal{I}^1\right|$, $\left|\mathcal{S}_2^1\right| \leqslant \left|\mathcal{I}^1\right|$ and $\otimes$ is $\left(\left(\mathcal{S}_1^1, \mathcal{S}_2^1\right), \left(\mathcal{I}^1, \mathcal{O}\right)\right)$-NI. In particular, this means that this gadget can be perfectly simulated from its input shares corresponding to indices in $\mathcal{S}_1^1$ and $\mathcal{S}_2^1$.

Gadget 2 As composition of squarings over a field of characteristic 2, the gadget $.^{16}$ is affine. Hence from [BBD+15a, Theorem 1], there exists a set of indices $\mathcal{S}^2$ such that $\left|\mathcal{S}^2\right| \leqslant \left|\mathcal{I}^2\right| + \left|\mathcal{S}_1^1\right|$ and $.^{16}$ is $\left(\mathcal{S}^2, \left(\mathcal{I}^2, \mathcal{S}_1^1\right)\right)$-NI. Note that by the previous step, $\left|\mathcal{S}_1^1\right| \leqslant \left|\mathcal{I}^1\right|$, so that $\left|\mathcal{S}^2\right| \leqslant \left|\mathcal{I}^2\right| + \left|\mathcal{I}^1\right|$.

Gadget 3 From the previous step, $\left|\mathcal{I}^3 \cup \mathcal{S}^2\right| \leqslant \left|\mathcal{I}^3\right| + \left|\mathcal{I}^2\right| + \left|\mathcal{I}^1\right| \leqslant t$. Since $\otimes$ is t-SNI, it follows that there exist two sets of indices $\mathcal{S}_1^3, \mathcal{S}_2^3$ such that $\left|\mathcal{S}_1^3\right| \leqslant \left|\mathcal{I}^3\right|$, $\left|\mathcal{S}_2^3\right| \leqslant \left|\mathcal{I}^3\right|$ and $\otimes$ is $\left(\left(\mathcal{S}_1^3, \mathcal{S}_2^3\right), \left(\mathcal{I}^3, \mathcal{S}^2\right)\right)$-NI.

Gadget 4 From Step 1, $\left|\mathcal{I}^4 \cup \mathcal{S}_2^1\right| \leqslant \left|\mathcal{I}^4\right| + \left|\mathcal{I}^1\right| \leqslant t$. Since $\otimes$ is t-SNI, this implies the existence of two sets of indices $\mathcal{S}_1^4, \mathcal{S}_2^4$ such that $\left|\mathcal{S}_1^4\right| \leqslant \left|\mathcal{I}^4\right|$, $\left|\mathcal{S}_2^4\right| \leqslant \left|\mathcal{I}^4\right|$ and $\otimes$ is $\left(\left(\mathcal{S}_1^4, \mathcal{S}_2^4\right), \left(\mathcal{I}^4, \mathcal{S}_2^1\right)\right)$-NI.

Gadget 5 It follows from Steps 3 and 4 that $\left|\mathcal{I}^5 \cup \left(\mathcal{S}_2^3 \cup \mathcal{S}_2^4\right)\right| \leqslant \left|\mathcal{I}^5\right| + \left|\mathcal{I}^3\right| + \left|\mathcal{I}^4\right| \leqslant t$. Since RefreshMask is t-SNI by Lemma 2, there exists a set of indices $\mathcal{S}^5 \leqslant \mathcal{I}^5$ such that gadget R is $\left(\mathcal{S}^5, \left(\mathcal{I}^5, \mathcal{S}_2^3 \cup \mathcal{S}_2^4\right)\right)$-NI.

Gadget 6 From Steps 3 and 4, $\left|\mathcal{I}^6 \cup \mathcal{S}_1^3 \cup \mathcal{S}_1^4\right| \leqslant \left|\mathcal{I}^6\right| + \left|\mathcal{I}^3\right| + \left|\mathcal{I}^4\right| \leqslant t \leqslant n-1$. Then Lemma 4 applies and ensures the existence of a set of indices $\mathcal{S}^6$ such that $\left|\mathcal{S}^6\right| \leqslant \left|\mathcal{S}_1^3\right| + \left|\mathcal{S}_1^4\right| + \left|\mathcal{I}^6\right|$. From Steps 3 and 4, it follows that $\left|\mathcal{S}^6\right| \leqslant \left|\mathcal{I}^3\right| + \left|\mathcal{I}^4\right| + \left|\mathcal{I}^6\right|$.

Gadget 7 Since gadget $.^4$ is affine, there exists a set of indices $\mathcal{S}^7$ such that $\left|\mathcal{S}^7\right| \leqslant \left|\mathcal{I}^7\right| + \left|\mathcal{S}^5\right|$ and $.^4$ is $\left(\mathcal{S}^7, \left(\mathcal{I}^7, \mathcal{S}^5\right)\right)$-NI. Note that by Step 5, $\left|\mathcal{S}^5\right| \leqslant \left|\mathcal{I}^5\right|$, so that $\left|\mathcal{S}^7\right| \leqslant \left|\mathcal{I}^7\right| + \left|\mathcal{I}^5\right|$.

Gadget 8 From Step 6 and 7, $\left|\mathcal{I}^8 \cup \left(\mathcal{S}^6 \cup \mathcal{S}^7\right)\right| \leqslant \left|\mathcal{I}^8\right| + \left|\mathcal{I}^3\right| + \left|\mathcal{I}^4\right| + \left|\mathcal{I}^5\right| + \left|\mathcal{I}^6\right| + \left|\mathcal{I}^7\right| \leqslant t$. Since $\otimes$ is t-SNI, this implies the existence of two sets of indices $\mathcal{S}_1^8, \mathcal{S}_2^8$ such that $\left|\mathcal{S}_1^8\right| \leqslant \left|\mathcal{I}^8\right|$, $\left|\mathcal{S}_2^8\right| \leqslant \left|\mathcal{I}^8\right|$ and $\otimes$ is $\left(\left(\mathcal{S}_1^8, \mathcal{S}_2^8\right), \left(\mathcal{I}^8, \mathcal{S}^6 \cup \mathcal{S}^7\right)\right)$-NI.

Gadget 9 Since gadget $.^2$ is affine, there exists a set of indices $\mathcal{S}^9$ such that $\left|\mathcal{S}^9\right| \leqslant \left|\mathcal{I}^9\right| + \left|\mathcal{S}_1^8 \cup \mathcal{S}^6\right|$ and $.^2$ is $\left(\mathcal{S}^9, \left(\mathcal{I}^9, \mathcal{S}_1^8 \cup \mathcal{S}^6\right)\right)$-NI. Note that by Steps 6 and 8, $\left|\mathcal{S}_1^8 \cup \mathcal{S}^6\right| \leqslant \left|\mathcal{I}^8\right| + \left|\mathcal{I}^3\right| + \left|\mathcal{I}^4\right| + \left|\mathcal{I}^6\right|$, so that $\left|\mathcal{S}^9\right| \leqslant \left|\mathcal{I}^9\right| + \left|\mathcal{I}^8\right| + \left|\mathcal{I}^3\right| + \left|\mathcal{I}^4\right| + \left|\mathcal{I}^6\right|$.

Gadget 10 From Step 8, $\left|\mathcal{I}^{10} \cup \mathcal{S}_2^8\right| \leqslant \left|\mathcal{I}^{10}\right| + \left|\mathcal{I}^8\right| \leqslant t$. Since RefreshMask is t-SNI, there exists a set of indices $\mathcal{S}^{10}$ such that $\left|\mathcal{S}^{10}\right| \leqslant \left|\mathcal{I}^{10}\right|$ such that gadget R is $\left(\mathcal{S}^{10}, \left(\mathcal{I}^{10}, \mathcal{S}_2^8\right)\right)$-NI.

Each of the previous steps ensures the existence of a simulator for each gadget. Let $I = \mathcal{S}^9 \cup \mathcal{S}^{10}$. We can then compose these simulators to simulate perfectly the computation of x^{254} from $a_{|I}$.

Furthermore, from Steps 9 and 10 we know that

$$|I| = \left|\mathcal{S}^9 \cup \mathcal{S}^{10}\right| \leqslant \left|\mathcal{I}^9\right| + \left|\mathcal{I}^8\right| + \left|\mathcal{I}^3\right| + \left|\mathcal{I}^4\right| + \left|\mathcal{I}^6\right| + \left|\mathcal{I}^{10}\right| \leqslant t_1,$$

which concludes the proof.

E Secure Computation of Parallel Multiplications

E.1 Secure Computation of m Parallel Multiplications

We describe the parallel computation of m multiplications of $c^{(j)} = a^{(j)} \cdot b^{(j)}$, using an n-sharing of $a^{(j)}$ and $b^{(j)}$. We obtain the ParaMult algorithm (Alg. 13). To ensure that m variables have the same $n/2$ shares, we use a geneneralization of the CommonShares algorithm; see Algorithm GeneralizedCommonShares (Alg. 14).

Algorithm 13 ParaMult

Input: Variables $a^{(j)}$ for $0 \leqslant j \leqslant m-1$ with shares $(a_i^{(j)})_{1\leqslant i\leqslant n}$ satisfying $\bigoplus_{i=1}^n a_i^{(j)} = a^{(j)}$, variables $b^{(j)}$ for $0 \leqslant j \leqslant m-1$ with shares $(b_i^{(j)})_{1\leqslant i\leqslant n}$ satisfying $\bigoplus_{i=1}^n b_i^{(j)} = b^{(j)}$,

Output: Variables $c^{(j)}$ for $0 \leqslant j \leqslant m-1$ with shares $(c_i^{(j)})_{1\leqslant i\leqslant n}$ satisfying $\bigoplus_{i=1}^n c_i^{(j)} = a^{(j)} \cdot b^{(j)}$.

1: $(a_i^{(j)}) \leftarrow \mathsf{GeneralizedCommonShares}(a_i^{(j)})$
2: $(b_i^{(j)}) \leftarrow \mathsf{GeneralizedCommonShares}(b_i^{(j)})$
3: **for** $j = 0$ **to** $m-1$ **do**
4: $\quad (c_i^{(j)}) \leftarrow \mathsf{SecMult}(a_i^{(j)}, b_i^{(j)})$
5: **end for**
6: **return** $(c_i^{(j)})_{1\leqslant i\leqslant n}$ for $0 \leqslant j < m$

Algorithm 14 GeneralizedCommonShares

Input: Variables $a^{(j)}$ for $0 \leqslant j \leqslant m-1$ with shares $(a_i^{(j)})_{1\leqslant i\leqslant n}$ satisfying $\bigoplus_{i=1}^n a_i^{(j)} = a^{(j)}$

Output: Variables $b^{(j)}$ for $0 \leqslant j \leqslant m-1$ with shares $(b_i^{(j)})_{1\leqslant i\leqslant n}$ satisfying $\bigoplus_{i=1}^n b_i^{(j)} = a^{(j)}$ and $b_i^{(0)} = b_i^{(1)} = \cdots = b_i^{(m-1)}$ for all $1 \leqslant i \leqslant n/2$

1: **for** $i = 1$ **to** $n/2$ **do**
2: $\quad r_i \leftarrow^{\$} \mathbb{F}_{2^k}$
3: $\quad$ **for** $j = 0$ **to** $m-1$ **do**
4: $\quad\quad b_i^{(j)} \leftarrow r_i$
5: $\quad\quad b_{n/2+i}^{(j)} \leftarrow (a_{n/2+i}^{(j)} \oplus r_i) \oplus a_i^{(j)}$
6: $\quad$ **end for**
7: **end for**
8: **return** $(b_i^{(j)})_{1\leqslant i\leqslant n}$ for $0 \leqslant j < m$

The following Lemma shows that the GeneralizedCommonShares algorithm is secure in the ISW model, with $n \geqslant t+1$.

Lemma 11 (t-NI of GeneralizedCommonShares). *Let $(a_i^{(j)})_{0\leqslant i<n}$ for $0 \leqslant j < m$ be the input shares of the algorithm GeneralizedCommonShares, and let $(b_i^{(j)})_{0\leqslant i<n}$ for $0 \leqslant j < m$ be the output shares.*

For any set of t intermediate variables and any subsets of indices $I^0, I^1, \ldots, I^{m-1} \subset [0, n-1]$ such that $t + \sum_{j=0}^{m-1} |I^j| \leqslant n-1$, there exists a subset $\mathcal{S} \subset [0, n-1]$ such that those t variables, the output shares $b^{(j)}_{|I^j}$ for $0 \leqslant j < m$ can be perfectly simulated from $a^{(j)}_{|\mathcal{S}}$ for $0 \leqslant j < m$, with $|\mathcal{S}| \leqslant t + \sum_{j=0}^{m-1} |I^j|$.

Proof. The proof is very similar to the proof of Lemma 4. The construction of the set $\mathcal{S}$ of indices is done as follows: we add i to $\mathcal{S}$ for every probed input variable $a_i^{(j)}$ (for any $i < n$ and $j < m$), and for all the other variables we add indices i and $n/2+i$ to $\mathcal{S}$ if $\lambda_i \geqslant 2$, where $\lambda_i = t_i + \sum_{j=0}^{m-1} |\{i, n/2+i\} \cap I^j|$ and t_i is the number of probed intermediate variables $a^{(j)}_{n/2+i} \oplus r_i$ for $0 \leqslant j < m$.

Then the simulation from $a^{(j)}_{|\mathcal{S}}$ for $0 \leqslant j < m$ of the intermediate and output variables is performed as in the proof of Lemma 4. Namely, every probed input variable $a_i^{(j)}$ (for any $i < n$ and $j < m$) is perfectly simulated since by construction we have $i \in \mathcal{S}$. Moreover, if $\lambda_i \geqslant 2$ then by construction the indices i and $n/2+i$ belong to $\mathcal{S}$ and we can simulate all output and intermediate variables as they would have been computed in Algorithm GeneralizedCommonShares, by assigning to r_i a uniformly random value. Eventually, if $\lambda_i = 1$, this means that only one variable is probed among the intermediate variables $a^{(j)}_{n/2+i} \oplus r_i$ and the output variables $b_i^{(j)}$ and $b^{(j)}_{n/2+i}$ (for all $j < m$), and we can treat the probed variable as a uniformly random and independent value.

Therefore, any $(t + \sum_{j=0}^{m-1} |I^j|)$-tuple of variables of GeneralizedCommonShares can be perfectly simulated from $a^{(j)}_{|\mathcal{S}}$ for $0 \leqslant j < m$. Note that we have $|\mathcal{S}| \leqslant t + \sum_{j=0}^{m-1} |I^j|$ for the same reason as for Lemma 4. □

Lemma 12 (t-SNI of ParaMult). *Let $(a_i^{(j)})_{0 \leqslant i < n}$ and $(b_i^{(j)})_{0 \leqslant i < n}$ for $0 \leqslant j < m$ be the input shares of the ParaMult operation, and let $(c_i^{(j)})_{0 \leqslant i < n}$ for $0 \leqslant j < m$ be the output shares. For any set of t_1 intermediate variables and any subsets $|\mathcal{O}^j| \leqslant t_2$ of output shares such that $t_1 + t_2 < n$, there exists two subsets I and J of indices such that both $|I| \leqslant t_1$ and $|J| \leqslant t_1$, and the distribution of those t_1 intermediate variables can be perfectly simulated from $a^{(j)}_{|I}$ and $b^{(j)}_{|J}$ for all $0 \leqslant j < m$. The output shares $c^{(j)}_{|\mathcal{O}^j}$ can also be perfectly simulated from $a^{(j)}_{|I}$ and $b^{(j)}_{|J}$ for all $0 \leqslant j < m$.*

Proof. We use Lemmas 1 and 11 to prove that the composition of the GeneralizedCommonShares gadgets with multiplications $\otimes$ allows the entire circuit ParaMult to be t-SNI. We label the gadgets from 0 to $m+1$ (see Figure 10).

Let $\mathcal{I} = \bigcup_{i=0}^{m+1} \mathcal{I}^i$ be a set of indices such that $|obsI| \leqslant t_1$ (corresponding to observations of intermediate variables done by the attacker in the $m+2$ gadgets) and let $\mathcal{O}^i$ for $0 \leqslant i < m$ be sets of indices such that $\left|\mathcal{O}^i\right| \leqslant t_2$ (corresponding to observations on the outputs made by the attacker in the m gadgets $\otimes$).

Gadgets i for $0 \leqslant i \leqslant m-1$ By assumption, $\left|\mathcal{I}^i \cup \mathcal{O}^i\right| \leqslant t_1 + t_2 \leqslant t$. Since from Lemma 1, the multiplication $\otimes$ is t-SNI, this means that there exist two sets of indices $\mathcal{S}_1^i, \mathcal{S}_2^i$ such that $\left|\mathcal{S}_1^i\right| \leqslant \left|\mathcal{I}^i\right|$, $\left|\mathcal{S}_2^i\right| \leqslant \left|\mathcal{I}^i\right|$ and $\otimes$ is $\left(\left(\mathcal{S}_1^i, \mathcal{S}_2^i\right), \left(\mathcal{I}^i, \mathcal{O}^i\right)\right)$-NI. In particular, this means that Gadgets i for $0 \leqslant i \leqslant m-1$ can be perfectly simulated from their input shares corresponding to indices in $\mathcal{S}_1^i$ and $\mathcal{S}_2^i$.

Gadget m From Step 1, we have $\left|\mathcal{I}^m \cup \bigcup_{i=0}^{m-1} \mathcal{S}_2^i\right| \leqslant \sum_{i=0}^{m} \left|\mathcal{I}^i\right| \leqslant t_1 < n$. Then Lemma 11 applies and ensures the existence of a set of indices $\mathcal{S}^m$ such that $|\mathcal{S}^m| \leqslant |\mathcal{I}^m| + \sum_{i=0}^{m-1} \left|\mathcal{S}_2^i\right| \leqslant \sum_{i=0}^{m} \left|\mathcal{I}^i\right|$ and Gadget m can be perfectly simulated from its input shares corresponding to indices in $\mathcal{S}^m$.

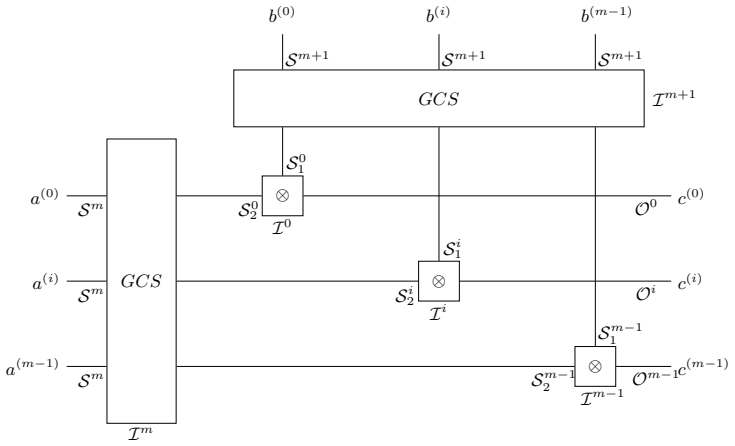

Fig. 10. The ParaMult algorithm as composition of gadgets. Each variable $a^{(j)}$ and $b^{(j)}$ contains actually n shares $a_i^{(j)}$ and $b_i^{(j)}$.

Gadget $\mathbf{m+1}$ From Step 1, we have $\left|\mathcal{I}^{m+1} \cup \bigcup_{i=0}^{m-1} \mathcal{S}_1^i\right| \leqslant \left|\mathcal{I}^{m+1}\right| + \sum_{i=0}^{m-1} \left|\mathcal{I}^i\right| \leqslant t_1 < n$. Then Lemma 11 applies and ensures the existence of a set of indices $\mathcal{S}^{m+1}$ such that $\left|\mathcal{S}^{m+1}\right| \leqslant \left|\mathcal{I}^{m+1}\right| + \sum_{i=0}^{m-1} \left|\mathcal{S}_1^i\right| \leqslant \left|\mathcal{I}^{m+1}\right| + \sum_{i=0}^{m-1} \left|\mathcal{I}^i\right|$ and Gadget $m+1$ can be perfectly simulated from its input shares corresponding to indices in $\mathcal{S}^{m+1}$.

Each of the previous steps ensures the existence of a simulator for each gadget. Let $I = \mathcal{S}^m$ and $J = \mathcal{S}^{m+1}$. We can then compose these simulators to perfectly simulate the computation of GeneralizedCommonShares from $a_{|I}^{(j)}$ and $b_{|J}^{(j)}$ for all $0 \leqslant j < m$. Furthermore, we have $|I| = |\mathcal{S}^m| \leqslant \sum_{i=0}^{m} \left|\mathcal{I}^i\right| \leqslant t_1$ and $|J| = \left|\mathcal{S}^{m+1}\right| \leqslant \left|\mathcal{I}^{m+1}\right| + \sum_{i=0}^{m-1} \left|\mathcal{I}^i\right| \leqslant t_1$.

E.2 Parallel Computation of $2m$ Multiplications with Common Operands

To perform the multiplications with same operand $x^{(j)} \leftarrow a^{(j)} \cdot b^{(j)}$ and $y^{(j)} \leftarrow a^{(j)} \cdot c^{(j)}$ for all $1 \leqslant j \leqslant m$, instead of doing $2m$ independent SecMult, we define the following ParaCommonMult algorithm.

1. Apply GeneralizedCommonShares to the m variables $a^{(j)}$
2. Apply GeneralizedCommonShares to the $2m$ variables $b^{(j)}$ and $c^{(j)}$
3. Apply SecMult to compute $x^{(j)} = a^{(j)} \cdot b^{(j)}$ and $y^{(j)} = a^{(j)} \cdot c^{(j)}$.

Lemma 13 (t-SNI of ParaCommonMult). *Let $(a_i^{(j)})_{0 \leqslant i < n}$ and $(b_i^{(j)})_{0 \leqslant i < n}$ and $(c_i^{(j)})_{0 \leqslant i < n}$ for $0 \leqslant j < m$ be the input shares of the ParaCommonMult operation, and let $(d_i^{(j)})_{0 \leqslant i < n}$ and $(e_i^{(j)})_{0 \leqslant i < n}$ for $0 \leqslant j < m$ be the output shares. For any set of t_1 intermediate variables and any subsets $|\mathcal{O}_1^j| \leqslant t_2$ and $|\mathcal{O}_2^j| \leqslant t_2$ of output shares such that $t_1 + t_2 < n$, there exists two subsets I and J of indices such that both $|I| \leqslant t_1$ and $|J| \leqslant t_1$, and the distribution of those t_1 intermediate variables can be perfectly simulated from $a_{|I}^{(j)}$, $b_{|J}^{(j)}$ and $c_{|J}^{(j)}$ for all $0 \leqslant j < m$. The output shares $d_{|\mathcal{O}_1^j}^{(j)}$ and $e_{|\mathcal{O}_2^j}^{(j)}$ can also be perfectly simulated from $a_{|I}^{(j)}$, $b_{|J}^{(j)}$ and $c_{|J}^{(j)}$ for all $0 \leqslant j < m$.*

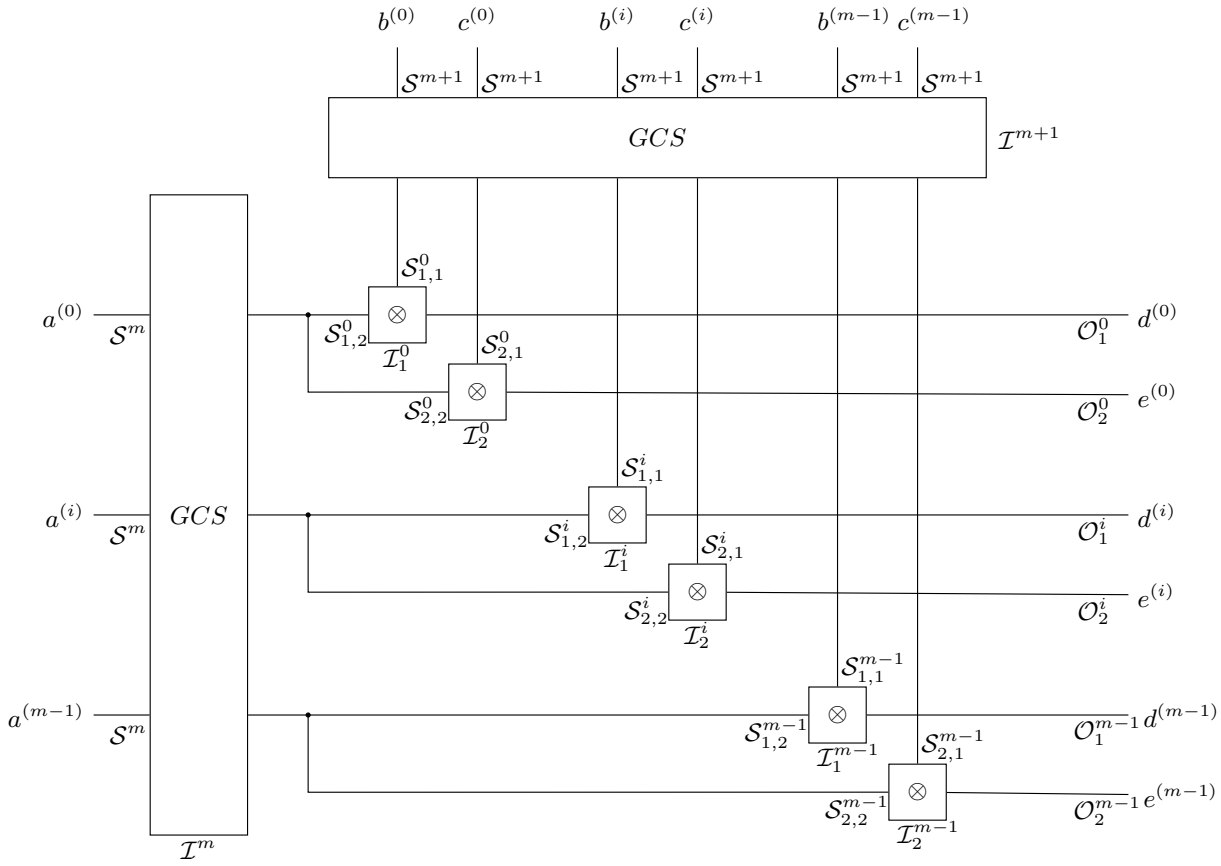

Fig. 11. The ParaCommonMult algorithm as composition of gadgets. Each variable $a^{(j)}$, $b^{(j)}$ and $c^{(j)}$ contains actually n shares $a_i^{(j)}$, $b_i^{(j)}$ and $c_i^{(j)}$.

Proof. We use Lemmas 1 and 11 to prove that the composition of the GeneralizedCommonShares gadgets with multiplication $\otimes$ gadgets allows the entire circuit ParaCommonMult to be t-SNI. We label two sets of gadgets from 0 to $m-1$ (one set with index 1 and the other with index 2) for multiplications, and two other labels m and $m+1$ for both GeneralizedCommonShares gadgets (see Figure 11).

Let $\mathcal{I} = \bigcup_{i=0}^{m-1}(\mathcal{I}_1^i \cup \mathcal{I}_2^i) \cup \mathcal{I}^m \cup \mathcal{I}^{m+1}$ be a set of indices such that $|\mathcal{I}| \leqslant t_1$ (corresponding to observations of intermediate variables done by the attacker in the $2m+2$ gadgets) and let $\mathcal{O}_1^i$ and $\mathcal{O}_2^i$ for $0 \leqslant i < m$ be sets of indices such that $\left|\mathcal{O}_1^i\right| \leqslant t_2$ and $\left|\mathcal{O}_2^i\right| \leqslant t_2$ (corresponding to observations on the outputs made by the attacker in the $2m$ gadgets $\otimes$).

Gadgets i for $0 \leqslant i \leqslant m-1$ with index 1 By assumption, $\left|\mathcal{I}_1^i \cup \mathcal{O}_1^i\right| \leqslant t_1 + t_2 \leqslant t$. Since from Lemma 1, the multiplication $\otimes$ is t-SNI, this means that there exist two sets of indices $\mathcal{S}_{1,1}^i$, $\mathcal{S}_{1,2}^i$ such that $\left|\mathcal{S}_{1,1}^i\right| \leqslant \left|\mathcal{I}_1^i\right|$, $\left|\mathcal{S}_{1,2}^i\right| \leqslant \left|\mathcal{I}_1^i\right|$ and $\otimes$ is $\left(\left(\mathcal{S}_{1,1}^i, \mathcal{S}_{1,2}^i\right), \left(\mathcal{I}_1^i, \mathcal{O}_1^i\right)\right)$-NI. In particular, this means that Gadgets i with index 1, for $0 \leqslant i \leqslant m-1$ can be perfectly simulated from their input shares corresponding to indices in $\mathcal{S}_{1,1}^i$ and $\mathcal{S}_{1,2}^i$.

Gadgets i for $0 \leqslant i \leqslant m-1$ with index 2 Similarly we have $\left|\mathcal{I}_2^i \cup \mathcal{O}_2^i\right| \leqslant t_1 + t_2 \leqslant t$. Since from Lemma 1, the multiplication $\otimes$ is t-SNI, this means that there exist two sets of indices $\mathcal{S}_{2,1}^i$, $\mathcal{S}_{2,2}^i$ such that $\left|\mathcal{S}_{2,1}^i\right| \leqslant \left|\mathcal{I}_2^i\right|$, $\left|\mathcal{S}_{2,2}^i\right| \leqslant \left|\mathcal{I}_2^i\right|$ and $\otimes$ is $\left(\left(\mathcal{S}_{2,1}^i, \mathcal{S}_{2,2}^i\right), \left(\mathcal{I}_2^i, \mathcal{O}_2^i\right)\right)$-NI. In particular, this means that Gadgets i with index 2, for $0 \leqslant i \leqslant m-1$ can be perfectly simulated from their input shares corresponding to indices in $\mathcal{S}_{2,1}^i$ and $\mathcal{S}_{2,2}^i$.

Gadget m From Step 1, we have $\left|\mathcal{I}^m \cup \bigcup_{i=0}^{m-1}(\mathcal{S}_{1,2}^i \cup \mathcal{S}_{2,2}^i)\right| \leqslant |\mathcal{I}^m| + \sum_{i=0}^{m-1}(\left|\mathcal{I}_1^i\right| + \left|\mathcal{I}_2^i\right|) \leqslant t_1 < n$. Then Lemma 11 applies and ensures the existence of a set of indices $\mathcal{S}^m$ such that $|\mathcal{S}^m| \leqslant |\mathcal{I}^m| + \sum_{i=0}^{m-1}(\left|\mathcal{S}_{1,2}^i\right| + \left|\mathcal{S}_{2,2}^i\right|) \leqslant |\mathcal{I}^m| + \sum_{i=0}^{m-1}(\left|\mathcal{I}_1^i\right| + \left|\mathcal{I}_2^i\right|)$ and Gadget m can be perfectly simulated from its input shares corresponding to indices in $\mathcal{S}^m$.

Gadget m+1 From Step 1, we have $\left|\mathcal{I}^{m+1} \cup \bigcup_{i=0}^{m-1}(\mathcal{S}_{1,1}^i \cup \mathcal{S}_{2,1}^i)\right| \leqslant \left|\mathcal{I}^{m+1}\right| + \sum_{i=0}^{m-1}(\left|\mathcal{I}_1^i\right| + \left|\mathcal{I}_2^i\right|) \leqslant t_1 < n$. Then Lemma 11 applies and ensures the existence of a set of indices $\mathcal{S}^{m+1}$ such that $\left|\mathcal{S}^{m+1}\right| \leqslant \left|\mathcal{I}^{m+1}\right| + \sum_{i=0}^{m-1}(\left|\mathcal{S}_{1,1}^i\right| + \left|\mathcal{S}_{2,1}^i\right|) \leqslant \left|\mathcal{I}^{m+1}\right| + \sum_{i=0}^{m-1}(\left|\mathcal{I}_1^i\right| + \left|\mathcal{I}_2^i\right|)$ and Gadget $m+1$ can be perfectly simulated from its input shares corresponding to indices in $\mathcal{S}^{m+1}$.

Each of the previous steps ensures the existence of a simulator for each gadget. Let $I = \mathcal{S}^m$ and $J = \mathcal{S}^{m+1}$. We can then compose these simulators to perfectly simulate the computation of ParaCommonMult from $a_{|I}^{(j)}$, $b_{|J}^{(j)}$ and $c_{|J}^{(j)}$ for all $0 \leqslant j < m$. Furthermore, we have $|I| = |\mathcal{S}^m| \leqslant |\mathcal{I}^m| + \sum_{i=0}^{m-1}(\left|\mathcal{I}_1^i\right| + \left|\mathcal{I}_2^i\right|) \leqslant t_1$ and $|J| = \left|\mathcal{S}^{m+1}\right| \leqslant \left|\mathcal{I}^{m+1}\right| + \sum_{i=0}^{m-1}(\left|\mathcal{I}_1^i\right| + \left|\mathcal{I}_2^i\right|) \leqslant t_1$.

□

F Evaluation of a Quadratic Function

F.1 Security Proof of the QuadraticEval Algorithm

In this section we prove that the previous QuadraticEval algorithm achieves the t-SNI property. As with our proof of SecMult in Appendix B.2, we use a modular approach, in which we first analyze the matrix construction algorithm MatrixQuadratic (Alg. 15); one can then apply the t-SNI property of the second layer, MatrixRows (Alg. 11), which is the same as for SecMult in Appendix B.2.

Algorithm 15 MatrixQuadratic

Input: $x_1, \ldots, x_n$ and a quadratic function h
Output: v_{ij} such that $\oplus_{i,j} v_{ij} = h(\oplus_i x_i)$

for $i = 1$ to n **do**
  $v_{ii} \leftarrow h(x_i)$
  for $j = i + 1$ to n **do**
    $s_{ij} \leftarrow^{\$} \mathbb{F}_{2^k}$
    $v_{ij} \leftarrow h(x_i + s_{ij}) + h((x_i + s_{ij}) + x_j)$
    $v_{ji} \leftarrow h(x_j + s_{ij}) + h(s_{ij})$
  end for
end for
if d is even **then**
  $v_{11} \leftarrow v_{11} + h(0)$
end if
return (v_{ij})

Lemma 14 (t-NI of MatrixQuadratic). *Let $(x_i)_{1 \leqslant i \leqslant n}$ be the input shares of the* MatrixQuadratic *operation. Let $\mathcal{O}$ be a subset of the output shares such that $\mathcal{O} = E \times E \cup \mathcal{O}_2$ where $|E| + |\mathcal{O}_2| \leqslant t_2$. Then for any set of t_1 intermediate variables, there exists a subset I of indices with $|I| \leqslant t_1 + t_2$, such that the t_1 intermediate variables as well as the output shares $(v_{ij})_{(i,j)\in\mathcal{O}}$ can be perfectly simulated from $x_{|I}$.*

Proof. We describe hereafter the construction of the set I of indices, initially empty. We divide the probed variables into 3 groups. Those groups are processed separately and sequentially, for all pairs $i < j$.

- Group 1: For every probed variable s_{ij}, x_i, $x_i + s_{ij}$, or h applied to these two last variables, add i in I.
- Group 2: For every probed v_{ij}, v_{ji}, $x_j + s_{ij}$ or $(x_i + s_{ij}) + x_j$, or h applied to these two last variables, if $j \in I$ add i to I, otherwise, add j to I.
- Group 3: For every pair $(i, j) \in \mathcal{O}$ or $(j, i) \in \mathcal{O}$ with $j > i$, if $j \in I$ add i to I, otherwise, add j to I.

Now that the set I has been determined, we show that the t_1 intermediate variables and the output shares $(v_{ij})_{(i,j)\in\mathcal{O}}$ of Algorithm MatrixQuadratic can be perfectly simulated from $x_{|I}$. We distinguish four cases depending on whether i and/or j (with $i < j$) belong to I or not.

- $\{i, j\} \in I$: All variables can straightforwardly be simulated since one has access to x_i and x_j.
- $i \in I$ and $j \notin I$: The simulation of variables from Group 1 is straightforward since one has access to x_i (because $i \in I$), by assigning to s_{ij} a uniformly random value as in the real circuit. Furthermore, variables from Group 2 and 3 are not probed, since otherwise we would have $j \in I$.
- $i \notin I$ and $j \in I$: By construction, this means that none of the variables from Group 1 is probed and that only one variable from Group 2 and 3 is probed. If it is the output variable v_{ji} (probed or such that $(j, i) \in \mathcal{O}$) or $x_j + s_{ij}$, it can be perfectly simulated since one has access to x_j, and s_{ij} is assigned a random value. If it is v_{ij} (probed or such that $(i, j) \in \mathcal{O}$), or $(h)((x_i + s_{ij}) + x_j)$ then it can be simulated since one has access to x_j, by treating $x_i + s_{ij}$ as a random variable since s_{ij} is not probed and not used anywhere else.
- $i \notin I$ and $j \notin I$: No variable among Group 1, 2 and 3 is probed.

Note that we have $|I| \leqslant t_1 + t_2$ since for each intermediate variable and for each output shares such that $(i,j) \in \mathcal{O}$, by construction at most one index was added to I. This shows that the t_1 intermediate variables and the output shares $(v_{ij})_{(i,j)\in\mathcal{O}}$ of Algorithm MatrixQuadratic can be perfectly simulated from $x_{|I}$. □

Combined with Lemma 10, this proves the following lemma:

Lemma 15 (t-SNI of QuadraticEval). *Let $(a_i)_{1\leqslant i\leqslant n}$ be the input shares of* QuadraticEval*, and let $(c_i)_{1\leqslant i\leqslant n}$ be the output shares. For any set of t_1 intermediate variables and any subset $|\mathcal{O}| \leqslant t_2$ of output shares such that $t_1 + t_2 < n$, there exists a subset I of indices with $|I| \leqslant t_1$, such that the t_1 intermediate variables as well as the output shares $c_{|\mathcal{O}}$ can be perfectly simulated from $a_{|I}$.*

Proof. We show that by combining lemmas 14 and 10, the composition of the MatrixRows and MatrixQuadratic algorithms allows Algorithm QuadraticEval to be t-SNI, which proves Lemma 15. Let $\mathcal{I} = \mathcal{I}^1 \cup \mathcal{I}^2$ be a set of indices such that $|\mathcal{I}| \leqslant t_1$ corresponds to observations of intermediate variables done by the attacker in both gadgets MatrixRows ($\mathcal{I}^1$) and MatrixQuadratic ($\mathcal{I}^2$), and let $\mathcal{O}$ be a set of indices such that $|\mathcal{O}| \leqslant t_2$ corresponds to observations on the outputs made by the attacker.

Gadget MatrixRows. By assumption, we have $\left|\mathcal{I}^1 \cup \mathcal{O}\right| \leqslant t_1 + t_2 < n$. Therefore, Lemma 10 ensures the existence of a set of indices $\mathcal{S}^1 = E \times E \cup A$ such that $|E| + |A| \leqslant \left|\mathcal{I}^1\right|$, and this gadget can be perfectly simulated from its input shares corresponding to indices in $\mathcal{S}^1$.

Gadget MatrixQuadratic. From Step 1, we have $|E| + |A| \leqslant \left|\mathcal{I}^1\right|$, which gives $|E| + |A| + \left|\mathcal{I}^2\right| \leqslant \left|\mathcal{I}^1\right| + \left|\mathcal{I}^2\right| \leqslant t_1 < n$. Therefore, Lemma 14 ensures the existence of one set of indices I such that $|I| \leqslant \left|\mathcal{I}^2\right| + |E| + |A|$ and Gadget MatrixQuadratic can be perfectly simulated from its input shares corresponding to indices in I.

From Steps 1 and 2, it follows that $|I| \leqslant \left|\mathcal{I}^2\right| + |E| + |A| \leqslant \left|\mathcal{I}^2\right| + \left|\mathcal{I}^1\right| \leqslant |I| \leqslant t_1$.

□

F.2 Proof of Lemma 8 (Algorithm TwoQuadraticEval)

We prove the t-SNI property of the TwoQuadraticEval algorithm, using the modular approach, by first considering the matrix construction step with TwoMatrixQuadratic.

Algorithm 16 TwoMatrixQuadratic

Input: a quadratic function h and x_i, x'_i, with $x_i = x'_i$ for all $1 \leqslant i \leqslant n/2$.
Output: v_{ij} such that $\oplus_{i,j} v_{ij} = h(\oplus_i x_i)$ and v'_{ij} such that $\oplus_{i,j} v'_{ij} = h(\oplus_i x'_i)$

for $i = 1$ to n **do**
  $v_{ii} \leftarrow h(x_i)$, $v'_{ii} \leftarrow h(x'_i)$
  for $j = i + 1$ to n **do**
    $s_{ij} \leftarrow^{\$} \mathbb{F}_{2^k}$
    $v_{ij} \leftarrow h(x_i + s_{ij}) + h((x_i + s_{ij}) + x_j)$, $v_{ji} \leftarrow h(x_j + s_{ij}) + h(s_{ij})$
    if $i \leqslant n/2$ and $j \leqslant n/2$ **then**
      $v'_{ij} \leftarrow v_{ij}$, $v'_{ji} \leftarrow v_{ji}$
    else
      $s'_{ij} \leftarrow^{\$} \mathbb{F}_{2^k}$
      $v'_{ij} \leftarrow h(x'_i + s'_{ij}) + h((x'_i + s'_{ij}) + x'_j)$, $v'_{ji} \leftarrow h(x'_j + s'_{ij}) + h(s'_{ij})$
    end if
  end for
end for
if d is even **then**
  $v_{11} \leftarrow v_{11} + h(0)$, $v'_{11} \leftarrow v'_{11} + h(0)$
end if
return (v_{ij}) and (v'_{ij})

Lemma 16 (t-NI of TwoMatrixQuadratic). *Let $(x_i)_{1\leqslant i\leqslant n}$ and $(x'_i)_{1\leqslant i\leqslant n}$ be the input shares of the* TwoMatrixQuadratic *operation. Let $\mathcal{O}$ and $\mathcal{O}'$ be subsets of the output shares such that $\mathcal{O} = E \times E \cup \mathcal{O}_2$ and $\mathcal{O}' = E' \times E' \cup \mathcal{O}'_2$ where $|E| + |\mathcal{O}_2| + |E'| + |\mathcal{O}'_2| \leqslant t_2$. Then for any set of t_1 intermediate variables, there exist two subsets I, I' of indices with $|I| + |I'| \leqslant t_1 + t_2$, such that the t_1 intermediate variables as well as the output shares $(v_{ij})_{(i,j)\in\mathcal{O}}$ and $(v'_{ij})_{(i,j)\in\mathcal{O}'}$ can be perfectly simulated from $x_{|I}$ and $x'_{|I'}$.*

Proof. We describe hereafter the construction of the sets I and I' of indices. We divide the internal probes into 6 groups. Those groups are processed separately and sequentially.

- Group 1: For every probed variable x_i, s_{ij}, $x_i + s_{ij}$, $x_i + s_{ji}$ for any $i \neq j$, or h applied to these variables, add i in I.
- Group 1': For every probed variable x'_i, s'_{ij}, $x'_i + s'_{ij}$, $x'_i + s'_{ji}$ for any $i \neq j$, or h applied to these variables, add i in I'.
- Group 2: For every probed v_{ij}, v_{ji} or $(h)((x_i + s_{ij}) + x_j)$ with $j > i$, or v'_{ij}, v'_{ji} with $i < j \leqslant n/2$: if $j \in I$ add i to I, otherwise, add j to I.
- Group 2': For every probed v'_{ij}, v'_{ji} or $(h)((x'_i + s'_{ij}) + x'_j)$ with $j > i$ and $j > n/2$, if $j \in I'$ add i to I', otherwise, add j to I'.
- Group 3: For every pair $(i, j) \in \mathcal{O}$ or $(j, i) \in \mathcal{O}$ with $j > i$, or every pair $(i, j) \in \mathcal{O}'$ or $(j, i) \in \mathcal{O}'$ with $i < j \leqslant n/2$, if $j \in I$ add i to I, otherwise, add j to I.
- Group 3': For every pair $(i, j) \in \mathcal{O}'$ or $(j, i) \in \mathcal{O}'$ with $j > i$ and $j > n/2$, if $j \in I'$ add i to I', otherwise, add j to I'.

Now that both sets I and I' have been determined, we show that the t_1 intermediate variables and the output shares $(v_{ij})_{(i,j)\in out}$ and $(v'_{ij})_{(i,j)\in\mathcal{O}'}$ of Algorithm TwoMatrixQuadratic can be perfectly simulated from $x_{|I}$ and $x'_{|I'}$. The simulation of variables from Group 1, 2 and 3 is performed identically as in Proof of Lemma 14 (by noting that variables v'_{ij} and v'_{ji} from Group 2 and 3 with $i < j \leqslant n/2$, probed or such that $(i, j) \in \mathcal{O}'$ or $(j, i) \in \mathcal{O}'$) are such that $v'_{ij} = v_{ij}$, $v'_{ji} = v_{ji}$

and are thus simulated as v_{ij} and v_{ji}). Furthermore, the simulation of variables from Group 1' and 2' is also performed in the same way, by considering I' instead of I. It remains to simulate the output variables from Group 3'. In this group, all variables are such that $j > i$ and $j > n/2$, and by construction we have $j \in I'$. We distinguish two cases depending on whether i belongs to I' or not. If $i \in I'$, then those output variables can straightforwardly be simulated since one has access to x'_i and x'_j. If $i \notin I'$, by construction, this means that s'_{ij} is not probed, and that only one variable among v'_{ij} and v'_{ji} has to be simulated. If it is v'_{ji} (probed or such that $(j,i) \in \mathcal{O}'$), it can be perfectly simulated since one has access to x'_j, and s'_{ij} is assigned a random value. If it is v'_{ij} (probed or such that $(i,j) \in \mathcal{O}'$), then it can be simulated since one has access to x'_j, by treating $x'_i + s'_{ij}$ as a random variable since s'_{ij} is not probed.

Note that we have $|I| + |I'| \leqslant t_1 + t_2$ since for each intermediate variable and for each output share such that $(i,j) \in \mathcal{O}$ or $(i,j) \in \mathcal{O}'$, by construction at most one index was added to I or I', and when probing a single variable, we never added an index simultaneously into I and I'.

We now proceed with the proof of Lemma 8. We show that by combining lemmas 4, 16 and 10, the composition of algorithms CommonShares, TwoMatrixQuadratic and MatrixRows allows Algorithm TwoQuadraticEval to be t-SNI, which would prove Lemma 8. Let $\mathcal{I} = \mathcal{I}^1 \cup \mathcal{I}^2 \cup \mathcal{I}^3 \cup \mathcal{I}^4$ be a set of indices such that $|\mathcal{I}| \leqslant t_1$ corresponds to observations of intermediate variables done by the attacker in the four gadgets $\mathsf{MatrixRows}_1$ ($\mathcal{I}^1$), $\mathsf{MatrixRows}_2$ ($\mathcal{I}^2$), TwoMatrixQuadratic ($\mathcal{I}^3$) and CommonShares ($\mathcal{I}^4$), and let $\mathcal{O}$ and $\mathcal{O}'$ be two sets of indices such that $|\mathcal{O}| \leqslant t_2$ and $\left|\mathcal{O}'\right| \leqslant t_2$ correspond to observations on the outputs made by the attacker.

Gadget $\mathsf{MatrixRows}_1$**.** By assumption, we have $\left|\mathcal{I}^1 \cup \mathcal{O}\right| \leqslant t_1 + t_2 < n$. Therefore, Lemma 10 ensures the existence of a set of indices $\mathcal{S}^1 = E \times E \cup A$ such that $|E| + |A| \leqslant \left|\mathcal{I}^1\right|$, and this gadget can be perfectly simulated from its input shares corresponding to indices in $\mathcal{S}^1$.

Gadget $\mathsf{MatrixRows}_2$**.** By assumption, we have $\left|\mathcal{I}^2 \cup \mathcal{O}'\right| \leqslant t_1 + t_2 < n$. Therefore, Lemma 10 ensures the existence of a set of indices $\mathcal{S}^2 = E' \times E' \cup A'$ such that $|E'| + |A'| \leqslant \left|\mathcal{I}^2\right|$, and this gadget can be perfectly simulated from its input shares corresponding to indices in $\mathcal{S}^2$.

Gadget TwoMatrixQuadratic**.** From Step 1 and 2, we have $|E| + |A| \leqslant \left|\mathcal{I}^1\right|$ and $|E'| + |A'| \leqslant \left|\mathcal{I}^2\right|$, which gives $|E| + |A| + |E'| + |A'| \leqslant \left|\mathcal{I}^1\right| + \left|\mathcal{I}^2\right| \leqslant t_1 < n$. Therefore, Lemma 16 ensures the existence of two sets of indices $\mathcal{S}_1^3$ and $\mathcal{S}_2^3$ such that $\left|\mathcal{S}_1^3\right| + \left|\mathcal{S}_2^3\right| \leqslant \left|\mathcal{I}^3\right| + |E| + |A| + |E'| + |A'|$ and Gadget TwoMatrixQuadratic can be perfectly simulated from its input shares corresponding to indices in $\mathcal{S}_1^3$ and $\mathcal{S}_2^3$.

Gadget CommonShares**.** From previous step, we have $\left|\mathcal{I}^4 \cup \mathcal{S}_1^3 \cup \mathcal{S}_2^3\right| \leqslant \left|\mathcal{I}^4\right| + \left|\mathcal{I}^3\right| + |E| + |A| + |E'| + |A'| \leqslant \left|\mathcal{I}^4\right| + \left|\mathcal{I}^3\right| + \left|\mathcal{I}^2\right| + \left|\mathcal{I}^1\right| \leqslant |\mathcal{I}| \leqslant t_1 < n$. Therefore, Lemma 4 ensures the existence of a set of indices I such that $|I| \leqslant \left|\mathcal{S}_1^3\right| + \left|\mathcal{S}_2^3\right| + \left|\mathcal{I}^4\right|$.

From Step 4, it follows that $|I| \leqslant \left|\mathcal{I}^4\right| + \left|\mathcal{I}^3\right| + |E| + |A| + |E'| + |A'| \leqslant \left|\mathcal{I}^4\right| + \left|\mathcal{I}^3\right| + \left|\mathcal{I}^2\right| + \left|\mathcal{I}^1\right| \leqslant |\mathcal{I}| \leqslant t_1$, which concludes the proof.

G Threshold Implementation

The soundness of masking/sharing against side-channel attacks relies on assumptions that do not necessarily hold in practice. For instance, software implementations may leak information about the transition between two states, which is typically visible in memory-element transitions. In this situation, ensuring that the manipulation of each share will leak independently may be tricky [CGP+12b].

Another well known illustration of the default of the probing security model has been given in [MPG05, MS06] where it is shown that *glitches*[5] enable successful attacks against theoretically sound masked implementations due to unsatisfactory leakage modeling. In a series of works, the concept of *Threshold Implementations* (TI) has been introduced (see *e.g.* [NRS11] and [BGN+14]). It aims at providing security developers with designs that are secure against (higher-order) side channel attacks, even in presence of glitches. Similarly as for Boolean masking, TI schemes start by splitting the function to protect into the composition of simple functions (*e.g.* field multiplications or quadratic functions). Then, the security scheme for the full function is deduced from the TI schemes dedicated to the smaller primitives.

In the rest of this section, it is argued that the ideas developed in this paper can be applied to save calculations when the TI scheme involved several multiplications with common shares (namely sharings having the first $\lfloor n/2 \rfloor$ shares in common). We start by recalling the principles of TI, then, following a similar approach as in [RBN+15], we recall the link between ISW scheme and TI implementations of the AND gate.

Threshold Implementation. Let f be a function defined from $\mathbb{F}_{2^k}$ into $\mathbb{F}_{2^m}$, and for every $x \in \mathbb{F}_{2^k}$ let $x[j]$ denote the jth-bit coordinate of x. A threshold implementation of a function $f(x)$ based on a sharing $\boldsymbol{x} = (x_1, ..., x_{s_{\text{in}}}) \in \mathbb{F}_{2^k}^{s_{\text{in}}}$ of $x \in \mathbb{F}_{2^k}$ consists in a family of s_{out} functions f_1, ..., $f_{s_{\text{out}}}$ such that the following properties are satisfied;

- **correctness:**
$$f(x) = \sum_{i \in [1..s_{\text{out}}]} f_i(\boldsymbol{x}) \tag{5}$$
- **non-completeness:** for every $i \in [1..s_{\text{out}}]$, there exists a function σ from $[1..k]$ into itself such that the algebraic description of $f_i(\boldsymbol{x})$ does not involve the coordinates $x_j[\sigma(j)]$ for j ranging over $[1..k]$ (which is denoted by $f_i(\boldsymbol{x}) \perp x_j[\sigma(j)]$).
- **uniformity:** for every $x \in \mathbb{F}_{2^k}$, the function $\boldsymbol{x} \mapsto f_i(\boldsymbol{x})$ is balanced[6].

The tricky point in (first-order) TI implementation is to define a scheme with s_{in} as small as possible and $s_{\text{out}} = s_{\text{in}}$ (which will ensure that the number of shares will not increase). The definition can be straightforwardly extended to higher orders [BGN+14].

Threshold Implementation for Field Multiplications. It can be proved that the minimum s_{in} to define a TI implementation of a multiplication is 3 (see *e.g.* [NRS11]). To process ab from $\boldsymbol{a} = (a_1, a_2, a_3)$ and $\boldsymbol{b} = (b_1, b_2, b_3)$, let us try to define a TI implementation with $s_{\text{out}} = s_{\text{in}}$. For such a purpose, we start by processing the following matrix where $c_{i,j} = a_i b_j$:

$$\begin{pmatrix} c_{1,1} & c_{1,2} & c_{1,3} \\ c_{2,1} & c_{2,2} & c_{2,3} \\ c_{3,1} & c_{3,2} & c_{3,3} \end{pmatrix}$$

[5] In static CMOS, a spurious transition of nodes in a combinational circuit within one clock cycle, resulting from different arrival times of the input signals.

[6] We recall that a function is balanced if every element of $\mathbb{F}_{2^m}$ has the same number of pre-images by f.

$$\left(\begin{array}{ccc} c_{1,1} & c_{1,2} & c_{1,3} \\ c_{2,1} & c_{2,2} & c_{2,3} \\ c_{3,1} & c_{3,2} & c_{3,3} \end{array}\right)$$

It may be checked that the three following functions satisfy the correctness and non-completeness properties:

$$f_1(\boldsymbol{a}, \boldsymbol{b}) = c_{1,1} + c_{1,2} + c_{2,1} \ ,$$

$$f_2(\boldsymbol{a}, \boldsymbol{b}) = c_{1,3} + c_{3,1} \ ,$$

$$f_3(\boldsymbol{a}, \boldsymbol{b}) = c_{2,2} + c_{2,3} + c_{3,2} + c_{3,3} \ .$$

Unfortunately, they do not satisfy the uniformity property. In order to satisfy the latter property and to have $s_{\text{out}} = s_{\text{in}}$, we increase the number of input shares to $s_{\text{in}} = 4$. We depict the solution hereafter:

$$\left(\begin{array}{cc|cc} c_{1,1} & c_{1,2} & c_{1,3} & c_{1,4} \\ c_{2,1} & c_{2,2} & c_{2,3} & c_{2,4} \\ \hline c_{3,1} & c_{3,2} & c_{3,3} & c_{3,4} \\ c_{4,1} & c_{4,2} & c_{4,3} & c_{4,4} \end{array}\right)$$

From the matrix above, we apply the same previous construction and we additionally add the "uniformizing" term in red:

$$\begin{array}{l} f_1(\boldsymbol{a}, \boldsymbol{b}) = (a_1 + a_2) \cdot (b_1 + b_2) + a_3 \\ f_2(\boldsymbol{a}, \boldsymbol{b}) = (a_1 + a_2) \cdot (b_3 + b_4) + a_3 \\ f_3(\boldsymbol{a}, \boldsymbol{b}) = (a_3 + a_4) \cdot (b_1 + b_2) + a_1 \\ f_4(\boldsymbol{a}, \boldsymbol{b}) = (a_3 + a_4) \cdot (b_3 + b_4) + a_1 \end{array} \ .$$

The construction gives an implementation of the multiplication secure against first-order side channel attacks, in the presence of glitches. Other constructions exist, offering different trade-offs for s_{in}, s_{out} and security (see *e.g.* [NRS11], [BGN+14] or [RBN+15]).

Factorize Internal Multiplications. It may be checked that the processing of the multiplication term in $f_1(\boldsymbol{a}, \boldsymbol{b})$ may be re-used if the processing between two new data a' and b' must be done while their respective sharings $\boldsymbol{a'}$ and $\boldsymbol{b'}$ have the shares a_1 and a_2 (resp. b_1 and b_2) in common with $\boldsymbol{a}$ and $\boldsymbol{b}$ respectively. To ensure that some sharings have the first $\lfloor n/2 \rfloor$ shares in common, the algorithm CommonShares (Alg. 4) can be applied. We show hereafter that it can be implemented to be secure against $\lfloor n/2 \rfloor$ side-channel attacks in presence of glitches.

TI for CommonShares. The new sharing $(a'_1, \cdots, a'_n)$ (resp. $(b'_1, \cdots, b'_n)$) of a (resp. b) produced by Alg. 4 from the old sharing $(a_1, \cdots, a_n)$ (resp. $(b_1, \cdots, b_n)$) is defined such that:

– for every $i < \lfloor n/2 \rfloor$, a'_i and b'_i equal a same fresh random value r_i, namely:

$$a'_i = b'_i = f_i(\emptyset) \doteq r_i \ ,$$

– for every $i \geqslant \lfloor n/2 \rfloor$:

$$a'_i = f_i(a_{i-\lfloor n/2 \rfloor}, a_i) \doteq a_{i-\lfloor n/2 \rfloor} + r_i + a_i \ ,$$
$$b'_i = f_i(b_{i-\lfloor n/2 \rfloor}, b_i) \ .$$

It may be checked that the family $(f_i)_{i \in [1..n]}$ is a threshold implementation of Alg. 4. Moreover, since each f_i manipulates at most 2 shares of a or b, Lemma 4 implies that this implementation stays secure against $\lfloor n/2 \rfloor$ side channel attacks in presence of glitches.