An extended abstract of this paper appears in the Proceedings of the 35th Annual International Conference on the Theory and Applications of Cryptographic Techniques (EUROCRYPT 2016), Part ?, ?? and ?? (Eds.), volume ???? of Lecture Notes in Computer Science, pages ???–???, Springer, May 2016. This is the full version.

Randomness Complexity of Private Circuits for Multiplication

Sonia Belaïd[1,2] Fabrice Benhamouda[1] Alain Passelègue[1] Emmanuel Prouff[3,4]
Adrian Thillard[1,3] Damien Vergnaud[1]

[1] ENS, CNRS, INRIA, and PSL, Paris, France
{sonia.belaid, fabrice.benhamouda, alain.passelegue, adrian.thillard, damien.vergnaud}@ens.fr
[2] Thales Communications & Security, France
[3] ANSSI, Paris, France
[4] UPMC, POLSYS, LIP6, Paris, France
emmanuel.prouff@ssi.gouv.fr

Abstract

Many cryptographic algorithms are vulnerable to side channel analysis and several leakage models have been introduced to better understand these flaws. In 2003, Ishai, Sahai and Wagner introduced the d-probing security model, in which an attacker can observe at most d intermediate values during a processing. They also proposed an algorithm that securely performs the multiplication of 2 bits in this model, using only $d(d+1)/2$ random bits to protect the computation. We study the randomness complexity of multiplication algorithms secure in the d-probing model. We propose several contributions: we provide new theoretical characterizations and constructions, new practical constructions and a new efficient algorithmic tool to analyze the security of such schemes.

We start with a theoretical treatment of the subject: we propose an algebraic model for multiplication algorithms and exhibit an algebraic characterization of the security in the d-probing model. Using this characterization, we prove a linear (in d) lower bound and a quasi-linear (non-constructive) upper bound for this randomness cost. Then, we construct a new generic algorithm to perform secure multiplication in the d-probing model that only uses $d + d^2/4$ random bits.

From a practical point of view, we consider the important cases $d \leq 4$ that are actually used in current real-life implementations and we build algorithms with a randomness complexity matching our theoretical lower bound for these small-order cases. Finally, still using our algebraic characterization, we provide a new dedicated verification tool, based on information set decoding, which aims at finding attacks on algorithms for fixed order d at a very low computational cost.

Keywords. Side-Channel Analysis, Probing Model, Randomness Complexity, Constructions, Lower bounds, Probabilistic Method, Information Set Decoding, Algorithmic Tool.

Contents

1 Introduction

Most commonly used cryptographic algorithms are now considered secure against classical black-box attacks, when the adversary has only knowledge of their inputs or outputs. Today, it is however well known that their implementations are vulnerable to side-channel attacks, as revealed in the academic community by Kocher in 1996 [Koc96]. These attacks exploit the physical emanations of the underlying device such as the execution time, the device temperature, or the power consumption during the algorithm execution.

To thwart side-channel attacks, many countermeasures have been proposed by the community. Among them, the most widely deployed one is probably *masking* (a.k.a. secret/processing sharing) [GP99; CJRR99], which has strong links with techniques usually applied in secure multi-party computation (see e.g., [Yao82; BOGKW88]) or private circuits theory [ISW03]. For many kinds of real-life implementations, this countermeasure indeed demonstrated its effectiveness when combined with noise and processing jittering. The idea of the masking approach is to split every single *sensitive* variable/processing, which depends on the secret and on known variables, into several shares. Each share is generated uniformly at random except the last one which ensures that the combination of all the shares is equal to the initial sensitive value. This technique aims at making the physical leakage of one variable independent of the secret and thus useless for the attacker. The tuple of shares still brings information about the shared data but, in practice, the leakages are noisy and the complexity of extracting useful information increases exponentially with the number of shares, the basis of the exponent being related to the amount of noise [CJRR99].

In order to formally prove the security of masking schemes, the community has made important efforts to define leakage models that accurately capture the leakage complexity and simultaneously enable to build security arguments. In 2003, Ishai, Sahai, and Wagner introduced the *d-probing model* in which the attacker can observe at most d exact intermediate values [ISW03]. This model is very convenient to make security proofs but does not fit the reality of embedded devices which leak noisy functions of all their intermediate variables. In 2013, Prouff and Rivain extended the noisy leakage model [PR13], initially introduced by Chari et al. [CJRR99], to propose a new one more accurate than [ISW03] but not very convenient for security proofs. The two models [ISW03] and [PR13] were later unified by Duc, Dziembowski, and Faust [DDF14] and Duc, Faust, and Standaert [DFS15a] who showed that a security proof in the noisy leakage model can be deduced from security proofs in the d-probing model. This sequence of works shows that proving the security of implementations in the d-probing model makes sense both from a theoretical and practical point of view. An implementation secure in the d-probing model is said to satisfy the *d-privacy property* or equivalently to be *d-private* [ISW03] (or secure at order d).

It is worth noting that there is a tight link between sharing techniques, *Multi Party Computation* (MPC) and also *threshold implementations* [BGN+14a; BGN+14b; NRS11]. In particular, the study in the classical d-probing security model can be seen as a particular case of MPC with honest players. Furthermore, the threshold implementations manipulate sharing techniques with additional restrictions to thwart further hardware attacks resulting from the leakage of electronic glitches. This problem can itself be similarly seen as a particular case of MPC, with Byzantine players [LSP82].

1.1 Our Problem

Since most symmetric cryptographic algorithms manipulate Boolean values, the most practical way to protect them is generally to implement *Boolean sharing* (a.k.a. *high-order masking*): namely, each sensitive intermediate result x is shared into several pieces, say $d+1$, which are manipulated by the algorithm and whose parity is equal to x. To secure the processing of a function f on a shared data, one must design a so-called *masking scheme* (or formally a *private circuit*) that describes how to build a sharing of $f(x)$ from that of x while maintaining the d-probing security.

In the context of Boolean sharing, we usually separate the protection of linear functions from that of non-linear ones. In particular, at the hardware level, any circuit can be implemented using only two gates: the linear `XOR` gate and the non-linear `AND` gate. While the protection of linear operations (e.g., `XOR`) is straightforward since the initial function f can be applied to each share separately, it becomes more difficult for non-linear operations (e.g., `AND`). In these cases, the shares cannot be manipulated separately and must generally be processed all together to compute the correct result. These values must then be further protected using additional random bits which results in an important timing overhead.

State-of-the-art solutions to implement Boolean sharing on non-linear functions [RP10; CPRR14] have focused on optimizing the computation complexity. Surprisingly, the amount of necessary random bits has only been in the scope of the seminal paper of Ishai, Sahai and Wagner[ISW03]. In this work, the authors proposed and proved a clever construction (further referred to as ISW multiplication) allowing to compute the multiplication of two shared bits by using $d(d+1)/2$ random bits, that is, half as many random bits as the straightforward solution uses. Their construction has since become a cornerstone of secure implementations [RP10; DDF14; DFS15b; RBN+15]. Even if this result is very important, the quantity of randomness remains very expensive to generate in embedded cryptographic implementations. Indeed, such a generation is usually performed using a physical generator followed by a deterministic random bit generator (DRBG). In addition of being a theoretical "chicken-and-egg" problem for this DRBG protection, in practice the physical generator has often a low throughput and the DRBG is also time-consuming. In general, for a DRBG based on a 128-bit block cipher, one call to this block cipher enables to generate 128 pseudorandom bits[1] (see [BK12]). However, one invocation of the standard AES-128 block cipher with the ISW multiplication requires as much as 30,720 random bits (6 random bytes per multiplication, 4 multiplications per S-box [RP10]) to protect the multiplications when masked at the low order $d = 3$, which corresponds to 240 preliminary calls to the DRBG.

1.2 Our Contributions

We analyze the quantity of randomness required to define a d-private multiplication algorithm at any order d. Given the sharings $\vec{a} = (a_i)_{0 \le i \le d}$, $\vec{b} = (b_i)_{0 \le i \le d}$ of two bits a and b, the problem we tackle out is to find the minimal number of random bits necessary to securely compute a sharing $(c_i)_{0 \le i \le d}$ of the bit $c = ab$ with a d-private algorithm. We limit our scope to the construction of a multiplication based on the sum of shares' products. That is, as in [ISW03], we start with the pairwise products of a's and b's shares and we work on optimizing their sum into $d+1$ shares with as few random bits as possible. We show that this reduces to studying the randomness complexity of some particular d-private compression algorithm that securely transforms the $(d+1)^2$ shares' products into $d+1$ shares of c. In our study we make extensive use of the following theorem that gives an alternative characterization of the d-privacy:

Theorem 3.1 ***(informal).*** A compression algorithm is d-private if and only if there does not exist a set of ℓ intermediate results $\{p_1, \dots, p_\ell\}$ such that $\ell \le d$ and $\sum_{i=1}^{\ell} p_i$ can be written as $\vec{a}^\intercal \cdot \boldsymbol{M} \cdot \vec{b}$ with $\boldsymbol{M}$ being some matrix such that the all-ones vector is in the row space or in the column space of $\boldsymbol{M}$.

From this theorem, we deduce the following lower bound on the randomness complexity:

Theorems 4.3–4.4 ***(informal).*** If $d \ge 3$ (resp. $d = 2$), then a d-private compression algorithm for multiplication must involve at least $d+1$ random bits (resp. 2).

This theorem shows that the randomness complexity is in $\Omega(d)$. Following the probabilistic method, we additionally prove the following theorem which claims that there exists a d-private multiplication algorithm with randomness complexity $O(d \cdot \log d)$. This provides a quasi-linear upper bound $O(d \cdot \log d)$ for the randomness complexity, when $d \to \infty$.

Theorem 4.6 ***(informal).*** There exists a d-private multiplication algorithm with randomness complexity $O(d \cdot \log d)$, when $d \to \infty$.

This upper bound is non-constructive: we show that a randomly chosen multiplication algorithm (in some carefully designed family of multiplication algorithms using $O(d \cdot \log d)$ random bits) is d-private with non-zero probability. This means that there exists one algorithm in this family which is d-private.

In order to explicitly construct private algorithms with low randomness, we analyze the ISW multiplication to bring out necessary and sufficient conditions on the use of the random bits. In particular, we identify necessary chainings and we notice that some random bits may be used several times at several locations to protect more shares' products, while in the ISW multiplication, each random bit is only used twice. From this analysis, we deduce a new d-private multiplication algorithm requiring $\lfloor d^2/4 \rfloor + d$ random bits instead of $d(d+1)/2$. As a positive side-effect, our new construction also reduces the algorithmic complexity of ISW multiplication (i.e., its number of operations).

Based on this generic construction, we then try to optimize some widely used small order instances. In particular, we bring out new multiplication algorithms, for the orders $d = 2$, 3 and 4, which exactly achieve our proven linear lower bound while maintaining the d-privacy. Namely, we present the optimal multiplication algorithms for orders 2, 3 and 4 when summing the shares' products into $d+1$ shares. We

[1]Actually, the generation of pseudorandom bits roughly corresponds to the execution of a block cipher but we should also consider the regular internal state update.

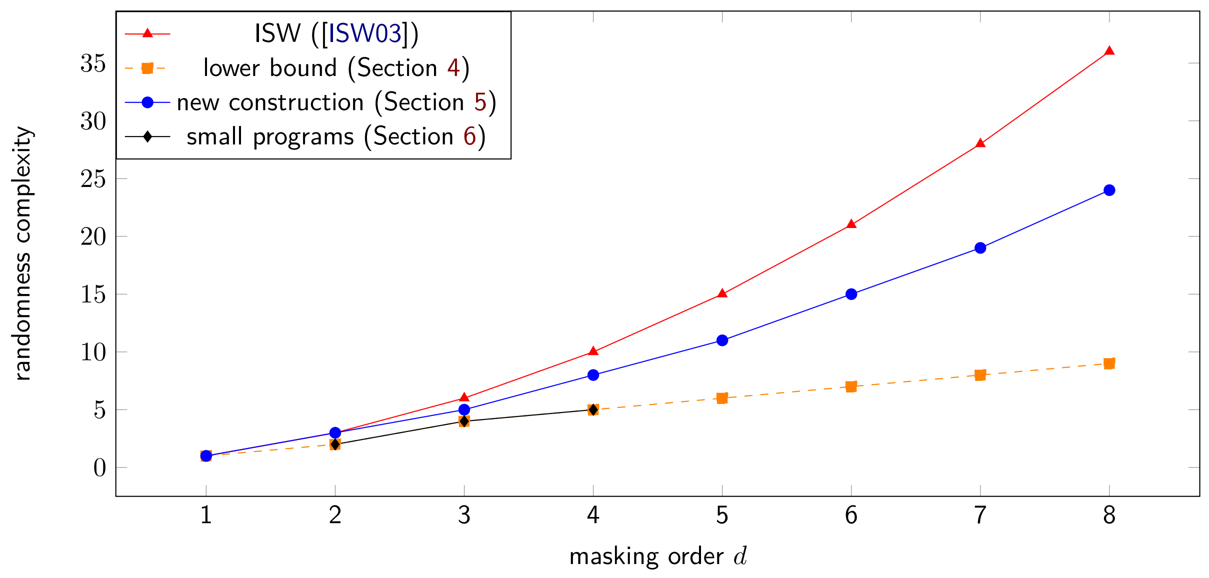

Figure 1: Randomness complexity of d-private multiplication algorithms

formally verify their security using the tool provided in [BBD+15b]. Figure 1 illustrates the randomness complexity of our constructions (for general orders d and small orders) and our lower bound. Note that while the ISW algorithm was initially given for multiplications of bits, it was later extended by Rivain and Prouff in [RP10] for any multiplication in $\mathbb{F}_{2^n}$. In the following, for the sake of simplicity, we refer to binary multiplications ($n = 1$) for our constructions, but note that all of them can also be adapted to multiplication in $\mathbb{F}_{2^n}$.

Contrary to the ISW algorithm, our new constructions are not directly composable — in the sense of Strong Non-Interferent (SNI) in [BBD+15a] — at any order. Fortunately, they can still be used in compositions instead of the ISW algorithms at carefully chosen locations. In this paper, we thus recall the different security properties related to compositions and we show that in the AES example, our new constructions can replace half the ISW ones while preserving the d-privacy of the whole algorithm.

Finally, while the tool provided in [BBD+15b] — which is based on Easycrypt — is able to reveal potential attack paths and formally prove security in the d-probing model with full confidence, it is limited to the verification of small orders ($d = 6$ in our case). Therefore, we propose a new dedicated probabilistic verification tool, which aims at finding attacks in fixed order private circuits (or equivalently masking schemes) at a very low cost. The tool [Tool] is developed in Sage (Python) [Sage] and though less generic than [BBD+15b] it is order of magnitudes faster. It relies on some heuristic assumption (i.e. it cannot be used to actually prove the security) but it usually finds attacks very swiftly for any practical order d. It makes use of information set decoding (a technique from coding theory introduced to the cryptographic community for the security analysis of the McEliece cryptosystem in [Pra62; McE78]).

2 Preliminaries

This section defines the notations and basic notions that we use in this paper, but also some elementary constructions we refer to. In particular, we introduce the notion of d-private compression algorithm for multiplication and we present its only concrete instance which was proposed by Ishai, Sahai, and Wagner [ISW03].

2.1 Notation

For a set S, we denote by $|S|$ its cardinality, and by $s \overset{\$}{\leftarrow} S$ the operation of picking up an element s of S uniformly at random. We denote by $\mathbb{F}_q$ the finite field with q elements. Vectors are denoted by lower case bold font letters, and matrices are denoted by upper case bold font letters. All vectors are column vectors unless otherwise specified. The *kernel* (resp. the *image*) of the linear map associated to a matrix $\boldsymbol{M}$ is denoted by $\ker(\boldsymbol{M})$ (resp. $\operatorname{im}(\boldsymbol{M})$). For a vector $\vec{x}$, we denote by x_i its i-th coordinate and by $\mathsf{hw}(\vec{x})$ its Hamming weight (i.e., the number of its coordinates that are different from 0).

For any fixed $n \geq 1$, let $\boldsymbol{U}_n \in \mathbb{F}_2^{n\times n}$ denote the matrix whose coefficients $u_{i,j}$ equal 1 for all $1 \leq i,j \leq n$. Let $\mathbf{0}_{n,\ell} \in \mathbb{F}_2^{n\times \ell}$ denote the matrix whose coefficients are all 0. Let $\vec{u}_n \in \mathbb{F}_2^n$ denote the vector $(1,\dots,1)^\intercal$ and $\vec{0}_n \in \mathbb{F}_2^n$ denote the vector $(0,\dots,0)^\intercal$. For vectors $\vec{x}_1,\dots,\vec{x}_t$ in $\mathbb{F}_2^n$ we denote $\langle \vec{x}_1,\dots,\vec{x}_t\rangle$ the vector space generated by the set $\{\vec{x}_1,\dots,\vec{x}_t\}$.

We say that an expression $f(x_1,\dots,x_n,r)$ functionally depends on the variable r if there exists $a_1,\dots,a_n$ such that the function $r \mapsto f(a_1,\dots,a_n,r)$ is not constant.

For an algorithm $\mathcal{A}$, we denote by $y \leftarrow \mathcal{A}(x_1,x_2,\dots)$ the operation of running $\mathcal{A}$ on inputs $(x_1,x_2,\dots)$ and letting y denote the output. Moreover, if $\mathcal{A}$ is randomized, we denote by $y \overset{\$}{\leftarrow} \mathcal{A}(x_1,x_2,\dots;r)$ the operation of running $\mathcal{A}$ on inputs $(x_1,x_2,\dots)$ and with uniform randomness r (or with fresh randomness if r is not specified) and letting y denote the output. The *probability density function* associated to a discrete random variable X defined over S (e.g., $\mathbb{F}_2$) is the function which maps $x \in S$ to $\Pr[\, X = x\,]$. It is denoted by $\{X\}$ or by $\{X\}_r$ if there is a need to precise the randomness source r over which the *distribution* is considered.

2.2 Private Circuits

We examine the privacy property in the setting of Boolean circuits and start with the definition of *circuit* and *randomized circuit* given in [ISW03]. A deterministic circuit C is a directed acyclic graph whose vertices are Boolean gates and whose edges are wires. A *randomized circuit* is a circuit augmented with random-bit gates. A random-bit gate is a gate with fan-in 0 that produces a random bit and sends it along its output wire; the bit is selected uniformly and independently of everything else afresh for each invocation of the circuit. From the two previous notions, we may deduce the following definition of a private circuit inspired from [IKL+13].

Definition 2.1. [IKL+13] A *private circuit* for $f\colon\ \mathbb{F}_2^n \to \mathbb{F}_2^m$ is defined by a triple (I,C,O), where

- $I\colon\ \mathbb{F}_2^n \to \mathbb{F}_2^{n'}$ is a randomized circuit with uniform randomness ρ and called input encoder;
- C is a randomized boolean circuit with input in $\mathbb{F}_2^{n'}$, output in $\mathbb{F}_2^{m'}$, and uniform randomness $r \in \mathbb{F}_2^t$;
- $O\colon\ \mathbb{F}_2^{m'} \to \mathbb{F}_2^m$ is a circuit, called output decoder.

We say that C is a *d-private* implementation of f with encoder I and decoder O if the following requirements hold:

- *Correctness*: for any input $w \in \mathbb{F}_2^n$, $\Pr[\, O(C(I(w;\rho);r)) = f(w)\,] = 1$, where the probability is over the randomness ρ and r;
- *Privacy*: for any $w,w' \in \mathbb{F}_2^n$ and any set P of d wires in C, the distributions $\{C_P(I(w;\rho);r)\}_{\rho,r}$ and $\{C_P(I(w';\rho);r)\}_{\rho,r}$ are identical, where $C_P(I(w;\rho);r)$ denotes the list of the d values on the wires from P.

Remark 2.2. It may be noticed that the notions of d-privacy and of security in the d-probing model used, e.g., in [BBD+15b] are perfectly equivalent.

Unless noted otherwise, we assume I and O to be the following *canonical* encoder and decoder: I encodes each bit-coordinate b of its input w by a block $(b_j)_{0\leq j\leq d}$ of $d+1$ random bits with parity b, and O takes the parity of each block of $d+1$ bits. Each block $(b_j)_{0\leq j\leq d}$ is called a *sharing* of b and each b_j is called a *share* of b.

From now on, the wires in a set P used to attack an implementation are referred as the *probes* and the corresponding values in $C_P(I(w;\rho);r)$ as the *intermediate results*. To simplify the descriptions, a probe p is sometimes used to directly denote the corresponding result. A set of probes P such that the distributions $\{C_P(I(w;\rho);r)\}_{\rho,r}$ and $\{C_P(I(w';\rho);r)\}_{\rho,r}$ are *not* identical for some inputs $w,w' \in \mathbb{F}_2^n$ shall be called an *attack*. When the inputs w are clear from the context, the distribution $\{C_P(I(w;\rho);r)\}_{\rho,r}$ is simplified to $\{(p)_{p\in P}\}$.

We now introduce the notions of multiplication algorithm and of d-compression algorithm for multiplication. In this paper, we deeply study d-private multiplication algorithms and d-private compression algorithms for multiplication.

Definition 2.3. A *multiplication algorithm* is a circuit for the multiplication of 2 bits (i.e., with f being the function $f\colon\ (a,b) \in \mathbb{F}_2^2 \mapsto a\cdot b \in \mathbb{F}_2$), using the canonical encoder and decoder.

Algorithm 1 ISW algorithm

```
Require: sharing $(\alpha_{i,j})_{0 \le i,j \le d}$
Ensure: sharing $(c_i)_{0 \le i \le d}$
  for $i = 0$ to $d$ do
    for $j = i + 1$ to $d$ do
      $r_{i,j} \xleftarrow{\$} \mathbb{F}_2$;   $t_{i,j} \leftarrow r_{i,j}$;   $t_{j,i} \leftarrow r_{i,j} + \alpha_{i,j} + \alpha_{j,i}$
  $c_i \leftarrow \alpha_{i,i}$
  for $i = 0$ to $d$ do
    for $j = 0$ to $d$ do
      if $i \neq j$ then
        $c_i \leftarrow c_i + t_{i,j}$
```

Before moving on to the next notion, let us first introduce a new particular encoder, called *multiplicative*, which has been used in all the previous attempts to build a d-private multiplication algorithm. This encoder takes as input two bits $(a, b) \in \mathbb{F}_2^2$, runs the canonical encoder on these two bits to get $d+1$ random bits $(a_0, \dots, a_d)$ and $(b_0, \dots, b_d)$ with parity a and b respectively, and outputs the $(d+1)^2$ bits $(\alpha_{i,j})_{0 \le i,j \le d}$ with $\alpha_{i,j} = a_i \cdot b_j$. Please note that, in particular, we have $a \cdot b = (\sum_{i=0}^{d} a_i) \cdot (\sum_{i=0}^{d} b_i) = \sum_{0 \le i,j \le d} \alpha_{i,j}$.

Definition 2.4. A *d-compression algorithm for multiplication* is a circuit for the multiplication of 2 bits (i.e., with f being the function $f\colon (a, b) \in \mathbb{F}_2^2 \mapsto a \cdot b \in \mathbb{F}_2$), using the canonical decoder and the multiplicative encoder. Moreover, we restrict the circuit C to only perform additions in $\mathbb{F}_2$.

When clear from the context, we often omit the parameter d and simply say "a compression algorithm for multiplication".

Remark 2.5. Any d-compression algorithm for multiplication yields a multiplication algorithm, as the algorithm can start by computing $\alpha_{i,j}$ given its inputs $(a_0, \dots, a_d, b_0, \dots, b_d)$.

Proposition 2.6. *A multiplication algorithm $\mathcal{B}$ constructed from a d-compression algorithm for multiplication $\mathcal{A}$ (as in Remark 2.5) is d-private if and only if the compression algorithm $\mathcal{A}$ is d-private.*

Clearly if $\mathcal{B}$ is d-private, so is $\mathcal{A}$. However, the converse is not straightforward, as an adversary can also probe the input shares a_i and b_i in $\mathcal{B}$, while it cannot in $\mathcal{A}$. The full proof is given in Appendix A and is surprisingly hard: we actually use a stronger version of our algebraic characterization (Theorem 3.1). In the remaining of the paper, we focus on compression algorithms and we do not need to consider probes of the input shares a_i and b_i, which makes notation much simpler.

In the sequel, a d-compression algorithm for multiplication is denoted by $\mathcal{A}(\vec{a}, \vec{b}; \vec{r})$ with $\vec{r}$ denoting the tuple of uniform random bits used by the algorithm and with $\vec{a}$ (resp. $\vec{b}$) denoting the vector of $d+1$ shares of the multiplication operand a (resp. b).

The purpose of the rest of this paper is to investigate how much randomness is needed for such an algorithm to satisfy the d-privacy and to propose efficient or optimal constructions with respect to the consumption of this resource. The number of bits involved in an algorithm $\mathcal{A}(\vec{a}, \vec{b}; \vec{r})$ (i.e., the size of $\vec{r}$) is called its *randomness complexity* or *randomness cost*.

2.3 ISW Algorithm

The first occurrence of a d-private compression circuit for multiplication in the literature is the ISW algorithm, introduced by Ishai, Sahai, and Wagner in [ISW03]. It is described in Algorithm 1. Its randomness cost is $d(d+1)/2$.

To better understand this algorithm, let us first write it explicitly for $d = 3$:

$$\begin{aligned}
c_0 &\leftarrow \alpha_{0,0} + r_{0,1} + r_{0,2} + r_{0,3} \\
c_1 &\leftarrow \alpha_{1,1} + (r_{0,1} + \alpha_{0,1} + \alpha_{1,0}) + r_{1,2} + r_{1,3} \\
c_2 &\leftarrow \alpha_{2,2} + (r_{0,2} + \alpha_{0,2} + \alpha_{2,0}) + (r_{1,2} + \alpha_{1,2} + \alpha_{2,1}) + r_{2,3} \\
c_3 &\leftarrow \alpha_{3,3} + (r_{0,3} + \alpha_{0,3} + \alpha_{3,0}) + (r_{1,3} + \alpha_{1,3} + \alpha_{3,1}) + (r_{2,3} + \alpha_{2,3} + \alpha_{3,2})
\end{aligned}$$

where, for the security to hold, the terms are added from left to right and where the brackets indicate the order in which the operations must be performed (from d-privacy point of view, the addition is not commutative). In particular, when the brackets gather three terms (e.g., $(r_{0,1}+\alpha_{0,1}+\alpha_{1,0})$), the attacker is allowed to probe two values from left to right (e.g., $r_{0,1}+\alpha_{0,1}$ and $(r_{0,1}+\alpha_{0,1}+\alpha_{1,0})$).

Let us now simplify the description by removing all the $+$ symbols, the assignments $c_i \leftarrow$, and defining $\hat{\alpha}_{i,j}$ as $\alpha_{i,j}+\alpha_{j,i}$ if $i \neq j$ and $\alpha_{i,i}$ if $i=j$. The ISW algorithm for $d=3$ can then be rewritten as:

$$\begin{array}{llllll}
\hat{\alpha}_{0,0} & r_{0,1} & & r_{0,2} & & r_{0,3} \\
\hat{\alpha}_{1,1} & (r_{0,1} & \hat{\alpha}_{0,1}) & r_{1,2} & & r_{1,3} \\
\hat{\alpha}_{2,2} & (r_{0,2} & \hat{\alpha}_{0,2}) & (r_{1,2} & \hat{\alpha}_{1,2}) & r_{2,3} \\
\hat{\alpha}_{3,3} & (r_{0,3} & \hat{\alpha}_{0,3}) & (r_{1,3} & \hat{\alpha}_{1,3}) & (r_{2,3} \quad \hat{\alpha}_{2,3}).
\end{array}$$

Please note that the expression of $\hat{\alpha}_{i,j}$ with $i \neq j$ (*i.e.* $\alpha_{i,j}+\alpha_{j,i}$) is expanded before the actual evaluation, i.e., as in the previous representation, the sum $\alpha_{i,j}+\alpha_{j,i}$ is not evaluated beforehand but evaluated during the processing of $r_{i,j}+\hat{\alpha}_{i,j}=r_{i,j}+\alpha_{i,j}+\alpha_{j,i}$.

3 Algebraic Characterization

In order to reason about the required quantity of randomness in d-private compression algorithms for multiplication, we define an algebraic condition on the security and we prove that an algorithm is d-private if and only if there is no set of probes which satisfies it.

3.1 Matrix Notation

As our condition is algebraic, it is practical to introduce some matrix notation for our probes. We write $\vec{a}=(a_0,\dots,a_d)^\intercal$ and $\vec{b}=(b_0,\dots,b_d)^\intercal$ the vectors corresponding to the shares of the inputs a and b respectively. We also denote by $\vec{r}=(r_1,\dots,r_R)^\intercal$ the vector of the random bits.

We remark that, for any probe p on a compression algorithm for multiplication, p is always an expression that can be written as a sum of $\alpha_{i,j}$'s (with $\alpha_{i,j}=a_i \cdot b_j$) and r_k's, and possibly a constant $c_p \in \mathbb{F}_2$. In other word, we can write p as

$$p=\vec{a}^\intercal \cdot \boldsymbol{M_p} \cdot \vec{b}+\vec{s_p}^{\,t} \cdot \vec{r}+c_p,$$

with $\boldsymbol{M_p}$ being a matrix in $\mathbb{F}_2^{(d+1)\times(d+1)}$ and $\vec{s_p}$ being a vector in $\mathbb{F}_2^R$. This matrix $\boldsymbol{M_p}$ and this vector $\vec{s_p}$ are uniquely defined. In addition, any sum of probes can also be written that way.

Furthermore, if $c_p=1$, we can always sum the probe with 1 and consider $p+1$ instead of p. This does not change anything on the probability distribution we consider. Therefore, for the sake of simplicity, we always assume $c_p=0$ in all the paper.

3.2 Algebraic Condition

We now introduce our algebraic condition:

Condition 1. *A set of probes $P=\{p_1,\dots,p_\ell\}$ on a d-compression algorithm for multiplication satisfies Condition 1 if and only if the expression $f=\sum_{i=1}^{\ell} p_i$ can be written as $f=\vec{a}^\intercal \cdot \boldsymbol{M} \cdot \vec{b}$ with $\boldsymbol{M}$ being some matrix such that $\vec{u}_{d+1}$ is in the row space or the column space of $\boldsymbol{M}$.*

As seen previously, the expression f can always be written as

$$f=\vec{a}^\intercal \cdot \boldsymbol{M} \cdot \vec{b}+\vec{s}^\intercal \cdot \vec{r},$$

for some matrix $\boldsymbol{M}$ and some vector $\vec{s}$. Therefore, what the condition enforces is that $\vec{s}=\vec{0}_R$ (or in other words, f does not functionally depend on any random bit) and the column space or the row space of $\boldsymbol{M}$ contains the vector $\vec{u}_{d+1}$.

A Weaker Condition. To better understand Condition 1, let us introduce a weaker condition which is often easier to deal with:

Condition 2 (weak condition)**.** *A set of probes $P=\{p_1,\dots,p_\ell\}$ on a d-compression algorithm for multiplication satisfies Condition 2 if and only if the expression $f=\sum_{i=1}^{\ell} p_i$ does not functionally depend on any r_k and there exists a map $\gamma\colon \{0,\dots,d\} \to \{0,\dots,d\}$ such that f does functionally depend on every $(\alpha_{i,\gamma(i)})_{0 \le i \le d}$ or on every $(\alpha_{\gamma(i),i})_{0 \le i \le d}$.*

This condition could be reformulated as $f=\sum_{i=1}^{\ell} p_i$ functionally depends on either all the a_i's or all the b_i's and does not functionally depend on any r_k. It is easy to see that any set P verifying Condition 1 also verifies Condition 2.

3.3 Algebraic Characterization

Theorem 3.1. *Let $\mathcal{A}$ be a d-compression algorithm for multiplication. Then, $\mathcal{A}$ is d-private if and only if there does not exist a set $P = \{p_1, \ldots, p_\ell\}$ of $\ell \le d$ probes that satisfies Condition 1. Furthermore any set $P = \{p_1, \ldots, p_\ell\}$ satisfying Condition 1 is an attack.*

Please note that Theorem 3.1 would not be valid with Condition 2 (instead of Condition 1). A counterexample is given in Appendix B.

Theorem 3.1. **Direction 1: Left to right.** We prove hereafter that if $\mathcal{A}$ is d-private, then there does not exist a set $P = \{p_1, \ldots, p_\ell\}$ of $\ell \le d$ probes that satisfies Condition 1.

By contrapositive, let us assume that there exists a set $P = \{p_1, \ldots, p_\ell\}$ of at most d probes that satisfies Condition 1. Let $\boldsymbol{M}$ be the matrix such that $f = \sum_{i=1}^{\ell} p_i = \vec{a}^\intercal \cdot \boldsymbol{M} \cdot \vec{b}$ and let us assume, without loss of generality, that $\vec{u}_{d+1}$ is in the vector subspace generated by the columns of $\boldsymbol{M}$. We remark that, for any $\vec{v} \in \mathbb{F}_2^{d+1}$:

$$\Pr\left[\,\vec{a}^\intercal \cdot \vec{v} = a\,\right] = \begin{cases} 1 & \text{when } \vec{v} = \vec{u}_{d+1} \\ \frac{1}{2} & \text{when } \vec{v} \neq \vec{u}_{d+1} \end{cases}$$

by definition of the sharing $\vec{a}$ of a (probability is taken over $\vec{a}$). Thus we have, when $a = 0$ (assuming that b is uniformly random)

$$\begin{aligned}
&\Pr\left[\,f = 0 \mid a = 0\,\right] \\
&= \Pr\left[\vec{a}^\intercal \cdot \boldsymbol{M} \cdot \vec{b} = 0 \;\middle|\; \vec{a}^\intercal \cdot \vec{u}_{d+1} = 0\right] \\
&= \Pr\left[\vec{a}^\intercal \cdot \vec{u}_{d+1} = 0 \;\middle|\; a = 0 \text{ and } \boldsymbol{M} \cdot \vec{b} = \vec{u}_{d+1}\right] \cdot \Pr\left[\boldsymbol{M} \cdot \vec{b} = \vec{u}_{d+1}\right] \\
&\quad + \textstyle\sum_{\vec{v} \in \mathbb{F}_2^{d+1} \setminus \{\vec{u}_{d+1}\}} \Pr\left[\vec{a}^\intercal \cdot \vec{v} = 0 \;\middle|\; a = 0 \text{ and } \boldsymbol{M} \cdot \vec{b} = \vec{v}\right] \cdot \Pr\left[\boldsymbol{M} \cdot \vec{b} = \vec{v}\right] \\
&= 1 \cdot \Pr\left[\boldsymbol{M} \cdot \vec{b} = \vec{u}_{d+1}\right] + \textstyle\sum_{\vec{v} \in \mathbb{F}_2^{d+1} \setminus \{\vec{u}_{d+1}\}} \frac{1}{2} \cdot \Pr\left[\boldsymbol{M} \cdot \vec{b} = \vec{v}\right] \\
&= 1 \cdot \Pr\left[\boldsymbol{M} \cdot \vec{b} = \vec{u}_{d+1}\right] + \tfrac{1}{2}\left(1 - \Pr\left[\boldsymbol{M} \cdot \vec{b} = \vec{u}_{d+1}\right]\right) \\
&= \tfrac{1}{2} + \tfrac{1}{2}\Pr\left[\boldsymbol{M} \cdot \vec{b} = \vec{u}_{d+1}\right].
\end{aligned}$$

Similarly, when $a = 1$, we have

$$\Pr\left[\,f = 0 \mid a = 1\,\right] = \tfrac{1}{2} - \tfrac{1}{2}\Pr\left[\boldsymbol{M} \cdot \vec{b} = \vec{u}_{d+1}\right].$$

As $\vec{u}_{d+1}$ is in the column space of $\boldsymbol{M}$, the distribution of $\{f\}$ is not the same when $a = 0$ and when $a = 1$. This implies that the distribution $\{(p_1, \ldots, p_\ell)\}$ is also different when $a = 0$ and $a = 1$. Hence $\mathcal{A}$ is not d-private.

This concludes the proof of the first implication and the fact that any set $P = \{p_1, \ldots, p_\ell\}$ satisfying Condition 1 is an attack.

Direction 2: Right to left. Let us now prove by contradiction that if there does not exist a set $P = \{p_1, \ldots, p_\ell\}$ of $\ell \le d$ probes that satisfies Condition 1, then $\mathcal{A}$ is d-private.

Let us assume that $\mathcal{A}$ is not d-private. Then there exists an attack using a set of probes $P = \{p_1, \ldots, p_\ell\}$ with $\ell \le d$. This is equivalent to say that there exists two inputs $(a^{(0)}, b^{(0)}) \neq (a^{(1)}, b^{(1)})$ such that the distribution $\{(p_1, \ldots, p_\ell)\}$ is not the same whether $(a, b) = (a^{(0)}, b^{(0)})$ or $(a, b) = (a^{(1)}, b^{(1)})$.

We first remark that we can consider $0 = a^{(0)} \neq a^{(1)} = 1$, without loss of generality as the $a^{(i)}$'s and the $b^{(i)}$'s play a symmetric role (and $(a^{(0)}, b^{(0)}) \neq (a^{(1)}, b^{(1)})$). Furthermore, we can always choose $b^{(0)} = b^{(1)}$, as if the distribution $\{(p_1, \ldots, p_\ell)\}$ is not the same whether $(a, b) = (0, b^{(0)})$ or $(a, b) = (1, b^{(1)})$, with $b^{(0)} \neq b^{(1)}$, then:

- it is not the same whether $(a, b) = (0, b^{(0)})$ or $(a, b) = (1, b^{(0)})$ (in which case, we could have taken $b^{(1)} = b^{(0)}$), or
- it is not the same whether $(a, b) = (1, b^{(0)})$ or $(a, b) = (1, b^{(1)})$ (in which case, we can just exchange the a's and the b's roles).

To summarize, there exists $b^{(0)}$ such that the distribution $\{(p_1, \dots, p_\ell)\}$ is not the same whether $(a, b) = (0, b^{(0)})$ or $(a, b) = (1, b^{(0)})$.

In the sequel $b^{(0)}$ is fixed and we call a tuple $(p_1, \dots, p_\ell)$ satisfying the previous property an *attack tuple*.

We now remark that if $\ell = 1$ or if even the distribution $\{(\sum_{i=1}^{\ell} p_i)\}$ is not the same whether $(a, b) = (0, b^{(0)})$ or $(a, b) = (1, b^{(0)})$ (i.e., $(\sum_{i=1}^{\ell} p_i)$ is an attack tuple), then it follows easily from the probability analysis of the previous proof for the other direction of the theorem, that the set P satisfies Condition 1. The main difficulty is that it is not necessarily the case that $\ell = 1$ or $(\sum_{i=1}^{\ell} p_i)$ is an attack tuple. To overcome it, we use linear algebra.

But first, let us introduce some useful notations and lemmas. We write $\vec{p}$ the vector $(p_1, \dots, p_\ell)^\intercal$ and we say that $\vec{p}$ is an *attack vector* if and only if $(p_1, \dots, p_\ell)$ is an attack tuple. Elements of $\vec{p}$ are polynomials in the a_i's, the b_j's and the r_k's.

Lemma 3.2. *If $\vec{p}$ is an attack vector and $\boldsymbol{N}$ is an invertible matrix in $\mathbb{F}_2^{\ell \times \ell}$, then $\boldsymbol{N} \cdot \vec{p}$ is an attack vector.*

Proof. This is immediate from the fact that $\boldsymbol{N}$ is invertible. Indeed, as a matrix over $\mathbb{F}_2$, $\boldsymbol{N}^{-1}$ is also a matrix over $\mathbb{F}_2$. Hence, multiplying the set of probes $\{\boldsymbol{N} \cdot \vec{p}\}$ by $\boldsymbol{N}^{-1}$ (which leads to the first set of probes $\{\vec{p}\}$) can be done by simply computing sums of elements in $\{\boldsymbol{N} \cdot \vec{p}\}$. Hence, as the distribution of $\{\vec{p}\}$ differs when $(a, b) = (0, b^{(0)})$ and $(a, b) = (1, b^{(0)})$, the same is true for the distribution $\{\boldsymbol{N} \cdot \vec{p}\}$. □

We also use the following straightforward lemma.

Lemma 3.3. *If $(p_1, \dots, p_\ell)$ is an attack tuple such that the $\ell - t + 1$ random variables $(p_1, \dots, p_t)$, p_{t+1}, ..., and p_ℓ are mutually independent, and the distributions of $(p_{t+1}, \dots, p_\ell)$ is the same for all the values of the inputs (a, b), then $(p_1, \dots, p_t)$ is an attack tuple.*

Let us consider the matrix $\boldsymbol{S} \in \mathbb{F}_2^{\ell \times R}$ whose coefficients $s_{i,j}$ are defined as $s_{i,j} = 1$ if and only if the expression p_i functionally depends on r_j. In other words, if we write $p_i = \vec{a}^\intercal \cdot \boldsymbol{M_{p_i}} \cdot \vec{b} + \vec{s_{p_i}}^\intercal \cdot \vec{r}$, the i-th row of $\boldsymbol{S}$ is $\vec{s_{p_i}}^\intercal$. We can permute the random bits (i.e., the columns of $\boldsymbol{S}$ and the rows of $\vec{r}$) such that a row reduction on the matrix $\boldsymbol{S}$ yields a matrix of the form:

$$\boldsymbol{S'} = \begin{pmatrix} \boldsymbol{0}_{t,t} & \boldsymbol{0}_{t,\ell-t} \\ \boldsymbol{I}_t & \boldsymbol{S''} \end{pmatrix}.$$

Let $\boldsymbol{N}$ be the invertible matrix in $\mathbb{F}_2^{\ell \times \ell}$ such that $\boldsymbol{N} \cdot \boldsymbol{S} = \boldsymbol{S'}$. And we write $\vec{p'} = (p'_1, \dots, p'_\ell)^\intercal = \boldsymbol{N} \cdot \vec{p}$. Then, $\vec{p'}$ is also an attack vector according to Lemma 3.2. In addition, for $t < i \leq \ell$, p'_i does functionally depend on r_i and no other p'_j does functionally depend on r_j (due to the shape of $\boldsymbol{S'}$). Therefore, according to Lemma 3.3, $(p'_1, \dots, p'_t)$ is an attack tuple.

We remark that $(p'_1, \dots, p'_t)$ does not functionally depend on any random bit, due to the shape of $\boldsymbol{S'}$. Therefore, for each $1 \leq i \leq t$, we can write:

$$p'_i = \vec{a}^\intercal \cdot \boldsymbol{M'_i} \cdot \vec{b},$$

for some matrix $\boldsymbol{M'_i}$.

We now need a final lemma to be able to conclude.

Lemma 3.4. *If $(p'_1, \dots, p'_t)$ is an attack tuple, then there exists a vector $\vec{b^*} \in \mathbb{F}_2^{d+1}$ such that $\vec{u}_{d+1}$ is in the vector space $\langle \boldsymbol{M'_1} \cdot \vec{b^*}, \dots, \boldsymbol{M'_t} \cdot \vec{b^*} \rangle$.*

Proof. This lemma can be seen as a generalization of the probability analysis in the proof of the first direction of the theorem.

We suppose by contradiction that $(p'_1, \dots, p'_t)$ is an attack vector but there does not exist a vector $\vec{b^*} \in \mathbb{F}_2^{d+1}$ such that $\vec{u}_{d+1}$ is in the vector space $\langle \boldsymbol{M'_1} \cdot \vec{b^*}, \dots, \boldsymbol{M'_t} \cdot \vec{b^*} \rangle$. Then, for any value $a^{(0)}$, any vector $\vec{b^{(0)}} \in \mathbb{F}_2^{d+1}$, and any vector $\vec{x} = (x_1, \dots, x_t)^\intercal \in \mathbb{F}_2^t$:

$$\begin{aligned} &\Pr\left[(p'_1, \dots, p'_t) = (x_1, \dots, x_t) \;\middle|\; a = a^{(0)} \text{ and } \vec{b} = \vec{b^{(0)}} \right] \\ &= \Pr\left[(\vec{a}^\intercal \cdot \boldsymbol{M'_1} \cdot \vec{b^{(0)}}, \dots, \vec{a}^\intercal \cdot \boldsymbol{M'_t} \cdot \vec{b^{(0)}}) = (x_1, \dots, x_t) \;\middle|\; \vec{a}^\intercal \cdot \vec{u}_{d+1} = a^{(0)} \right] \\ &= \Pr\left[\vec{a}^\intercal \cdot \boldsymbol{B} = \vec{x}^\intercal \;\middle|\; \vec{a}^\intercal \cdot \vec{u}_{d+1} = a^{(0)} \right], \end{aligned}$$

where $\boldsymbol{B}$ is the matrix whose i-th column is the vector $\boldsymbol{M_i'} \cdot \vec{b^{(0)}}$. To conclude, we just need to remark that

$$\Pr[\vec{a}^\intercal \cdot \boldsymbol{B} = \vec{x}^\intercal \mid \vec{a}^\intercal \cdot \vec{u}_{d+1} = 0] = \Pr[\vec{a}^\intercal \cdot \boldsymbol{B} = \vec{x}^\intercal \mid \vec{a}^\intercal \cdot \vec{u}_{d+1} = 1],$$

which implies that the probability distribution of $(p_1', \ldots, p_t')$ is independent of the value of a, which contradicts the fact the $(p_1', \ldots, p_t')$ is an attack tuple.

To prove the previous equality, we use the fact that $\vec{u}_{d+1}$ is not in the column space of $\boldsymbol{B}$ and therefore the value of $\vec{a}^\intercal \cdot \vec{u}_{d+1}$ is uniform and independent of the value of $\vec{a}^\intercal \cdot \boldsymbol{B}$ (when $\vec{a}$ is a uniform vector in $\mathbb{F}_2^{d+1}$). □

Thanks to Lemma 3.4, there exists a vector $\vec{\sigma} = (\sigma_1, \ldots, \sigma_t)^\intercal \in \mathbb{F}_2^t$ and a vector $\vec{b^*} \in \mathbb{F}_2^{d+1}$ such that

$$\left(\sum_{i=1}^{t} \sigma_i \cdot \boldsymbol{M_i'}\right) \cdot \vec{b^*} = \vec{u}_{d+1} \quad . \tag{1}$$

Let $\vec{\sigma'}$ be the vector in $\mathbb{F}_2^\ell$ defined by $\vec{\sigma'}^\intercal = \begin{pmatrix}\vec{\sigma}^\intercal & \vec{0}_{\ell-t}^\intercal\end{pmatrix} \cdot \boldsymbol{N}$. We have:

$$\vec{\sigma'}^\intercal \cdot \vec{p} = \sum_{i=1}^{t} \sigma_i \cdot p_i' = \sum_{i=1}^{t} \sigma_i \cdot \vec{a}^\intercal \cdot \boldsymbol{M_i'} \cdot \vec{b} = \vec{a}^\intercal \cdot \left(\sum_{i=1}^{t} \sigma_i \cdot \boldsymbol{M_i'}\right) \cdot \vec{b} \quad . \tag{2}$$

Therefore, we can define the set $P' = \{p_i \mid \sigma_i = 1\}$. This set satisfies Condition 1, according to Equations (1) and (2).

This concludes the proof. □

4 Theoretical Lower and Upper Bounds

In this section, we exhibit lower and upper bounds for the randomness complexity of a d-private compression algorithm for multiplication. We first prove an algebraic result and an intermediate lemma that we then use to show that at least $d + 1$ random bits are required to construct a d-private compression algorithm for multiplication, for any $d \geq 3$ (and 2 random bits are required for $d = 2$). Finally, we provide a (non-constructive) proof that for large enough d, there exists a d-private multiplication algorithm with a randomness complexity $O(d \cdot \log d)$.

4.1 A Splitting Lemma

We first prove an algebraic result, stated in the lemma below, that we further use to prove Lemma 4.2. The latter allows us to easily exhibit attacks in order to prove our lower bounds.

Lemma 4.1. *Let $n \geq 1$. Let $\boldsymbol{M}_0, \boldsymbol{M}_1 \in \mathbb{F}_2^{n\times n}$ such that $\boldsymbol{M}_0 + \boldsymbol{M}_1 = \boldsymbol{U}_n$. Then, there exists a vector $\vec{v} \in \mathbb{F}_2^n$ such that:*

$$\boldsymbol{M}_0 \cdot \vec{v} = \vec{u}_n \qquad or \qquad \boldsymbol{M}_1 \cdot \vec{v} = \vec{u}_n \qquad or \qquad \boldsymbol{M}_0^\intercal \cdot \vec{v} = \vec{u}_n \qquad or \qquad \boldsymbol{M}_1^\intercal \cdot \vec{v} = \vec{u}_n \quad .$$

Lemma 4.1. We show the above lemma by induction on n.

Base case: for $n = 1$, $\boldsymbol{M}_0, \boldsymbol{M}_1, \boldsymbol{U} \in \mathbb{F}_2$, so $\boldsymbol{M}_0 + \boldsymbol{M}_1 = 1$, which implies $\boldsymbol{M}_0 = 1$ or $\boldsymbol{M}_1 = 1$ and the claim immediately follows.

Inductive case: let us assume that the claim holds for a fixed $n \geq 1$. Let us consider two matrices $\boldsymbol{M}_0, \boldsymbol{M}_1 \in \mathbb{F}_2^{(n+1)\times(n+1)}$ such that $\boldsymbol{M}_0 + \boldsymbol{M}_1 = \boldsymbol{U}_{n+1}$.

Clearly, if $\boldsymbol{M}_0$ (or $\boldsymbol{M}_1$) is invertible, then the claim is true (as $\vec{u}_{n+1}$ is in its range). Then, let us assume that $\boldsymbol{M}_0$ is not invertible. Then, there exists a non-zero vector $\vec{x} \in \ker(\boldsymbol{M_0})$. Now, as $\operatorname{im}(\boldsymbol{U}_{n+1}) = \{\vec{0}_{n+1}, \vec{u}_{n+1}\}$, if $\boldsymbol{U}_{n+1} \cdot \vec{x} = \vec{u}_{n+1}$, then $\boldsymbol{M}_1 \cdot \vec{x} = \vec{u}_{n+1}$ and the claim is true. Hence, clearly, the claim is true if $\ker(\boldsymbol{M}_0) \neq \ker(\boldsymbol{M}_1)$ (with the symmetric remark). The same remarks hold when considering matrices $\boldsymbol{M}_0^\intercal$ and $\boldsymbol{M}_1^\intercal$.

Hence, the only remaining case to consider is when $\ker(\boldsymbol{M}_0) \neq \{\vec{0}_{n+1}\}$, $\ker(\boldsymbol{M}_0^\intercal) \neq \{\vec{0}_{n+1}\}$ and when $\ker(\boldsymbol{M}_0) = \ker(\boldsymbol{M}_1)$ and $\ker(\boldsymbol{M}_0^\intercal) = \ker(\boldsymbol{M}_1^\intercal)$. In particular, we have $\ker(\boldsymbol{M}_0) \subseteq \ker(\boldsymbol{U}_{n+1})$ and $\ker(\boldsymbol{M}_0^\intercal) \subseteq \ker(\boldsymbol{U}_{n+1})$.

Let $\vec{x} \in \ker(\boldsymbol{M}_0)$ (and then $\vec{x} \in \ker(\boldsymbol{M}_1)$ as well) be a non-zero vector. Up to some rearrangement of the *columns* of $\boldsymbol{M}_0$ and $\boldsymbol{M}_1$ (by permuting some columns), we can assume without loss of generality that $\vec{x} = (1,\dots,1,0,\dots,0)^\intercal$. Let $\boldsymbol{X}$ denote the matrix $(\vec{x}, \vec{e}_2, \dots, \vec{e}_{n+1})$ where $\vec{e}_i = (0,\dots,0,1,0,\dots,0)^\intercal$ is the i-th canonical vector of length $n+1$, so that it has a 1 in the i-th position and 0's everywhere else.

Now, let $\vec{y} \in \ker(\boldsymbol{M}_0^\intercal)$ (and then $\vec{y} \in \ker(\boldsymbol{M}_1^\intercal)$ as well) be a non-zero vector, so $\vec{y}^\intercal \cdot \boldsymbol{M}_0^\intercal = \vec{0}_{n+1}^\intercal$. Moreover, up to some rearrangement of the *rows* of $\boldsymbol{M}_0$ and $\boldsymbol{M}_1$, we can assume that $\vec{y} = (1,\dots,1,0,\dots,0)^\intercal$. Let $\boldsymbol{Y}$ denote the matrix $(\vec{y}, \vec{e}_2, \dots, \vec{e}_{n+1})$.

Please note that rearrangements apply to the columns in the first case and to the rows in the second case, so we can assume without loss of generality that there exists both $\vec{x} \in \ker(\boldsymbol{M}_0)$ and $\vec{y} \in \ker(\boldsymbol{M}_0^\intercal)$ with the above form and matrices $\boldsymbol{X}$ and $\boldsymbol{Y}$ are well defined.

We now define the matrices $\boldsymbol{M}_0' = \boldsymbol{Y}^\intercal \cdot \boldsymbol{M}_0 \cdot \boldsymbol{X}$ and $\boldsymbol{M}_1' = \boldsymbol{Y}^\intercal \cdot \boldsymbol{M}_1 \cdot \boldsymbol{X}$. We have:

$$\boldsymbol{M}_0' = \begin{pmatrix} & \vec{y}^\intercal & \\ \vec{0}_n & & \boldsymbol{I}_n \end{pmatrix} \cdot \boldsymbol{M}_0 \cdot \begin{pmatrix} \vec{x} & \vec{0}_n^\intercal \\ & \boldsymbol{I}_n \end{pmatrix} = \begin{pmatrix} & \vec{y}^\intercal & \\ \vec{0}_n & & \boldsymbol{I}_n \end{pmatrix} \cdot \begin{pmatrix} \vec{0}_{n+1} & \boldsymbol{M}_0^{(1)} \end{pmatrix}$$

where $\boldsymbol{M}_0^{(1)}$ is the matrix extracted from $\boldsymbol{M}_0$ by removing its first column. Hence:

$$\boldsymbol{M}_0' = \begin{pmatrix} 0 & \vec{0}_n^\intercal \\ \vec{0}_n & \boldsymbol{M}_0^{(1,1)} \end{pmatrix}$$

where $\boldsymbol{M}_0^{(1,1)}$ is the matrix extracted from $\boldsymbol{M}_0$ by removing its first column and its first row. Similar equation holds for $\boldsymbol{M}_1'$ as well. Thus, it is clear that:

$$\boldsymbol{M}_0' + \boldsymbol{M}_1' = \begin{pmatrix} 0 & \vec{0}_n^\intercal \\ \vec{0}_n & \boldsymbol{U}_n \end{pmatrix}.$$

Let us consider the matrices $\boldsymbol{M}_0''$ and $\boldsymbol{M}_1''$ in $\mathbb{F}_2^{n\times n}$ that are extracted from matrices $\boldsymbol{M}_0'$ and $\boldsymbol{M}_1'$ by removing their first row and their first column (i.e., $\boldsymbol{M}_i'' = \boldsymbol{M}_i'^{(1,1)}$ with the previous notation). Then, it is clear that $\boldsymbol{M}_0'' + \boldsymbol{M}_1'' = \boldsymbol{U}_n$. As matrices in $\mathbb{F}_2^{n\times n}$, by induction hypothesis, there exists $\vec{v}'' \in \mathbb{F}_2^n$ such that at least one of the 4 propositions from Lemma 4.1 holds. We can assume without loss of generality that $\boldsymbol{M}_0'' \cdot \vec{v}'' = \vec{u}_n$.

Let $\vec{v}' = \begin{pmatrix} 0 \\ \vec{v}'' \end{pmatrix} \in \mathbb{F}_2^{n+1}$. Then, we have:

$$\boldsymbol{M}_0' \cdot \vec{v}' = \begin{pmatrix} 0 & \vec{0}_n^\intercal \\ \vec{0}_n & \boldsymbol{M}_0'' \end{pmatrix} \cdot \begin{pmatrix} 0 \\ \vec{v}'' \end{pmatrix} = \begin{pmatrix} \vec{0}_n \cdot \vec{v}'' \\ \boldsymbol{M}_0'' \cdot \vec{v}'' \end{pmatrix} = \begin{pmatrix} 0 \\ \vec{u}_n \end{pmatrix}.$$

Now, let $\vec{v} = \boldsymbol{X} \cdot \vec{v}'$ and $\vec{w} = \boldsymbol{M}_0 \cdot \vec{w}$, so $\boldsymbol{Y}^\intercal \cdot \vec{w} = \boldsymbol{Y}^\intercal \cdot \boldsymbol{M}_0 \boldsymbol{X} \cdot \vec{v}' = \boldsymbol{M}_0' \cdot \vec{v}' = \begin{pmatrix} 0 \\ \vec{u}_n \end{pmatrix}$. Moreover, as $\boldsymbol{Y}$ is invertible, $\vec{w}$ is the *unique* vector such that $\boldsymbol{Y}^\intercal \cdot \vec{w} = \begin{pmatrix} 0 \\ \vec{u}_n \end{pmatrix}$. Finally, as the vector $\vec{u}_{n+1}$ satisfies $\boldsymbol{Y}^\intercal \cdot \vec{u}_{n+1} = \begin{pmatrix} 0 \\ \vec{u}_n \end{pmatrix}$, then $\vec{w} = \vec{u}_{n+1}$, and the claim follows for $n+1$, since $\vec{v}$ satisfies $\boldsymbol{M}_0 \cdot \vec{v} = \vec{w} = \vec{u}_{n+1}$.

Conclusion: The claim follows for any $n \geq 1$, and so does Lemma 4.1. □

We can now easily prove the following statement that is our main tool for proving our lower bounds, as explained after its proof.

Lemma 4.2. *Let $\mathcal{A}$ be a d-compression algorithm for multiplication. If there exists two sets S_1 and S_2 of at most d probes such that $s_i = \sum_{p \in S_i} p$ does not functionally depend on any of the random bits, for $i \in \{0,1\}$, and such that $s_0 + s_1 = a \cdot b$, then $\mathcal{A}$ is not d-private.*

Lemma 4.2. Let $\mathcal{A}$, S_0, S_1, s_0 and s_1 defined in the above statement. Then, there exists $\boldsymbol{M}_i \in \mathbb{F}_2^{(d+1)\times(d+1)}$ such that $s_i = \vec{a}^\intercal \cdot \boldsymbol{M}_i \cdot \vec{b}$, for $i \in \{0,1\}$. Furthermore, as $s_0 + s_1 = a \cdot b = \vec{a}^\intercal \cdot \boldsymbol{U}_{d+1} \cdot \vec{b}$, we have $\boldsymbol{M}_0 + \boldsymbol{M}_1 = \boldsymbol{U}_{d+1}$. Hence, via Lemma 4.1, there exists $\vec{v} \in \mathbb{F}_2^{d+1}$ and $i \in \{0,1\}$ such that $\boldsymbol{M}_i \cdot \vec{v} = \vec{u}_{d+1}$ or $\boldsymbol{M}_i^\intercal \cdot \vec{v} = \vec{u}_{d+1}$. This means that $\vec{u}_{d+1}$ is in the row subspace or in the column subspace of $\boldsymbol{M}_i$, and therefore, $\boldsymbol{M}_i$ satisfies Condition 1. Therefore, as $|S_i| \leq d$, applying Theorem 3.1, $\mathcal{A}$ is not d-private. Lemma 4.2 follows. □

We use the above lemma to prove our lower bounds as follows: for proving that at least $R(d)$ random bits are required in order to achieve d-privacy for a compression algorithm for multiplication, we prove that any algorithm with a lower randomness complexity is not d-private by exhibiting two sets of probes S_0 and S_1 that satisfy the requirements of Lemma 4.2.

4.2 Simple Linear Lower Bound

As a warm-up, we show that at least d random bits are required, for $d \geq 2$.

Theorem 4.3. *Let $d \geq 2$. Let us consider a d-compression algorithm for multiplication $\mathcal{A}$. If $\mathcal{A}$ uses only $d-1$ random bits, then $\mathcal{A}$ is not d-private.*

Theorem 4.3. Let $r_1, \dots, r_{d-1}$ denote the random bits used by $\mathcal{A}$. Let $c_0, \dots, c_d$ denote the outputs of $\mathcal{A}$. Let us define $\boldsymbol{N} \in \mathbb{F}_2^{(d-1) \times d}$ as the matrix whose coefficients $n_{i,j}$ are equal to 1 if and only if c_j functionally depends on r_i, for $1 \leq i \leq d-1$ and $1 \leq j \leq d$. Please note in particular that $\boldsymbol{N}$ does not depend on c_0.

As a matrix over $\mathbb{F}_2$ with d columns and $d-1$ rows, there is necessarily a vector $\vec{w} \in \mathbb{F}_2^d$ with $\vec{w} \neq \vec{0}_d$ such that $\boldsymbol{N} \cdot \vec{w} = \vec{0}_{d-1}$.

The latter implies that the expression $s_0 = \sum_{i=1}^{d} w_i \cdot c_i$ does not functionally depend on any of the r_k's. Furthermore, by correctness, we also have that $s_1 = c_0 + \sum_{i=1}^{d} (1 - w_i) \cdot c_i$ does not functionally depend on any of the r_k's, and $s_0 + s_1 = \sum_{i=0}^{d} c_i = a \cdot b$. Then, the sets of probes $S_0 = \{c_i \mid w_i = 1\}$ and $S_1 = \{c_0\} \cup \{c_i \mid w_i = 0\}$ (whose cardinalities are at most d) satisfy the requirements of Lemma 4.2, and then, $\mathcal{A}$ is not d-private. Theorem 4.3 follows. □

4.3 Better Linear Lower Bound

We now show that at least $d+1$ random bits are actually required if $d \geq 3$.

Theorem 4.4. *Let $d \geq 3$. Let us consider a d-compression algorithm for multiplication $\mathcal{A}$. If $\mathcal{A}$ uses only d random bits, then $\mathcal{A}$ is not d-private.*

The full proof is in Appendix C.1.

4.4 (Non-Constructive) Quasi-Linear Upper Bound

We now construct a d-private compression algorithm for multiplication which requires a quasi-linear number of random bits. More precisely, we show that with non-zero probability, a random algorithm in some family of algorithms (using a quasi-linear number of random bits) is secure, which directly implies the existence of such an algorithm. Note that it is an interesting open problem (though probably difficult) to derandomize this construction.

Concretely, let d be some masking order and R be some number of random bits (used in the algorithm), to be fixed later. For $i = 0, \dots, d-1$ and $j = i+1, \dots, d$, let us define $\rho(i,j)$ as:

$$\rho(i,j) = \textstyle\sum_{k=1}^{R} X_{i,j,k} \cdot r_k$$

with $X_{i,j,k} \overset{\$}{\leftarrow} \{0,1\}$ for $i = 0, \dots, d-1$, $j = i+1, \dots, d$ and $k = 1, \dots, R$, so that $\rho(i,j)$ is a random sum of all the random bits $r_1, \dots, r_R$ where each bit appears in $\rho(i,j)$ with probability $1/2$. We also define $X_{d,d,k} = \sum_{i=0}^{d-1} \sum_{j=i+1}^{d} X_{i,j,k}$ and $\rho(d,d)$ as:

$$\rho(d,d) = \textstyle\sum_{k=1}^{R} X_{d,d,k} \cdot r_k.$$

We generate a (random) algorithm as in Algorithm 2. This algorithm is correct because the sum of all $\rho(i,j)$ is equal to 0.

We point out that we use two kinds of random which should not be confused: the R fresh random bits $r_1, \dots, r_R$ used in the algorithm to ensure its d-privacy (R is what we really want to be as low as possible), and the random variables $X_{i,j,k}$ used to define a random family of such algorithms (which are "meta"-random bits). In a concrete implementation or algorithm, these latter values are fixed.

Lemma 4.5. *Algorithm 2 is d-private with probability at least*

$$1 - \binom{(R+3) \cdot d \cdot (d+1)/2}{d} \cdot 2^{-R}$$

over the values of the $X_{i,j,k}$'s.

Algorithm 2 Random algorithm

```
Require: sharing $(\alpha_{i,j})_{0 \le i,j \le d}$
Ensure: sharing $(c_i)_{0 \le i \le d}$
  for $i = 1$ to $R$ do
    $r_i \xleftarrow{\$} \mathbb{F}_2$
  for $i = 0$ to $d$ do
    $c_i \leftarrow \alpha_{i,i}$
    for $j = i + 1$ to $d$ do
      $c_i \leftarrow c_i + \rho(i,j) + \alpha_{i,j} + \alpha_{j,i}$        ▷ $\rho(i,j)$ is not computed first
  $c_d \leftarrow c_d + \rho(d,d)$
```

Lemma 4.5. In order to simplify the proof, we are going to show that, with non-zero probability, there is no set of probes $P = \{p_1, \dots, p_\ell\}$ with $\ell \le d$ that satisfies Condition 2. In particular, this implies that, with non-zero probability, there is no set of probes $P = \{p_1, \dots, p_\ell\}$ with $\ell \le d$ that satisfies Condition 1, which, via Theorem 3.1, is equivalent to the algorithm being d-private.

One can only consider sets of exactly d probes as if there is a set of $\ell < d$ probes P' that satisfies Condition 2, one can always complete P' into a set P with exactly d probes by adding $d - \ell$ times the same probe on some input $\alpha_{i,j}$ such that P' initially does not depend on $\alpha_{i,j}$. That is, if $\boldsymbol{M}'$ denotes the matrix such that $\sum_{p' \in P'} p' = \vec{a} \cdot \boldsymbol{M}' \cdot \vec{b}$, one could complete P' with any $\alpha_{i,j}$ such that $m'_{i,j} = 0$, so that P, with $\sum_{p \in P} p = \vec{a} \cdot \boldsymbol{M} \cdot \vec{b}$ still satisfies Condition 2 if P' initially satisfied the condition.

Thus, let us consider an arbitrary set of d probes $P = \{p_1, \dots, p_d\}$ and let us bound the probability that P satisfies Condition 2. Let $f = \sum_{i=1}^{d} p_i$. Let us first show that f has to contain at least one $\rho(i,j)$ (meaning that it appears an odd number of times in the sum). Let us assume the contrary, so f does not contain any $\rho(i,j)$. Every $\rho(i,j)$ appears only once in the shares (in the share c_i precisely). Then, one can assume that every probe is made on the same share. Let us assume (without loss of generality) that every probe is made on c_0. If no probe contains any $\rho(0,j)$, then clearly P cannot satisfy Condition 2 as this means that each probe contain at most one $\alpha_{0,j}$, to P cannot contain more than d different $\alpha_{0,j}$. Hence, at least one (so at least two) probe contains at least one $\rho(0,j)$. We note that every probe has one of the following form: either it is exactly a random r_k, a share $\alpha_{0,j}$, a certain $\rho(0,j)$, a certain $\rho(0,j) + \alpha_{0,j}$ or $\rho(0,j) + \alpha_{0,j} + \alpha_{j,0}$, or a subsum (starting from $\alpha_{0,0}$) of c_0. Every form gives at most one $\alpha_{0,j}$ with a new index j except probes on subsums. However, in any subsum, there is always a random $\rho(i,j)$ between $\alpha_{0,j}$ and $\alpha_{0,j+1}$ and one needs to get all the $d+1$ indices to get a set satisfying Condition 2. Then, it is clear that one cannot achieve this unless there is a $\rho(i,j)$ that does not cancel out in the sum, which is exactly what we wanted to show. Now, let $1 \le k \le R$ be an integer and let us compute the probability (over the $X_{i,j,k}$'s) that f contains r_k. There exists some set S of pairs (i,j), such that f is the sum of $\sum_{(i,j) \in S} X_{i,j,k} \cdot r_k$ and some other expression not containing any $X_{i,j,k} \cdot r_k$. From the previous point, S is not empty. Furthermore, as there are $d+1$ outputs $c_0, \dots, c_d$ and as there are only d probes, S cannot contain all the possible pairs (i,j), and therefore, all the random variables $X_{i,j,k}$ for $(i,j) \in S$ are mutually independent. Therefore, $\sum_{(i,j) \in S} X_{i,j,k}$ is 1 with probability $1/2$ and f functionally depends on the random r_k with probability $1/2$. As there are R possible randoms, f does not functionally depend on any r_k (and then P satisfies Condition 2) with probability $(1/2)^R$.

There are N possibles probes with

$$N \le \frac{d \cdot (d+1)}{2} + R + (R+2) \cdot \frac{d \cdot (d-1)}{2} \le (R+3) \cdot \frac{d \cdot (d+1)}{2}$$

, as every ρ contains at most R random bits r_k. Also, there are $\binom{N}{d}$ possible sets $P = \{p_1, \dots, p_d\}$. Therefore, by union bound, the above algorithm is not secure (so there is an attack) with probability at most

$$\binom{N}{d} / 2^R \le \binom{(R+3) \cdot d \cdot (d+1)/2}{d} \cdot 2^{-R}$$

which concludes the proof of Lemma 4.5. □

Theorem 4.6. *For some $R = O(d \cdot \log d)$, there exists a choice of $\rho(i,j)$ such that Algorithm 2 is a d-private d-compression algorithm for multiplication, when $d \to \infty$.*

We just need to remark that for some $R = O(d \cdot \log d)$, the probability that Algorithm 2 is d-private, according to Lemma 4.5 is non-zero.

$$\begin{array}{lllllll}
\hat{\alpha}_{0,0} & r_{0,1} & r_{0,2} & r_{0,3} & r_{0,4} & r_{0,5} & r_{0,6} \\
\hat{\alpha}_{1,1} & (r_{0,1}\ \ \hat{\alpha}_{0,1}) & r_{1,2} & r_{1,3} & r_{1,4} & r_{1,5} & r_{1,6} \\
\hat{\alpha}_{2,2} & (r_{0,2}\ \ \hat{\alpha}_{0,2}) & (r_{1,2}\ \ \hat{\alpha}_{1,2}) & r_{2,3} & r_{2,4} & r_{2,5} & r_{2,6} \\
\hat{\alpha}_{3,3} & (r_{0,3}\ \ \hat{\alpha}_{0,3}) & (r_{1,3}\ \ \hat{\alpha}_{1,3}) & (r_{2,3}\ \ \hat{\alpha}_{2,3}) & r_{3,4} & r_{3,5} & r_{3,6} \\
\hat{\alpha}_{4,4} & (r_{0,4}\ \ \hat{\alpha}_{0,4}) & (r_{1,4}\ \ \hat{\alpha}_{1,4}) & (r_{2,4}\ \ \hat{\alpha}_{2,4}) & (r_{3,4}\ \ \hat{\alpha}_{3,4}) & r_{4,5} & r_{4,6} \\
\hat{\alpha}_{5,5} & (r_{0,5}\ \ \hat{\alpha}_{0,5}) & (r_{1,5}\ \ \hat{\alpha}_{1,5}) & (r_{2,5}\ \ \hat{\alpha}_{2,5}) & (r_{3,5}\ \ \hat{\alpha}_{3,5}) & (r_{4,5}\ \ \hat{\alpha}_{4,5}) & r_{5,6} \\
\hat{\alpha}_{6,6} & (r_{0,6}\ \ \hat{\alpha}_{0,6}) & (r_{1,6}\ \ \hat{\alpha}_{1,6}) & (r_{2,6}\ \ \hat{\alpha}_{2,6}) & (r_{3,6}\ \ \hat{\alpha}_{3,6}) & (r_{4,6}\ \ \hat{\alpha}_{4,6}) & (r_{5,6}\ \ \hat{\alpha}_{5,6})
\end{array}$$

Figure 2: ISW construction for $d = 6$, with $\hat{\alpha}_{i,j} = \alpha_{i,j} + \alpha_{j,i}$

The full proof is given in Appendix C.2.

5 New Construction

The goal of this section is to propose a new d-private multiplication algorithm. Compared to the construction in [ISW03], our construction halves the number of required random bits. It is therefore the most efficient existing construction of a d-private multiplication.

Some rationales behind our new construction may be found in the two following necessary conditions deduced from a careful study of the original work of Ishai, Sahai and Wagner [ISW03].

Lemma 5.1. *Let $\mathcal{A}(\vec{a}, \vec{b}; \vec{r})$ be a d-compression algorithm for multiplication. Let f be an intermediate result taking the form $f = \vec{a}^\intercal \cdot \boldsymbol{M} \cdot \vec{b} + \vec{s}^\intercal \cdot \vec{r}$. Let t denote the greatest Hamming weight of an element in the vector subspace generated by the rows of $\boldsymbol{M}$ or by the columns of $\boldsymbol{M}$. If $\mathsf{hw}(\vec{s}) < t-1$, then $\mathcal{A}(\vec{a}, \vec{b}; \vec{r})$ is not d-private.*

Proof. By definition of $\vec{s}$, the value $\vec{a}^\intercal \cdot \boldsymbol{M} \cdot \vec{b}$ can be recovered by probing f and then each of the $\mathsf{hw}(\vec{s}) < t - 1$ random bits on which $\vec{s}^\intercal \cdot \vec{r}$ functionally depends and by summing all these probes. Let $P_1 = \{f, p_1, \dots, p_j\}$ with $j < t - 1$ denote the set of these at most $t - 1$ probes. Then, we just showed that $f + \sum_{i=1}^{j} p_i = \vec{a}^\intercal \cdot \boldsymbol{M} \cdot \vec{b}$.

To conclude the proof, we want to argue that there is a set of at most $d - (t - 1)$ probes $P_2 = \{p'_1, \dots, p'_k\}$ such that $f + \sum_{i=1}^{j} p_i + \sum_{\ell=1}^{k} p'_\ell = \vec{a}^\intercal \cdot \boldsymbol{M'} \cdot \vec{b}$, where $\boldsymbol{M'}$ is a matrix such that $\vec{u}_{d+1}$ is in its row space or in its column space. If such a set P_2 exists, then the set of probes $P_1 \cup P_2$ (whose cardinality is at most d) satisfies Condition 1, and then $\mathcal{A}$ is not d-private, via Theorem 3.1.

We now use the fact that there is a vector of Hamming weight t in the row space or in the column space of $\boldsymbol{M}$. We can assume (without loss of generality) that there exists a vector $\vec{w} \in \mathbb{F}_2^{d+1}$ of Hamming weight t in the column subspace of $\boldsymbol{M}$, so that $\vec{w} = \sum_{j \in J} \vec{m}_j$, with $J \subseteq \{0, \dots, d\}$ and $\vec{m}_j$ the j-th column vector of $\boldsymbol{M}$. Let $i_1, \dots, i_{d+1-t}$ denote the indices i of $\vec{w}$ such that $w_i = 0$. Then, let $j \in J$, we claim that $P_2 = \{\alpha_{i_1,j}, \dots, \alpha_{i_{d+1-t},j}\}$ allows us to conclude the proof. Please note that all these values are probes of intermediate values of $\mathcal{A}$.

Indeed, we have $f + \sum_{i=1}^{j} p_i + \sum_{k=1}^{d+1-t} \alpha_{i_k,j} = \vec{a}^\intercal \cdot \boldsymbol{M'} \cdot \vec{b}$ where all coefficients of $\boldsymbol{M'}$ are the same as coefficients of $\boldsymbol{M}$ except for coefficients in positions $(i_1, j), \dots, (i_{d+1-t}, j)$ which are the opposite, and now $\sum_{j \in J} \vec{m}'_j = \vec{u}_{d+1}$, where $\vec{m}'_j$ is the j-th column vector of $\boldsymbol{M'}$. Lemma 5.1 easily follows. □

In our construction, we satisfy the necessary condition in Lemma 5.1 by ensuring that any intermediate result that functionally depends on t shares of a (resp. of b) also functionally depends on at least $t - 1$ random bits.

The multiplication algorithm of Ishai, Sahai and Wagner is the starting point of our construction. Before exhibiting it, we hence start by giving the basic ideas thanks to an illustration in the particular case $d = 6$. In Figure 2 we recall the description of ISW already introduced in Section 2.3.

The first step of our construction is to order the expressions $\hat{\alpha}_{i,j}$ differently. Precisely, to compute the output share c_i (which corresponds, in ISW, to the sum $r_{i,i,+} \sum_{j<i}(r_{j,i} + \hat{\alpha}_{j,i}) + \sum_{j>i} r_{i,j}$ from left to right), we process $r_{i,i,+} \sum_{j<d-i}(r_{i,d-j} + \hat{\alpha}_{i,j}) + \sum_{1 \le j \le i} r_{d-j,i}$ from left to right. Of course, we also put particular care to satisfy the necessary condition highlighted by Lemma 5.1. This leads to the construction illustrated in Figure 3.

Then, the core idea is to decrease the randomness cost by reusing some well chosen random bit to protect different steps of the processing. Specifically, for any even positive number k, we show that replacing all the random bits $r_{i,j}$ such that $k = j - i$ with a fixed random bit r_k preserves the d-privacy of ISW algorithm. Note, however, that the computations then have to be performed with a slightly

$$\begin{array}{ccccccccccccc}
\hat{\alpha}_{0,0} & (r_{0,6} & \hat{\alpha}_{0,6}) & (r_{0,5} & \hat{\alpha}_{0,5}) & (r_{0,4} & \hat{\alpha}_{0,4}) & (r_{0,3} & \hat{\alpha}_{0,3}) & (r_{0,2} & \hat{\alpha}_{0,2}) & (r_{0,1} & \hat{\alpha}_{0,1}) \\
\hat{\alpha}_{1,1} & (r_{1,6} & \hat{\alpha}_{1,6}) & (r_{1,5} & \hat{\alpha}_{1,5}) & (r_{1,4} & \hat{\alpha}_{1,4}) & (r_{1,3} & \hat{\alpha}_{1,3}) & (r_{1,2} & \hat{\alpha}_{1,2}) & r_{0,1} & \\
\hat{\alpha}_{2,2} & (r_{2,6} & \hat{\alpha}_{2,6}) & (r_{2,5} & \hat{\alpha}_{2,5}) & (r_{2,4} & \hat{\alpha}_{2,4}) & (r_{2,3} & \hat{\alpha}_{2,3}) & r_{1,2} & & r_{0,2} & \\
\hat{\alpha}_{3,3} & (r_{3,6} & \hat{\alpha}_{3,6}) & (r_{3,5} & \hat{\alpha}_{3,5}) & (r_{3,4} & \hat{\alpha}_{3,4}) & r_{2,3} & & r_{1,3} & & r_{0,3} & \\
\hat{\alpha}_{4,4} & (r_{4,6} & \hat{\alpha}_{4,6}) & (r_{4,5} & \hat{\alpha}_{4,5}) & r_{3,4} & & r_{2,4} & & r_{1,4} & & r_{0,4} & \\
\hat{\alpha}_{5,5} & (r_{5,6} & \hat{\alpha}_{5,6}) & r_{4,5} & & r_{3,5} & & r_{2,5} & & r_{1,5} & & r_{0,5} & \\
\hat{\alpha}_{6,6} & r_{5,6} & & r_{4,6} & & r_{3,6} & & r_{2,6} & & r_{1,6} & & r_{0,6} &
\end{array}$$

Figure 3: First step of our new construction for $d = 6$, with $\hat{\alpha}_{i,j} = \alpha_{i,j} + \alpha_{j,i}$

$$\begin{array}{ccccccccccccc}
\hat{\alpha}_{0,0} & (r_{0,6} & \hat{\alpha}_{0,6} & r_5 & \hat{\alpha}_{0,5}) & (r_{0,4} & \hat{\alpha}_{0,4} & r_3 & \hat{\alpha}_{0,3}) & (r_{0,2} & \hat{\alpha}_{0,2} & r_1 & \hat{\alpha}_{0,1}) \\
\hat{\alpha}_{1,1} & (r_{1,6} & \hat{\alpha}_{1,6} & r_5 & \hat{\alpha}_{1,5}) & (r_{1,4} & \hat{\alpha}_{1,4} & r_3 & \hat{\alpha}_{1,3}) & (r_{1,2} & \hat{\alpha}_{1,2}) & r_1 & \\
\hat{\alpha}_{2,2} & (r_{2,6} & \hat{\alpha}_{2,6} & r_5 & \hat{\alpha}_{2,5}) & (r_{2,4} & \hat{\alpha}_{2,4} & r_3 & \hat{\alpha}_{2,3}) & r_{1,2} & & r_{0,2} & \\
\hat{\alpha}_{3,3} & (r_{3,6} & \hat{\alpha}_{3,6} & r_5 & \hat{\alpha}_{3,5}) & (r_{3,4} & \hat{\alpha}_{3,4}) & r_3 & & r_3 & & r_3 & \\
\hat{\alpha}_{4,4} & (r_{4,6} & \hat{\alpha}_{4,6} & r_5 & \hat{\alpha}_{4,5}) & r_{3,4} & & r_{2,4} & & r_{1,4} & & r_{0,4} & \\
\hat{\alpha}_{5,5} & (r_{5,6} & \hat{\alpha}_{5,6}) & r_5 & & r_5 & & r_5 & & r_5 & & r_5 & \\
\hat{\alpha}_{6,6} & r_{5,6} & & r_{4,6} & & r_{3,6} & & r_{2,6} & & r_{1,6} & & r_{0,6} &
\end{array}$$

Figure 4: Second step of our new construction for $d = 6$, with $\hat{\alpha}_{i,j} = \alpha_{i,j} + \alpha_{j,i}$

different bracketing in order to protect the intermediate variables which involve the same random bits. The obtained construction is illustrated in Figure 4.

Finally, we suppress from our construction the useless repetitions of random bits that appear at the end of certain computations. Hence, we obtain our new construction, illustrated in Figure 5.

Before proving that this scheme is indeed d-private, we propose a formal description in Algorithm 3. As can be seen, this new scheme involves $3d^2/2 + d(d + 2)/4 + 2d$ sums if d is even and $3(d^2 - 1)/2 + (d + 1)^2/4 + 3(d + 1)/2$ if d is odd. In every case, it also involves $(d + 1)^2$ multiplications and requires the generation of $d^2/4 + d$ random values in $\mathbb{F}_2$ if d is even and $(d^2 - 1)/4 + d$ otherwise (see Table 1 for values at several orders and comparison with ISW).

Proposition 5.2. *Algorithm 3 is d-private.*

Algorithm 3 was proven to be d-private with the verifier built by Barthe et al. [BBD+15b] up to order $d = 6$. Furthermore, a pen-and-paper proof for any order d is given in Appendix D.

6 Optimal Small Cases

We propose three secure compression algorithms using less random bits than the generic solution given by ISW and than our new solution for the specific small orders $d = 2$, 3 and 4. These algorithms actually use only the optimal numbers of random bits for these small quantity of probes, as proven in Section 4. Furthermore, since they all are dedicated to a specific order d (among 2, 3, and 4), we got use of the verifier proposed by Barthe et al. in [BBD+15b] to formally prove their correctness and their d-privacy.

Proposition 6.1. *Algorithms 4, 5, and 6 are correct and respectively $2, 3$ and 4-private.*

Table 1 (Section 5) compares the amount of randomness used by the new construction proposed in Section 5 and by our optimal small algorithms. We recall that each of them attains the lower bound proved in Section 4.

$$\begin{array}{ccccccccccccc}
\hat{\alpha}_{0,0} & (r_{0,6} & \hat{\alpha}_{0,6} & r_5 & \hat{\alpha}_{0,5}) & (r_{0,4} & \hat{\alpha}_{0,4} & r_3 & \hat{\alpha}_{0,3}) & (r_{0,2} & \hat{\alpha}_{0,2} & r_1 & \hat{\alpha}_{0,1}) \\
\hat{\alpha}_{1,1} & (r_{1,6} & \hat{\alpha}_{1,6} & r_5 & \hat{\alpha}_{1,5}) & (r_{1,4} & \hat{\alpha}_{1,4} & r_3 & \hat{\alpha}_{1,3}) & (r_{1,2} & \hat{\alpha}_{1,2}) & r_1 & \\
\hat{\alpha}_{2,2} & (r_{2,6} & \hat{\alpha}_{2,6} & r_5 & \hat{\alpha}_{2,5}) & (r_{2,4} & \hat{\alpha}_{2,4} & r_3 & \hat{\alpha}_{2,3}) & r_{1,2} & & r_{0,2} & \\
\hat{\alpha}_{3,3} & (r_{3,6} & \hat{\alpha}_{3,6} & r_5 & \hat{\alpha}_{3,5}) & (r_{3,4} & \hat{\alpha}_{3,4}) & r_3 & & & & & \\
\hat{\alpha}_{4,4} & (r_{4,6} & \hat{\alpha}_{4,6} & r_5 & \hat{\alpha}_{4,5}) & r_{3,4} & & r_{2,4} & & r_{1,4} & & r_{0,4} & \\
\hat{\alpha}_{5,5} & (r_{5,6} & \hat{\alpha}_{5,6}) & r_5 & & & & & & & & & \\
\hat{\alpha}_{6,6} & r_{5,6} & & r_{4,6} & & r_{3,6} & & r_{2,6} & & r_{1,6} & & r_{0,6} &
\end{array}$$

Figure 5: Application of our new construction for $d = 6$, with $\hat{\alpha}_{i,j} = \alpha_{i,j} + \alpha_{j,i}$

Algorithm 3 New construction for d-secure multiplication

```
Require: sharing $(\alpha_{i,j})_{0 \le i,j \le d}$
Ensure: sharing $(c_i)_{0 \le i \le d}$
 1: for $i = 0$ to $d$ do                                              ▷ Random Bits Generation
 2:     for $j = 0$ to $d - i - 1$ by 2 do
 3:         $r_{i,d-j} \xleftarrow{\$} \mathbb{F}_2$
 4: for $j = d - 1$ downto 1 by 2 do
 5:     $r_j \xleftarrow{\$} \mathbb{F}_2$
 6: for $i = 0$ to $d$ do                                              ▷ Multiplication
 7:     $c_i \leftarrow \alpha_{i,i}$
 8:     for $j = d$ downto $i + 2$ by 2 do
 9:         $t_{i,j} \leftarrow r_{i,j} + \alpha_{i,j} + \alpha_{j,i} + r_{j-1} + \alpha_{i,j-1} + \alpha_{j-1,i}$;    $c_i \leftarrow c_i + t_{i,j}$
10:     if $i \not\equiv d \pmod 2$ then
11:         $t_{i,i+1} \leftarrow r_{i,i+1} + \alpha_{i,i+1} + \alpha_{i+1,i}$;    $c_i \leftarrow c_i + t_{i,i+1}$
12:         if $i \equiv 1 \pmod 2$ then                                 ▷ Correction $r_i$
13:             $c_i \leftarrow c_i + r_i$
14:     else
15:         for $j = i - 1$ downto 0 do                                  ▷ Correction $r_{i,j}$
16:             $c_i \leftarrow c_i + r_{j,i}$
```

Table 1: Complexities of ISW, our new d-private compression algorithm for multiplication and our specific algorithms at several orders

Complexities	Algorithm ISW	Algorithm 3	Algorithms 4, 5 and 6
Second-Order Masking			
sums	12	12	10
products	9	9	9
random bits	3	3	2
Third-Order Masking			
sums	24	22	20
products	16	16	16
random bits	6	5	4
Fourth-Order Masking			
sums	40	38	30
products	25	25	25
random bits	10	8	5
d^{th}-Order Masking			
sums	$2d(d+1)$	$\begin{cases} d(7d+10)/4 & (d \text{ even}) \\ (7d+1)(d+1)/4 & (d \text{ odd}) \end{cases}$	-
products	$(d+1)^2$	$(d+1)^2$	-
random bits	$d(d+1)/2$	$\begin{cases} d^2/4 + d & (d \text{ even}) \\ (d^2-1)/4 + d & (d \text{ odd}) \end{cases}$	-

Algorithm 4 Second-Order Compression Algorithm

```
Require: sharing $(\alpha_{i,j})_{0 \le i,j \le 2}$
Ensure: sharing $(c_i)_{0 \le i \le 2}$
  $r_0 \xleftarrow{\$} \mathbb{F}_2$;    $r_1 \leftarrow \mathbb{F}_2$
  $c_0 \leftarrow \alpha_{0,0} + r_0 + \alpha_{0,2} + \alpha_{2,0}$
  $c_1 \leftarrow \alpha_{1,1} + r_1 + \alpha_{0,1} + \alpha_{1,0}$
  $c_2 \leftarrow \alpha_{2,2} + r_0 + r_1 + \alpha_{1,2} + \alpha_{2,1}$
```

Algorithm 5 Third-Order Compression Algorithm

```
Require: sharing $(\alpha_{i,j})_{0 \le i,j \le 3}$
Ensure: sharing $(c_i)_{0 \le i \le 3}$
  $r_0 \xleftarrow{\$} \mathbb{F}_2$;    $r_1 \xleftarrow{\$} \mathbb{F}_2$;    $r_2 \xleftarrow{\$} \mathbb{F}_2$;    $r_3 \xleftarrow{\$} \mathbb{F}_2$
  $c_0 \leftarrow \alpha_{0,0} + r_0 + \alpha_{0,3} + \alpha_{3,0} + r_1 + \alpha_{0,2} + \alpha_{2,0}$
  $c_1 \leftarrow \alpha_{1,1} + r_2 + \alpha_{1,3} + \alpha_{3,1} + r_1 + \alpha_{1,2} + \alpha_{2,1}$
  $c_2 \leftarrow \alpha_{2,2} + r_3 + \alpha_{2,3} + \alpha_{3,2}$
  $c_3 \leftarrow \alpha_{3,3} + r_3 + r_2 + r_0 + \alpha_{0,1} + \alpha_{1,0}$
```

Algorithm 6 Fourth-Order Compression Algorithm

```
Require: sharing $(\alpha_{i,j})_{0\leq i,j\leq 4}$
Ensure: sharing $(c_i)_{0\leq i\leq 4}$
  $r_0 \xleftarrow{\$} \mathbb{F}_2$;   $r_1 \xleftarrow{\$} \mathbb{F}_2$;   $r_2 \xleftarrow{\$} \mathbb{F}_2$;   $r_3 \xleftarrow{\$} \mathbb{F}_2$;   $r_4 \xleftarrow{\$} \mathbb{F}_2$
  $c_0 \leftarrow \alpha_{0,0} + r_0 + \alpha_{0,1} + \alpha_{1,0} + r_1 + \alpha_{0,2} + \alpha_{2,0}$
  $c_1 \leftarrow \alpha_{1,1} + r_1 + \alpha_{1,2} + \alpha_{2,1} + r_2 + \alpha_{1,3} + \alpha_{3,1}$
  $c_2 \leftarrow \alpha_{2,2} + r_2 + \alpha_{2,3} + \alpha_{3,2} + r_3 + \alpha_{2,4} + \alpha_{4,2}$
  $c_3 \leftarrow \alpha_{3,3} + r_3 + \alpha_{3,4} + \alpha_{4,3} + r_4 + \alpha_{3,0} + \alpha_{0,3}$
  $c_4 \leftarrow \alpha_{4,4} + r_4 + \alpha_{4,0} + \alpha_{0,4} + r_0 + \alpha_{4,1} + \alpha_{1,4}$
```

7 Composition

Our new algorithms are all d-private, when applied on the outputs of a multiplicative encoder parameterized at order d. We now aim to show how they can be involved in the design of larger functions (e.g., block ciphers) to achieve a global d-privacy. In [BBD+15a], Barthe et al. introduce and formally prove a method to compose small d-private algorithms (a.k.a., *gadgets*) into d-private larger functions. The idea is to carefully refresh the sharings when necessary, according to the security properties of the gadgets. Before going further into the details of this composition, we recall some security properties used in [BBD+15a].

7.1 Compositional Security Notions

Before stating the new security definitions, we first need to introduce the notion of simulatability. For the sake of simplicity, we only state this notion for multiplication algorithm, but this can easily be extended to more general algorithms.

Definition 7.1. A set $P = \{p_1, \ldots, p_\ell\}$ of ℓ probes of a multiplication algorithm can be *simulated* with at most t shares of each input, if there exists two sets $I = \{i_1, \ldots, i_t\}$ and $J = \{j_1, \ldots, j_t\}$ of t indices from $\{0, \ldots, d\}$ and a random function f taking as input $2t$ bits and outputting ℓ bits such that for any fixed bits $(a_i)_{0\leq i\leq d}$ and $(b_j)_{0\leq j\leq d}$, the distributions $\{p_1, \ldots, p_\ell\}$ (which implicitly depends on $(a_i)_{0\leq i\leq d}$, $(b_j)_{0\leq j\leq d}$, and the random coins used in the multiplication algorithm) and $\{f(a_{i_1}, \ldots, a_{i_t}, b_{j_1}, \ldots, b_{j_t})\}$ are identical.

We write $f(a_{i_1}, \ldots, a_{i_t}, b_{j_1}, \ldots, b_{j_t}) = f(a_I, b_J)$.

Definition 7.2. An algorithm is *d-non-interferent (or d-NI)* if and only if every set of at most d probes can be simulated with at most d shares of each input.

While this notion might be stronger than the notion of security we used, all our concrete constructions in Sections 5 and 6 satisfy it. The proof of Algorithm 3 is indeed a proof by simulation, while the small cases in Section 6 are proven using the verifier by Barthe et al. in [BBD+15b], which directly proves NI.

Definition 7.3. An algorithm is *d-tight non-interferent (or d-TNI)* if and only if every set of $t \leq d$ probes can be simulated with at most t shares of each input.

While this notion of d-tight non-interference was assumed to be stronger than the notion of d-non-interference in [BBD+15a], we show hereafter that these two security notions are actually equivalent. In particular, this means that all our concrete constructions are also TNI.

Proposition 7.4. *(d-**NI** $\Leftrightarrow$ d-**TNI**) An algorithm is d-non-interferent if and only if it is d-tight non-interferent.*

Proof. The right-to-left implication is straightforward from the definitions. Let us thus consider the left-to-right direction.

For that purpose, we first need to introduce a technical lemma. Again, for the sake of simplicity, we only consider multiplication algorithm, with only two inputs, but the proof can easily be generalized to any algorithm.

Lemma 7.5. *Let $P = \{p_1, \ldots, p_\ell\}$ be a set of ℓ probes which can be simulated by the sets (I, J) and also by the sets (I', J'). Then it can also be simulated by $(I \cap I', J \cap J')$.*

Proof. Let f the function corresponding to I, J and f' the function corresponding to I', J'. We have that for any bits $(a_i)_{0 \le i \le d}$ and $(b_j)_{0 \le j \le d}$, the distributions $\{p_1, \dots, p_\ell\}$, $\{f(a_I, b_J)\}$, and $\{f'(a_{I'}, b_{J'})\}$ are identical. Therefore, f does not depend on a_i nor b_j for $i \in I \setminus I'$ and $j \in J \setminus J'$, since f' does not depend on them. Thus, P can be simulated by only shares from $I \cap I'$, $J \cap J'$ (using the function f where the inputs corresponding to a_i and b_j for $i \in I \setminus I'$ and $j \in J \setminus J'$ are just set to zero, for example). □

We now assume that an algorithm $\mathcal{A}$ is d-NI, that is, every set of at most d probes can be simulated with at most d shares of each input. Now, by contradiction, let us consider a set P with minimal cardinality $t < d$ of probes on $\mathcal{A}$, such that it cannot be simulated by at most t shares of each input. Let us consider the sets I, J corresponding to the intersection of all sets I', J' (respectively) such that the set P can be simulated by I', J'. The sets I, J also simulate P thanks to Lemma 7.5. Furthermore, by hypothesis, $t < |I| \le d$ or $t < |J| \le d$. Without loss of generality, let us suppose that $|I| > t$.

Let i^* be an arbitrary element of $\{0, \dots, d\} \setminus I$ (which is not an empty set as $|I| \le d$). Let us now consider the set of probes $P' = P \cup \{a_{i^*}\}$. By hypothesis, P' can be simulated by at most $|P'| = t + 1$ shares of each input. Let I', J' two sets of size at most $t+1$ simulating P'. These two sets also simulate $P \subseteq P'$, therefore, $I \cap I', J \cap J'$ also simulate P. Furthermore, $i^* \in I$, as all the shares a_i are independent. Since $i^* \notin I$, $|I \cap I'| \le t$ and $I \cap I' \subsetneq I$, which contradicts the fact that I and J were the intersection of all sets I'', J'' simulating P. □

Definition 7.6. An algorithm $\mathcal{A}$ is *d-strong non-interferent (or d-SNI)* if and only if for every set $\mathcal{I}$ of t_1 probes on intermediate variables (i.e., no output wires or shares) and every set $\mathcal{O}$ of t_2 probes on output shares such that $t_1 + t_2 \le d$, the set $\mathcal{I} \cup \mathcal{O}$ of probes can be simulated by only t_1 shares of each input.

The composition of two d-SNI algorithms is itself d-SNI, while that of d-TNI algorithms is not necessarily d-TNI. This implies that d-SNI gadgets can be directly composed while maintaining the d-privacy property, whereas a so-called *refreshing* gadget must sometimes be involved before the composition of d-TNI algorithms. Since the latter refreshing gadgets consume the same quantity of random values as ISW, limiting their use is crucial if the goal is to reduce the global amount of randomness.

7.2 Building Compositions with our New Algorithms

In [BBD+15a], the authors show that the ISW multiplication is d-SNI and use it to build secure compositions. Unfortunately, our new multiplication algorithms are d-TNI but not d-SNI. Therefore, as discussed in the previous section, they can replace only some of the ISW multiplications in secure compositions. Let us take the example of the AES inversion that is depicted in [BBD+15a]. We can prove that replacing the first ($\mathcal{A}^7$) and the third ($\mathcal{A}^2$) ISW multiplications by d-TNI multiplications (e.g., our new constructions) and moving the refreshing algorithm R in different locations preserves the strong non-interference of the inversion, while benefiting from our reduction of the randomness consumption.

The tweaked inversion is given in Figure 6. $\otimes$ denotes the d-SNI ISW multiplication, $\cdot^\alpha$ denotes the exponentiation to the power α, $\mathcal{A}^i$ refers to the i-th algorithm or gadget (indexed from left to right), R denotes the d-SNI refreshing gadget, $\mathcal{I}^i$ denotes the set of internal probes in the i-th algorithm, $\mathcal{S}^i_j$ denotes the set of shares from the j inputs of algorithm $\mathcal{A}^i$ used to simulate all further probes. Finally, x denotes the inversion input and $\mathcal{O}$ denotes the set of probes at the output of the inversion. The global constraint for the inversion to be d-SNI (and thus itself composable) is that: $|\mathcal{S}^8 \cup \mathcal{S}^9| \le \sum_{1 \le i \le 9} |\mathcal{I}^i|$, i.e., all the internal probes can be perfectly simulated with at most $\sum_{1 \le i \le 9} |\mathcal{I}^i|$ shares of x.

Proposition 7.7. *The AES inversion given in Figure 6 with $\mathcal{A}^1$ and $\mathcal{A}^4$ being d-SNI multiplications and $\mathcal{A}^2$ and $\mathcal{A}^7$ being d-TNI multiplications is d-SNI.*

Proof. From the d-probing model, we assume that the total number of probes used to attack the inversion is limited to d, that is $\sum_{1 \le i \le 9} |\mathcal{I}^i| + |\mathcal{O}| \le d$. As in [BBD+15a], we build the proof from right to left by simulating each algorithm. Algorithm $\mathcal{A}^1$ is d-SNI, thus $|\mathcal{S}^1_1|, |\mathcal{S}^1_2| \le |\mathcal{I}^1|$. Algorithm $\mathcal{A}^2$ is d-TNI, thus $|\mathcal{S}^2_1|, |\mathcal{S}^2_2| \le |\mathcal{I}^1 + \mathcal{I}^2|$. As explained in [BBD+15a], since Algorithm $\mathcal{A}^3$ is affine, then $|\mathcal{S}^3| \le |\mathcal{S}^2_1 + \mathcal{I}^3| \le |\mathcal{I}^1 + \mathcal{I}^2 + \mathcal{I}^3|$. Algorithm $\mathcal{A}^4$ is d-SNI, thus $|\mathcal{S}^4_1|, |\mathcal{S}^4_2| \le |\mathcal{I}^4|$. Algorithm $\mathcal{A}^5$ is d-SNI, thus $|\mathcal{S}^5| \le |\mathcal{I}^5|$. Algorithm $\mathcal{A}^6$ is affine, thus $|\mathcal{S}^6| \le |\mathcal{S}^5 + \mathcal{I}^6| \le |\mathcal{I}^5 + \mathcal{I}^6|$. Algorithm $\mathcal{A}^7$ is d-TNI, thus $|\mathcal{S}^7_1|, |\mathcal{S}^7_2| \le |\mathcal{S}^6 + \mathcal{S}^4_1 + \mathcal{I}^7| \le |\mathcal{I}^4 + \mathcal{I}^5 + \mathcal{I}^6 + \mathcal{I}^7|$. Algorithm $\mathcal{A}^8$ is d-SNI, thus $|\mathcal{S}^8| \le |\mathcal{I}^8|$. Algorithm $\mathcal{A}^9$ is affine, thus $|\mathcal{S}^9| \le |\mathcal{I}^9 + \mathcal{S}^8| \le |\mathcal{I}^8 + \mathcal{I}^9|$. Finally, all the probes of this inversion can be perfectly simulated from $|\mathcal{S}^9 \cup \mathcal{S}^7_1| \le |\mathcal{I}^4 + \mathcal{I}^5 + \mathcal{I}^6 + \mathcal{I}^7 + \mathcal{I}^8 + \mathcal{I}^9|$ shares of x, which proves that the inversion is still d-SNI. □

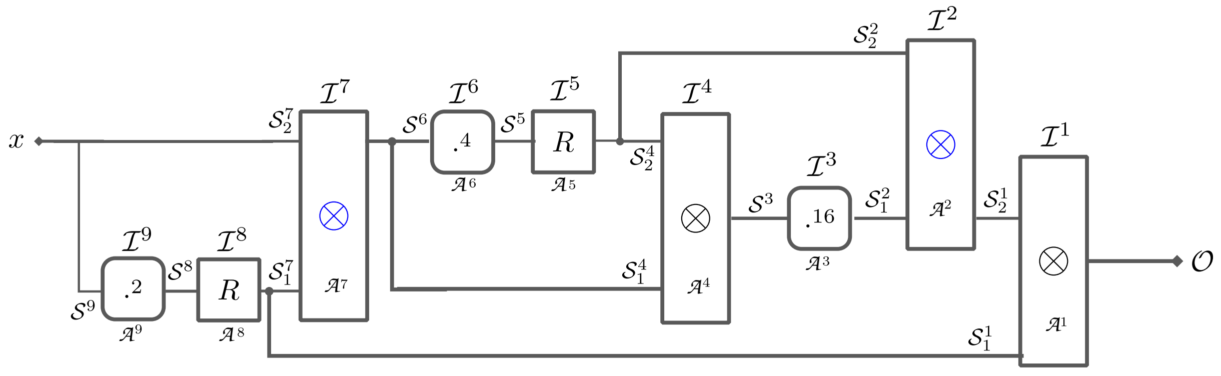

Figure 6: AES $\cdot^{254}$

From Proposition 7.7, our new constructions can be used to build d-SNI algorithms. In the case of the AES block cipher, half of the d-SNI ISW multiplications can be replaced by ours while preserving the whole d-SNI security.

8 New Automatic Tool for Finding Attacks

In this section, we describe a new automatic tool for finding attacks on compression algorithms for multiplication which is developed in Sage (Python) [Sage]. Compared to the verifier developed by Barthe *et al.* [BBD+15b] and based on Easycrypt, to find attacks in practice, our tool is not as generic as it focuses on compression algorithms for multiplication and its soundness is not perfect (and relies on some heuristic assumption). Nevertheless, it is order of magnitudes faster.

A non-perfect soundness means that the algorithm may not find an attack and can only guarantee that there does not exist an attack except with probability ε. We believe that, in practice, this limitation is not a big issue as if ε is small enough (e.g., 2^{-20}), a software bug is much more likely than an attack on the scheme. Furthermore, the running time of the algorithm depends only linearly on $\log(1/\varepsilon)$. Concretely, for all the schemes we manually tested for $d = 3, 4, 5$ and 6, attacks on invalid schemes were found almost immediately. If not used to formally prove schemes, our tool can at least be used to quickly eliminate (most) incorrect schemes, and enables to focus efforts on trying to prove "non-trivially-broken" schemes.

8.1 Algorithm of the Tool

From Theorem 3.1, in order to find an attack $P = \{p_1, \dots, p_\ell\}$ with $\ell \leq d$, we just need to find a set $P = \{p_1, \dots, p_\ell\}$ satisfying Condition 1. If no such set P exists, the compression algorithm for multiplication is d-private.

A naive way to check the existence of such a set P is to enumerate all the sets of d probes. However, there are $\binom{N}{d}$ such sets, with N being the number of intermediate variables of the algorithm. For instance, to achieve 4-privacy, our construction (see Section 6) uses $N = 81$ intermediate variables, which makes more than 2^{20} sets of four variables to test. In [BBD+15b], the authors proposed a faster way of enumerating these sets by considering larger sets which are still independent from the secret. However, their method falls short for the compression algorithms in our paper as soon as $d > 6$, as shown in Section 8.4. Furthermore even for $d = 3, 4, 5$, their tool takes several minutes to prove security (around 5 minutes to check security of Algorithm 3 with $d = 5$) or to find an attack for incorrect schemes, which prevent people from quickly checking the validity of a newly designed scheme.

To counteract this issue, we design a new tool which is completely different and which borrows ideas from coding theory to enumerate the sets of d or less intermediate variables. Let $\gamma_1, \dots, \gamma_\nu$ be all the intermediate results whose expression functionally depends on at least one random and $\gamma'_1, \dots, \gamma'_{\nu'}$ be the other intermediate results that we refer to as deterministic intermediate results ($\nu + \nu' = N$). We remark that all the $\alpha_{i,j} = a_i b_j$ are intermediate results and that no intermediate result can functionally depend on more than one shares' product $\alpha_{i,j} = a_i b_j$ without also depending on a random bit. Otherwise, the compression algorithm would not be d-private, according to Lemma 5.1. As this condition can be easily tested, we now assume that the only deterministic intermediate results are the $\alpha_{i,j} = a_i b_j$ that we refer to as γ'_k in the following. As an example, intermediate results of Algorithm 4 are depicted in Table 2.

Table 2: Intermediate results of Algorithm 4

non-deterministic ($\nu = 12$)		deterministic ($\nu' = 9$)	
$\gamma_1 = a_0 b_0 + r_0$	$\gamma_7 = c_1$	$\gamma'_1 = a_0 b_0$	$\gamma'_6 = a_1 b_0$
$\gamma_2 = a_0 b_0 + r_0 + a_0 b_2$	$\gamma_8 = r_1$	$\gamma'_2 = a_0 b_2$	$\gamma'_7 = a_2 b_2$
$\gamma_3 = c_0$	$\gamma_9 = a_2 b_2 + r_1$	$\gamma'_3 = a_2 b_0$	$\gamma'_8 = a_1 b_2$
$\gamma_4 = r_0$	$\gamma_{10} = a_2 b_2 + r_1 + r_0$	$\gamma'_4 = a_1 b_1$	$\gamma'_9 = a_2 b_1$
$\gamma_5 = a_1 b_1 + r_1$	$\gamma_{11} = a_2 b_2 + r_1 + r_0 + a_1 b_2$	$\gamma'_5 = a_0 b_1$	
$\gamma_6 = a_1 b_1 + r_1 + a_0 b_1$	$\gamma_{12} = c_2$		

An attack set $P = \{p_1, \dots, p_\ell\}$ can then be separated into two sets $Q = \{\gamma_{i_1}, \dots, \gamma_{i_\delta}\}$ and $Q' = \{\gamma_{i'_1}, \dots, \gamma_{i'_{\delta'}}\}$, with $\ell = \delta + \delta' \le d$. We remark that necessarily $\sum_{p \in Q} p$ does not functionally depend on any random value. Actually, we even have the following lemma:

Lemma 8.1. *Let $\mathcal{A}(\vec{a}, \vec{b}; \vec{r})$ be a compression algorithm for multiplication. Then $\mathcal{A}$ is d-private if and only if there does not exist a set of non-deterministic probes $Q = \{\gamma_{i_1}, \dots, \gamma_{i_\delta}\}$ with $\delta \le d$ such that $\sum_{p \in Q} p = \vec{a}^\intercal \cdot \boldsymbol{M} \cdot \vec{b}$ where the column space or the row space of $\boldsymbol{M}$ contains a vector of Hamming weight at least $\delta + 1$.*

Furthermore, if such a set Q exists, there exists a set $\{\gamma_{i'_1}, \dots, \gamma_{i'_{\delta'}}\}$, with $\delta + \delta' \le d$, such that $P = Q \cup Q'$ is an attack.

Moreover, the lemma is still true when we restrict ourselves to sets Q such that there exists no proper subset $\hat{Q} \subsetneq Q$ such that $\sum_{p \in \hat{Q}} p$ does not functionally depend on any random.

Proof. The two first paragraphs of the lemma can be proven similarly to Lemma 5.1. Thus, we only need to prove its last part.

By contradiction, let us suppose that there exists a set Q of non-deterministic probes $Q = \{\gamma_{i_1}, \dots, \gamma_{i_\delta}\}$ such that $\sum_{p \in Q} p = \vec{a}^\intercal \cdot \boldsymbol{M} \cdot \vec{b}$ and the column space (without loss of generality, by symmetry of the a_i's and b_i's) of $\boldsymbol{M}$ contains a vector of Hamming weight at least $\delta + 1$, but such that any subset $\hat{Q} \subsetneq Q$ where $\sum_{p \in \hat{Q}} p$ that does not functionally depend on any random. Consequently, the sum $\sum_{p \in \hat{Q}} p = \vec{a}^\intercal \cdot \hat{\boldsymbol{M}} \cdot \vec{b}$, is such that the column space (still without loss of generality) of $\hat{\boldsymbol{M}}$ does not contain any vector of Hamming weight at least $|\hat{Q}| + 1$.

First, let us set $\bar{\boldsymbol{M}} = \hat{\boldsymbol{M}} + \boldsymbol{M}$ (over $\mathbb{F}_2$), so $\sum_{p \in Q \setminus \hat{Q}} p = \vec{a}^\intercal \cdot \bar{\boldsymbol{M}} \cdot \vec{b}$, as $\sum_{p \in \hat{Q}} p + \sum_{p \in Q \setminus \hat{Q}} = \sum_{p \in Q} p = \vec{a}^\intercal \cdot \boldsymbol{M} \cdot \vec{b}$ and let $\hat{\delta} = |\hat{Q}|$ and $\bar{\delta} = |Q \setminus \hat{Q}| = \delta - \hat{\delta}$. Let also ω, $\hat{\omega}$, and $\bar{\omega}$ be the maximum Hamming weights of the vectors in the column space of $\boldsymbol{M}$, $\hat{\boldsymbol{M}}$, and $\bar{\boldsymbol{M}}$, respectively. Since $\boldsymbol{M} = \hat{\boldsymbol{M}} + \bar{\boldsymbol{M}}$, then $\omega \le \hat{\omega} + \bar{\omega}$ and since $\omega > \delta + 1$, and $\delta = \hat{\delta} + \bar{\delta}$, then $\hat{\omega} > \hat{\delta}$ or $\bar{\omega} > \bar{\delta}$. We set $\tilde{Q} = \hat{Q}$ if $\hat{\omega} > \hat{\delta}$, and $\tilde{Q} = Q \setminus \hat{Q}$ otherwise. According to the definitions of $\hat{\delta}$ and $\bar{\omega}$, we have that $\tilde{Q} \subsetneq Q$ is such that $\sum_{p \in Q} p = \vec{a}^\intercal \cdot \tilde{\boldsymbol{M}} \cdot \vec{b}$ where the column space of $\tilde{\boldsymbol{M}}$ contains a vector of Hamming weight at least $|\tilde{Q}| + 1$. This contradicts the definition of Q and concludes the proof of the lemma. ☐

To quickly enumerate all the possible attacks, we first enumerate the sets $Q = \{\gamma_{i_1}, \dots, \gamma_{i_\delta}\}$ of size $\delta \le d$ such that $\sum_{p \in Q} p$ does not functionally depend on any random bit (and no proper subset of $\hat{Q} \subsetneq Q$ is such that $\sum_{p \in \hat{Q}} p$ does not functionally depend on any random bit), using *information set decoding*, recalled in the next section. Then, for each possible set Q, we check if the column space or the row space of $\boldsymbol{M}$ (as defined in the previous lemma) contains a vector of Hamming weight at least $\delta + 1$. A naive approach would consist in enumerating all the vectors in the row space and the column space of $\boldsymbol{M}$. Our tool however uses the two following facts to perform this test very quickly in most cases:

- when $\boldsymbol{M}$ contains at most δ non-zero rows and at most δ non-zero columns, Q does not yield an attack;
- when $\boldsymbol{M}$ contains exactly $\delta + 1$ non-zero rows (resp. columns), that we assume to be the first $\delta + 1$ (without loss of generality), Q yields an attack if and only if the vector $(\vec{u}_{\delta+1}^\intercal, \vec{0}_{d-\delta}^\intercal)$ is in the row space (resp. $(\vec{u}_{\delta+1}, \vec{0}_{d-\delta})$ is in the column space) of $\boldsymbol{M}$ (this condition can be checked in polynomial time in d).

8.2 Information Set Decoding and Error Probability

We now explain how to perform the enumeration step of our algorithm using information set decoding. Information set decoding was introduced in the original security analysis of the McEliece cryptosystem in [Pra62; McE78] as a way to break the McEliece cryptosystem by finding small code words in a random linear code. It was further explored by Lee and Brickell in [LB88]. We should point out that since then, many improvements were proposed, e.g., in [Leo88; Ste88]. However, for the sake of simplicity and because it already gives very good results, we use the original information set decoding algorithm. Furthermore, it is not clear that the aforementioned improvements also apply in our case, as the codes we consider are far from the Singleton bound.

We assume that random bits are denoted $r_1, \dots, r_R$. For each intermediate γ_k containing some random bit, we associate the vector $\vec{\tau} \in \mathbb{Z}_2^R$, where $\tau_i = 1$ if and only if γ_k functionally depends on the random bit r_i. We then consider the matrix $\Gamma \in \mathbb{Z}_2^{R \times \nu}$ whose k-th column is $\vec{\tau}$. For instance, for Algorithm 4, we have:

$$\Gamma = \begin{array}{c} \\ \\ \end{array} \begin{array}{cccccccccccc} \gamma_1 & \gamma_2 & \gamma_3 & \gamma_4 & \gamma_5 & \gamma_6 & \gamma_7 & \gamma_8 & \gamma_9 & \gamma_{10} & \gamma_{11} & \gamma_{12} \end{array}$$
$$\Gamma = \left(\begin{array}{cccccccccccc} 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 1 & 1 & 1 \end{array} \right) \begin{array}{c} r_0 \\ r_1 \end{array} .$$

For every $\delta \leq d$, enumerating the sets $Q = \{\gamma_{i_1}, \dots, \gamma_{i_\delta}\}$, such that $\sum_{p \in Q} p$ does not functionally depend on any random, consists in enumerating the vectors $\vec{x}$ of Hamming weight δ such that $\Gamma \cdot \vec{x} = \vec{0}$ (specifically, $\{i_1, \dots, i_\delta\}$ are the coordinates of the non-zero components of $\vec{x}$). Furthermore, we can restrict ourselves to vector $\vec{x}$ such that no vector $\vec{\hat{x}} < \vec{x}$ satisfies $\Gamma \cdot \vec{\hat{x}} = \vec{0}$ (where $\vec{\hat{x}} < \vec{x}$ means that $\vec{\hat{x}} \neq \vec{x}$ and for any $1 \leq i \leq \nu$, if $x_i = 0$ then $\hat{x}_i = 0$), since we can restrict ourselves to sets Q such that no proper subset $\hat{Q} \subsetneq Q$ is such that $\sum_{p \in \hat{Q}} p$ does not functionally depend on any random bit. This is close to the problem of finding code words $\vec{x}$ of small Hamming weight for the linear code of parity matrix Γ and we show this can be solved using information set decoding.

The basic idea is the following one. We first apply a row-reduction to Γ. Let us call the resulting matrix Γ'. We remark that, for any vector $\vec{x}$, $\Gamma \cdot \vec{x} = 0$ if and only if $\Gamma' \cdot \vec{x} = 0$ and thus we can use Γ' instead of Γ in our problem . We assume in a first time that the first R columns of Γ are linearly independent (recall that the number ν of columns of Γ is much larger than its number R of rows), so that the R first columns of Γ' forms an identity matrix. Then, for any $k^* > R$, if the k^*-th column of Γ' has Hamming weight at most $d-1$, we can consider the vector $\vec{x}$ defined as $x_{k^*} = 1$, $x_k = 1$ when $\Gamma'_{k,k^*} = 1$, and $x_k = 0$ otherwise; and this vector satisfies the conditions we were looking for: its Hamming weight is at most d and $\Gamma' \cdot \vec{x} = 0$. That way, we have quickly enumerated all the vectors $\vec{x}$ of Hamming weight at most d such that $\Gamma' \cdot \vec{x} = 0$ and with the additional property that $x_k = 0$ for all $k > R$ except for at most[2] one index k^*. Without the condition $\Gamma' \cdot \vec{x} = 0$, there are $(\nu - R + 1) \cdot \sum_{i=0}^{d-1} \binom{R}{i} + \binom{R}{d}$ such vectors, as there are $\sum_{i=0}^{d} \binom{R}{i}$ vectors $\vec{x}$ such that $\mathrm{HW}(\vec{x}) \leq d$ and $x_k = 0$ for every $k > R$, and there are $(\nu - R) \cdot \sum_{i=0}^{d-1} \binom{R}{i}$ vectors $\vec{x}$ such that $\mathrm{HW}(\vec{x}) \leq d$ and $x_k = 1$, for a single $k > R$. In other words, using row-reduction, we have been able to check $(\nu - R + 1) \cdot \sum_{i=0}^{d-1} \binom{R}{i} + \binom{R}{d}$ possible vectors $\vec{x}$ among at most $\sum_{i=1}^{d} \binom{\nu}{i}$ vectors which could be used to mount an attack, by testing at most $\nu - R$ vectors.[3]

Then, we can randomly permute the columns of Γ and repeat this algorithm. Each iteration would find an attack (if there was one attack) with probability at least $\left((\nu - R + 1) \cdot \sum_{i=0}^{d-1} \binom{R}{i} + \binom{R}{d}\right) / \sum_{i=1}^{d} \binom{\nu}{i}$. Therefore, after K iterations, the error probability is only

$$\varepsilon \leq \left(1 - \frac{(\nu - R + 1) \cdot \sum_{i=0}^{d-1} \binom{R}{i} + \binom{R}{d}}{\sum_{i=1}^{d} \binom{\nu}{i}} \right)^K ,$$

and the required number of iterations is linear with $\log(1/\varepsilon)$, which is what we wanted.

Now, we just need to handle the case when the first R columns of Γ are not linearly independent, for some permuted matrix Γ at some iteration. We can simply redraw the permutation or taking the pivots in the row-reduction instead of taking the first R columns of Γ. In both cases, this may slightly bias the probability. We make the *heuristic assumption* that the bias is negligible. To support this heuristic assumption, we remark that if we iterate the algorithm for all the permutations for which the first R columns of Γ are not linearly independent, then we would enumerate all the vectors $\vec{x}$ we are interested in, thanks to the additional condition that there is no vector $\vec{\hat{x}} < \vec{x}$ such that $\Gamma \cdot \vec{\hat{x}} = \vec{0}$.

[2]We have seen that for one index k^*, but it is easy to see that, as the first R columns of Γ' form an identity matrix, there does not exist such vector $\vec{x}$ so that $x_k = 0$ for all $k > R$ anyway.

[3]There are exactly $\sum_{i=1}^{d} \binom{\nu}{i}$ vectors of Hamming weight at most d, but here we recall that we only consider vectors $\vec{x}$ satisfying the following additional condition: there is no vector $\vec{\hat{x}} < \vec{x}$ such that $\Gamma \cdot \vec{\hat{x}} = \vec{0}$. We also remark that the vectors $\vec{x}$ generated by the described algorithm all satisfy this additional condition.

Table 3: Complexities of exhibiting an attack at several orders

		Time to find an attack	
Order	Target Algorithm	Verifier [BBD+15b]	New tool
$d = 2$	tweaked Algorithm 4	less than 1 ms	less than 10 ms
$d = 3$	tweaked Algorithm 5	36 ms	less than 10 ms
$d = 4$	tweaked Algorithm 6	108 ms	less than 10 ms
$d = 5$	tweaked Algorithm 3	6.264 s	less than 100 ms
$d = 6$	tweaked Algorithm 3	26 min	less than 300 ms

8.3 The Tool

The tool takes as input a description of a compression algorithm for multiplication similar to the ones we used in this paper (see Figure 2 for instance) and the maximum error probability ε we allow, and tries to find an attack. If no attack is found, then the scheme is secure with probability $1 - \varepsilon$. The tool can also output a description of the scheme which can be fed off into the tool in [BBD+15b].

The source code of the tool and its documentation are provided in [Tool].

8.4 Complexity Comparison

we try to give some values for the verification time of both tools when we intentionally modify our constructions to yield an attack. From order 2 to 4, we start with our optimal constructions and we just invert two random bits in an output share c_i. Similarly, for orders 5 and 6, we use our generic construction and apply the same small modification. The computations were performed on a Intel(R) Core(TM) i5-2467M CPU @ 1.60GHz and the results are given in Table 3. We can see that in all the considered cases, our new tool reveals the attack in less than 300 ms while the generic verifier of Barthe et al. needs up to 26 minutes for order $d = 6$.

Acknowledgements.

The authors thank the anonymous reviewers for their constructive comments. This work was supported in part by the French ANR Project ANR-12-JS02-0004 ROMAnTIC, the *Direction Générale de l'Armement* (DGA), the CFM Foundation.

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. http://eprint.iacr.org/2015/506. 2015 (cit. on pp. 5, 18, 19).

[BBD+15b] Gilles Barthe, Sonia Belaïd, François Dupressoir, Pierre-Alain Fouque, Benjamin Grégoire, and Pierre-Yves Strub. "Verified Proofs of Higher-Order Masking". In: *EUROCRYPT 2015, Part I*. Ed. by Elisabeth Oswald and Marc Fischlin. Vol. 9056. LNCS. Springer, Heidelberg, Apr. 2015, pp. 457–485. DOI: 10.1007/978-3-662-46800-5_18 (cit. on pp. 5, 6, 16, 18, 20, 23).

[BGN+14a] Begül Bilgin, Benedikt Gierlichs, Svetla Nikova, Ventzislav Nikov, and Vincent Rijmen. "A More Efficient AES Threshold Implementation". In: *AFRICACRYPT 14*. Ed. by David Pointcheval and Damien Vergnaud. Vol. 8469. LNCS. Springer, Heidelberg, May 2014, pp. 267–284. DOI: 10.1007/978-3-319-06734-6_17 (cit. on p. 3).

[BGN+14b] Begül Bilgin, Benedikt Gierlichs, Svetla Nikova, Ventzislav Nikov, and Vincent Rijmen. "Higher-Order Threshold Implementations". In: *ASIACRYPT 2014, Part II*. Ed. by Palash Sarkar and Tetsu Iwata. Vol. 8874. LNCS. Springer, Heidelberg, Dec. 2014, pp. 326–343. DOI: 10.1007/978-3-662-45608-8_18 (cit. on p. 3).

[BK12] Elaine B. Barker and John M. Kelsey. *SP 800-90A. Recommendation for Random Number Generation Using Deterministic Random Bit Generators*. Tech. rep. Gaithersburg, MD, United States, 2012 (cit. on p. 4).

[BOGKW88] Michael Ben-Or, Shafi Goldwasser, Joe Kilian, and Avi Wigderson. "Multi-Prover Interactive Proofs: How to Remove Intractability Assumptions". In: *20th ACM STOC*. ACM Press, May 1988, pp. 113–131 (cit. on p. 3).

[CJRR99] Suresh Chari, Charanjit S. Jutla, Josyula R. Rao, and Pankaj Rohatgi. "Towards Sound Approaches to Counteract Power-Analysis Attacks". In: *CRYPTO'99*. Ed. by Michael J. Wiener. Vol. 1666. LNCS. Springer, Heidelberg, Aug. 1999, pp. 398–412 (cit. on p. 3).

[CPRR14] Jean-Sébastien Coron, Emmanuel Prouff, Matthieu Rivain, and Thomas Roche. "Higher-Order Side Channel Security and Mask Refreshing". In: *FSE 2013*. Ed. by Shiho Moriai. Vol. 8424. LNCS. Springer, Heidelberg, Mar. 2014, pp. 410–424. DOI: 10.1007/978-3-662-43933-3_21 (cit. on pp. 3, 29).

[DDF14] Alexandre Duc, Stefan Dziembowski, and Sebastian Faust. "Unifying Leakage Models: From Probing Attacks to Noisy Leakage". In: *EUROCRYPT 2014*. Ed. by Phong Q. Nguyen and Elisabeth Oswald. Vol. 8441. LNCS. Springer, Heidelberg, May 2014, pp. 423–440. DOI: 10.1007/978-3-642-55220-5_24 (cit. on pp. 3, 4).

[DFS15a] Alexandre Duc, Sebastian Faust, and François-Xavier Standaert. "Making Masking Security Proofs Concrete - Or How to Evaluate the Security of Any Leaking Device". In: *EUROCRYPT 2015, Part I*. Ed. by Elisabeth Oswald and Marc Fischlin. Vol. 9056. LNCS. Springer, Heidelberg, Apr. 2015, pp. 401–429. DOI: 10.1007/978-3-662-46800-5_16 (cit. on p. 3).

[DFS15b] Stefan Dziembowski, Sebastian Faust, and Maciej Skorski. "Noisy Leakage Revisited". In: *EUROCRYPT 2015, Part II*. Ed. by Elisabeth Oswald and Marc Fischlin. Vol. 9057. LNCS. Springer, Heidelberg, Apr. 2015, pp. 159–188. DOI: 10.1007/978-3-662-46803-6_6 (cit. on p. 4).

[GP99] Louis Goubin and Jacques Patarin. "DES and Differential Power Analysis (The "Duplication" Method)". In: *CHES'99*. Ed. by Çetin Kaya Koç and Christof Paar. Vol. 1717. LNCS. Springer, Heidelberg, Aug. 1999, pp. 158–172 (cit. on p. 3).

[IKL+13] Yuval Ishai, Eyal Kushilevitz, Xin Li, Rafail Ostrovsky, Manoj Prabhakaran, Amit Sahai, and David Zuckerman. "Robust Pseudorandom Generators". In: *ICALP 2013, Part I*. Ed. by Fedor V. Fomin, Rusins Freivalds, Marta Z. Kwiatkowska, and David Peleg. Vol. 7965. LNCS. Springer, Heidelberg, July 2013, pp. 576–588. DOI: 10.1007/978-3-642-39206-1_49 (cit. on p. 6).

[ISW03] Yuval Ishai, Amit Sahai, and David Wagner. "Private Circuits: Securing Hardware against Probing Attacks". In: *CRYPTO 2003*. Ed. by Dan Boneh. Vol. 2729. LNCS. Springer, Heidelberg, Aug. 2003, pp. 463–481 (cit. on pp. 3–7, 15).

[Koc96] Paul C. Kocher. "Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS, and Other Systems". In: *CRYPTO'96*. Ed. by Neal Koblitz. Vol. 1109. LNCS. Springer, Heidelberg, Aug. 1996, pp. 104–113 (cit. on p. 3).

[LB88] Pil Joong Lee and Ernest F. Brickell. "An Observation on the Security of McEliece's Public-Key Cryptosystem". In: *EUROCRYPT'88*. Ed. by C. G. Günther. Vol. 330. LNCS. Springer, Heidelberg, May 1988, pp. 275–280 (cit. on p. 22).

[Leo88] Jeffrey S. Leon. "A probabilistic algorithm for computing minimum weights of large error-correcting codes". In: *IEEE Transactions on Information Theory* 34.5 (1988), pp. 1354–1359 (cit. on p. 22).

[LSP82] Leslie Lamport, Robert E. Shostak, and Marshall C. Pease. "The Byzantine Generals Problem". In: *ACM Trans. Program. Lang. Syst.* 4.3 (1982), pp. 382–401 (cit. on p. 3).

[McE78] Robert J McEliece. "A public-key cryptosystem based on algebraic coding theory". In: *DSN progress report* 42.44 (1978), pp. 114–116 (cit. on pp. 5, 22).

[NRS11] Svetla Nikova, Vincent Rijmen, and Martin Schläffer. "Secure Hardware Implementation of Nonlinear Functions in the Presence of Glitches". In: *Journal of Cryptology* 24.2 (Apr. 2011), pp. 292–321 (cit. on p. 3).

[PR13] Emmanuel Prouff and Matthieu Rivain. "Masking against Side-Channel Attacks: A Formal Security Proof". In: *EUROCRYPT 2013*. Ed. by Thomas Johansson and Phong Q. Nguyen. Vol. 7881. LNCS. Springer, Heidelberg, May 2013, pp. 142–159. DOI: 10.1007/978-3-642-38348-9_9 (cit. on p. 3).

[Pra62] E. Prange. “The use of information sets in decoding cyclic codes”. In: *Information Theory, IRE Transactions on* 8.5 (Sept. 1962), pp. 5–9. ISSN: 0096-1000. DOI: 10.1109/TIT.1962.1057777 (cit. on pp. 5, 22).

[RBN+15] Oscar Reparaz, Begül Bilgin, Svetla Nikova, Benedikt Gierlichs, and Ingrid Verbauwhede. “Consolidating Masking Schemes”. In: *CRYPTO 2015, Part I*. Ed. by Rosario Gennaro and Matthew J. B. Robshaw. Vol. 9215. LNCS. Springer, Heidelberg, Aug. 2015, pp. 764–783. DOI: 10.1007/978-3-662-47989-6_37 (cit. on p. 4).

[RP10] Matthieu Rivain and Emmanuel Prouff. “Provably Secure Higher-Order Masking of AES”. In: *CHES 2010*. Ed. by Stefan Mangard and François-Xavier Standaert. Vol. 6225. LNCS. Springer, Heidelberg, Aug. 2010, pp. 413–427 (cit. on pp. 3–5, 29).

[Sage] The Sage Developers. *Sage Mathematics Software (Version 6.8)*. http://www.sagemath.org. 2015. DOI: 10.5281/zenodo.28513 (cit. on pp. 5, 20).

[Ste88] Jacques Stern. “A method for finding codewords of small weight”. In: *Coding Theory and Applications, 3rd International Colloquium, Toulon, France, November 2-4, 1988, Proceedings*. Ed. by Gérard D. Cohen and Jacques Wolfmann. Vol. 388. Lecture Notes in Computer Science. Springer, 1988, pp. 106–113 (cit. on p. 22).

[Tool] https://github.com/fabrice102/private_multiplication (cit. on pp. 5, 23).

[Yao82] Andrew Chi-Chih Yao. “Protocols for Secure Computations (Extended Abstract)”. In: *23rd FOCS*. IEEE Computer Society Press, Nov. 1982, pp. 160–164 (cit. on p. 3).

A Proof of the Relation between Compression Algorithm and Multiplication Algorithm (Proposition 2.6)

As explained in Section 2.2, to prove Proposition 2.6, we just need to prove that we can transform an attack set $P = \{p_1, \dots, p_\ell\}$ of size $\ell \le d$ for the multiplication algorithm $\mathcal{B}$ into an attack set P' for the compression algorithm $\mathcal{A}$. The only difference is that some probes in P may be the inputs a_i or b_i, while such probes are forbidden in P'.

This is actually surprisingly very hard. We first need to review Section 3 to add the possibility to have probes for the inputs a_i and b_j. The following subsections (except the last one) follows the subsections of Section 3. We highly recommend the reader to understand completely Section 3 before reading the sequel.

A.1 Extended Matrix Notation

We now write probes p as:

$$p = \vec{a}^\intercal \cdot \boldsymbol{M_p} \cdot \vec{b} + \vec{a}^\intercal \cdot \vec{m_{p,a}} + \vec{m_{p,b}}^\intercal \cdot \vec{b} + \vec{s_p}^\intercal \cdot \vec{r} + c_p,$$

where $\boldsymbol{M_p}$ is a matrix in $\mathbb{F}_2^{(d+1)\times(d+1)}$, $\vec{m_{p,a}}$ and $\vec{m_{m,b}}$ are two vectors in $\mathbb{F}_2^{d+1}$, and $c_p \in \mathbb{F}_2$ is a constant which is supposed to be zero in the sequel (as in Section 3). This notation can be extended to the sum of probes.

Notice that actually, for all the probes we consider: at most one of the matrices or vectors $\boldsymbol{M_p}$, $\vec{m_{p,a}}$, and $\vec{m_{p,b}}$ is non-zero. Furthermore the Hamming weight of $\vec{m_{p,a}}$, and $\vec{m_{p,b}}$ is at most 1. However, it is easier to deal with this slight generalization.

A.2 Extended Algebraic Condition

We now introduce our extended algebraic condition:

Condition 3. *A set of probes $P = \{p_1, \dots, p_\ell\}$ on a multiplication algorithm satisfies Condition 1 if and only if the expression $f = \sum_{i=1}^{\ell} p_i$ can be written as $f = \vec{a}^\intercal \cdot \boldsymbol{M} \cdot \vec{b} + \vec{a}^\intercal \cdot \vec{m_a} + \vec{m_b}^\intercal \cdot \vec{b}$ with $\boldsymbol{M}$ being some matrix and $\vec{m_a}$ and $\vec{m_b}$ being some vectors such that $\vec{u}_{d+1}$ is in the affine space $\vec{m_a} + \operatorname{im}(\boldsymbol{M})$ or $\vec{m_b} + \operatorname{im}(\boldsymbol{M}^\intercal)$, where $\operatorname{im}(\boldsymbol{M})$ is the column space of $\boldsymbol{M}$ and $\operatorname{im}(\boldsymbol{M}^\intercal)$ is the row space of $\boldsymbol{M}$.*

A.3 Extended Algebraic Characterization

Theorem A.1. *Let $\mathcal{B}$ be a multiplication algorithm constructed from a d-compression algorithm for multiplication as in Remark 2.5. Then, $\mathcal{B}$ is d-private if and only if there does not exist a set $P = \{p_1, \dots, p_\ell\}$ of $\ell \le d$ probes that satisfies Condition 3. Furthermore any set $P = \{p_1, \dots, p_\ell\}$ satisfying Condition 1 is an attack.*

Proof. The left-to-right direction can be proven similarly as for Theorem 3.1. Let us focus on the right-to-left direction.

The proof is exactly the same up to the definition of the p'_i (since what comes before that only considers the random bits in the probes and hence is similar when probes of the form a_i or b_i are taken into account), which can now be written as:

$$p'_i = \vec{a}^\intercal \cdot \boldsymbol{M'_i} \cdot \vec{b} + \vec{a}^\intercal \cdot \vec{m'_{i,a}} + \vec{m'_{i,b}}^\intercal \cdot \vec{b},$$

for some matrix $\boldsymbol{M'_i}$ and vectors $\vec{m'_{i,a}}$ and $\vec{m'_{i,b}}$.

We now can conclude using the following lemma, which is an extended version of Lemma 3.4 and whose proof is similar:

Lemma A.2. *If $(p'_1, \dots, p'_t)$ is an attack tuple, then there exists a vector $\vec{b^*} \in \mathbb{F}_2^{d+1}$ such that $\vec{u}_{d+1}$ is in the vector space $\langle \boldsymbol{M'_1} \cdot \vec{b^*} + \vec{m'_{1,a}}, \dots, \boldsymbol{M'_t} \cdot \vec{b^*} + \vec{m'_{t,a}} \rangle$.*

Thanks to Lemma A.2, there exists a vector $\vec{\sigma} = (\sigma_1, \dots, \sigma_t)^\intercal \in \mathbb{F}_2^t$ and a vector $\vec{b^*} \in \mathbb{F}_2^{d+1}$, such that

$$\left(\sum_{i=1}^{t} \sigma_i \cdot \boldsymbol{M'_i} \right) \cdot \vec{b^*} + \left(\sum_{i=1}^{t} \sigma_i \cdot \vec{m'_{i,b}} \right) = \vec{u}_{d+1} \quad . \tag{3}$$

Let $\vec{\sigma}'$ be the vector in $\mathbb{F}_2^{\ell}$ defined by $\vec{\sigma}'^{\intercal} = \begin{pmatrix} \vec{\sigma}^{\intercal} & \vec{0}_{\ell-t}^{\intercal} \end{pmatrix} \cdot \boldsymbol{N}$. We have:

$$\vec{\sigma}'^{\intercal} \cdot \vec{p} = \sum_{i=1}^{t} \sigma_i \cdot p'_i = \sum_{i=1}^{t} \sigma_i \cdot \vec{a}^{\intercal} \cdot \boldsymbol{M'_i} \cdot \vec{b} + \sum_{i=1}^{t} \sigma_i \cdot \vec{a}^{\intercal} \cdot \vec{m}'_{i,a} + \sum_{i=1}^{t} \sigma_i \cdot \vec{m}'^{\intercal}_{i,b} \cdot \vec{b}$$

so that:

$$\vec{\sigma}'^{\intercal} \cdot \vec{p} = \vec{a}^{\intercal} \cdot \left(\sum_{i=1}^{t} \sigma_i \cdot \boldsymbol{M'_i} \right) \cdot \vec{b} + \vec{a}^{\intercal} \cdot \left(\sum_{i=1}^{t} \sigma_i \cdot \vec{m}'_{i,a} \right) + \left(\sum_{i=1}^{t} \sigma_i \cdot \vec{m}'^{\intercal}_{i,b} \right) \cdot \vec{b} \ . \tag{4}$$

Therefore, we can define the set $P' = \{p_i \mid \sigma_i = 1\}$. This set satisfies Condition 3, according to Equations (3) and (4).

This concludes the proof.

□

A.4 Proof of Proposition 2.6

We can now prove Proposition 2.6.

Proposition 2.6. Let us suppose by contraposition that $\mathcal{B}$ is not d-private. Thanks to Theorem A.1, this means that there exists a set of probes $P = \{p_1, \dots, p_\ell\}$ satisfying Condition 3. We suppose without loss of generality that:

$$\sum_{i=1}^{\ell} p_i = \vec{a}^{\intercal} \cdot \boldsymbol{M} \cdot \vec{b} + \vec{a}^{\intercal} \cdot \vec{m}_a + \vec{m}_b^{\intercal} \cdot \vec{b}$$

and $\vec{u}_{d+1}$ is in the affine space $\vec{m}_a + \mathrm{im}(\boldsymbol{M})$.

From the shapes of possibles probes, we know that for any i, if $m_{a,i} = 1$, then one of the probe p_j has to be a_i. Let us spit our set of probes P in three sets:

- P_1 contains all the probes p_j such that $p_j = a_i$ and $m_{a,i} = 1$;
- P_2 contains all the probes which are not of the form a_i or b_j;
- P_3 contains all the other probes.

We remark that:

$$\sum_{p \in P_1 \cup P_2} p = \vec{a}^{\intercal} \cdot \boldsymbol{M} \cdot \vec{b} + \vec{a}^{\intercal} \cdot \vec{m}_a.$$

Let $\vec{b^*} \in \mathbb{F}_2^{d+1}$ be a vector such that $\vec{u}_{d+1} = \vec{m}_a + \boldsymbol{B} \cdot \vec{b^*}$. The Hamming weight of $\vec{m}_a$ is exactly the cardinality of P_1 and is at most d. Therefore, $\vec{b^*} \neq \vec{0}_{d+1}$ and we can arbitrarily choose $0 \leq j^* \leq d$ such that $b^*_{j^*} = 1$. Finally, let us set $P'_1 = \{\alpha_{i,j^*} \mid m_{a,i} = 1\}$. We can write

$$\sum_{p \in P'_1 \cup P_2} p = \vec{a}^{\intercal} \cdot \boldsymbol{M'} \cdot \vec{b},$$

and we have that:

$$\boldsymbol{M'} \cdot \vec{b^*} = \boldsymbol{M} \cdot \vec{b^*} + \vec{m}_a = \vec{u}_{d+1}.$$

Therefore $P'_1 \cup P_2$ is a set of probes (for the compression algorithm $\mathcal{A}$) which satisfies Condition 1.

This concludes the proof. □

B Counterexample for a Variant of Theorem 3.1 with Condition 2

In this appendix, we show that Theorem 3.1 would not be valid with Condition 2 (instead of Condition 1). Let us indeed consider the following 2-compression scheme for multiplication:

$$\begin{array}{ccccc}
\alpha_{2,0} & r_1 & \alpha_{0,0} & r_2 & \alpha_{0,2} \\
\alpha_{2,1} & r_1 & \alpha_{1,1} & r_3 & \alpha_{1,2} \\
\alpha_{1,0} & r_2 & \alpha_{0,1} & r_3 & \alpha_{2,2}
\end{array}$$

It is easy to see that the only set of probes satisfying Condition 2 is

$$P = \{\alpha_{2,0} + r_1 + \alpha_{0,0}, \quad \alpha_{2,1} + r_1 + \alpha_{1,1}\} \ .$$

However, this set does not satisfy Condition 1, and therefore this compression algorithm is 2-private (since any set satisfying Condition 1 also satisfies Condition 2).

C Proofs of Section 4

C.1 Proof of the Better Linear Lower Bound (Section 4.3)

Theorem 4.4. Let $r_1, \dots, r_d$ denote the random bits used by $\mathcal{A}$. Let $c_0, \dots, c_d$ denote the outputs of $\mathcal{A}$. The proof is organized through 4 steps as follows:

1. we prove that at least one c_i (further referred to as c_0) functionally depends on at least two distinct random bits,
2. we prove that at least two shares c_0 and c_j (further referred to as c_1) both functionally depend on at least two distinct random bits,
3. we prove that at least one random bit r_1 is such that no additive share functionally depends on only r_1,
4. we exhibit an attack.

Step 1: c_0 functionally depends on at least two distinct random bits.

Let us first show that at least one of the $(c_i)_{0 \le i \le d}$ functionally depends on two different random bits. By contradiction, let us assume that every $(c_i)_{0 \le i \le d}$ functionally depends on at most 1 random bit. Then, as there are d random bits, it implies that, either one c_i does not functionally depend on any random bit, or there exist $i < j$ such that c_i and c_j functionally depend on the same random bit:

- In the first case, let us assume that c_0 does not functionally depend on any random bit. Then, by correctness, neither does $\sum_{i=1}^{d} c_i$. Then, $S_0 = \{c_0\}$ and $S_1 = \{c_1, \dots, c_d\}$ satisfy the requirements of Lemma 4.2 and $\mathcal{A}$ is not d-private.
- In the second case, let us assume without loss of generality that c_0 and c_1 functionally depend on the same random bit (and only on this one by assumption). Then, $c_0 + c_1$ does not depend on any random bit and so does $\sum_{i=2}^{d} c_i$ via correctness. Then, $S_0 = \{c_0, c_1\}$ and $S_1 = \{c_2, \dots, c_d\}$ satisfy the requirements of Lemma 4.2 and $\mathcal{A}$ is not d-private.

Then, we can now assume without loss of generality that c_0 functionally depends on at least two distinct random bits.

Step 2: both c_0 and c_1 functionally depend on at least two distinct random bits.

By contradiction, let us now assume that for all $1 \le i \le d$, c_i functionally depends on only one random bit. In order to achieve correctness, there must exist c_i, c_j with $1 \le i < j \le d$ that respectively depend on the first and second of the two distinct random bits on which c_0 functionally depends on. As we assume that there are d random bits in total, all the d random bits on which the c_i's functionally depend on for $i \ge 1$ have to be different. Hence, we can assume without loss of generality that c_i functionally depends on r_i, for $1 \le i \le d$. Thus, by correctness, c_0 functionally depends on all r_i for $i = 1, \dots, d$.

Then, we can simply probe the first subsum p of c_0 that functionally depends on at least 2 distinct random bits (actually, any subsum that functionally depends on at least 2 and at most $d-1$ random bits would work).

Let us denote by $I = \{i_1, i_2\}$ the set of indices corresponding to the random bits on which p functionally depends on. Then, $S_0 = \{p, c_{i_1}, c_{i_2}\}$ and $S_1 = \{c_0, p\} \cup \{c_i \mid i \in \{1, \dots, d\} \backslash \{i_1, i_2\}\}$ satisfy the requirements of Lemma 4.2 and $\mathcal{A}$ is not d-private.

Therefore, it is not possible that every c_i for $1 \le i \le d$ functionally depends on only one random bit and thus, there are at least two c_i, c_j with $i \ne j$ that functionally depend on at least 2 distinct random bits. We can assume without loss of generality that c_0 and c_1 each functionally depend on at least 2 distinct random bits.

Step 3: r_1 is such that no c_i (for $i = 0, \dots, d$) functionally depends on only r_1 (and on no other random bits).

By correctness, c_0 functionally depends on all the random bits on which $\sum_{i=1}^{d} c_i$ functionally depends on. We just proved that c_1 functionally depends on at least two distinct bits. Also, all random bits have to be used (otherwise there are at most $d-1$ bits and Theorem 4.3 already proves that $\mathcal{A}$ is not d-private). Consequently, each of the at most $d-2$ random bits on which c_1 does not functionally depend on have to satisfy that at least one of the c_i for $2 \leq i \leq d$ functionally depends on this random bit. Thus, it is not possible that all c_i for $i \geq 2$ functionally depend on only 1 random bit (otherwise, there would exist $2 \leq i_1 < i_2 \leq d$ such that c_{i_1} and c_{i_2} functionally depend only on the same random bit. Indeed, in that case, $S_0 = \{c_{i_1}, c_{i_2}\}$ and $S_1 = \{c_0\} \cup \{c_i \mid i \in \{1, \dots, d\} \backslash \{i_1, i_2\}\}$ satisfy the requirements of Lemma 4.2 and $\mathcal{A}$ is not d-private.

Thus, this proves that at least one of the random bits, say r_1 without loss of generality, is such that no c_i (for $i = 0, \dots, d$) functionally depends on only r_1 (and on no other random bits).

Step 4: there exists an attack with at most d probes.

Let us now consider, similarly to what we did in the proof of Theorem 4.3, the matrix $\boldsymbol{N} \in \mathbb{F}_2^{d \times d}$ defined as the matrix whose coefficients $n_{i,j}$ are equal to 1 if and only if c_j functionally depends on the random bit r_i, for $1 \leq i, j \leq d$. Please note once again that this matrix does not depend of c_0. In order to prevent the same kind of attack than the one we used in the proof of Theorem 4.3, it is clear that this matrix has to be invertible. Hence, there exists $\vec{w} \in \mathbb{F}_2^d$ such that $\boldsymbol{N} \cdot \vec{w} = \vec{e}_1$ where $\vec{e}_1$ denotes the first vector of the canonical basis of $\mathbb{F}_2^d$. Moreover, since c_0 and $\sum_{i=1}^{d} c_i$ both functionally depends on the same at least 2 distinct random bits, then by correctness, we have $\vec{w} \neq 1^d$. Also, the Hamming weight of $\vec{w}$ is at least 2, since r_1 never appears alone in an additive share, which implies that $\vec{e}_1$ is not a column in $\boldsymbol{N}$. Hence, we have $2 \leq \mathsf{hw}(\vec{w}) \leq d-1$.

To conclude the proof, we just note that $S_0 = \{r', c_0\} \cup \{c_i \mid w_i = 0\}$ and $S_1 = \{r'\} \cup \{c_i \mid w_i = 1\}$ satisfy the requirements of Lemma 4.2 and $\mathcal{A}$ is not d-private.

Theorem 4.4 follows. □

C.2 Proof of the Upper-Bound (Section 4.4)

Theorem 4.6. We remark that a random algorithm as defined in Algorithm 2 and Lemma 4.5 is secure with probability at least

$$\begin{aligned} 1 - \binom{(R+3) \cdot d \cdot (d+1)/2}{d} \cdot 2^{-R} &\geq 1 - ((R+3) \cdot d \cdot (d+1)/2)^d \cdot 2^{-R} \\ &= 1 - 2^{d \cdot \log((R+3) \cdot d \cdot (d+1)/2) - R}. \end{aligned}$$

When this probability is greater than 0, then there exists necessarily a choice of $\rho(i,j)$ leading to a secure algorithm. This condition can be rewritten as:

$$d \cdot \log((R+3) \cdot d \cdot (d+1)/2) - R < 0.$$

Let us take $R = K \cdot d \cdot \log d - 3$ with K some constant to be fixed later. As $d \cdot (d+1)/2 \leq d^2$, we have

$$\begin{aligned} d \cdot \log((R+3) \cdot d \cdot (d+1)/2) - R &\leq d \cdot \log(K \cdot d^3 \cdot \log d) - K \cdot d \cdot \log d + 3 \\ &= d \cdot ((3-K) \cdot \log d + \log K + \log \log d) + 3 \end{aligned}$$

When K is large enough, this is always negative. Theorem 4.6 easily follows. □

D Proof of the New Construction (Section 5)

Proposition 5.2. Inspired from simulation-based proofs of multiplications in [RP10; CPRR14], our proof consists in constructing two sets I and J of indices in $[0, d]$ of size at most d and such that the distribution of any d-tuple $(v_1, v_2, \dots, v_d)$ of intermediate variables can be perfectly simulated from $\alpha_{I,J} = (\alpha_{i,j})_{i \in I, j \in J}$. This proves our statement as long as the cardinalities of I and J are smaller than $d+1$. We now describe the construction of I and J.

Construction of the sets I and J.

Initially, I and J are empty. We fill them in the following specific order according to the possible attacker's probes.

1. for any observed variable $\alpha_{i,i}$, add i to I and i to J.
2. for any observed variable $\alpha_{i,j}$, add i to I and j to J.
3. for any observed variable r_j, put j in I and j in J.
4. for any observed intermediate sum occurring during the computation of c_i, assign from shortest sums (in terms of number of terms) to longest sums:
   - if $i \notin I$, add i to I. Otherwise, if c_i involves corrective terms (i.e., randoms not in $t_{i,j}$), consider them successively (from left to right). For a random of the form $r_{j,i}$, if $j \notin I$, add j to I, otherwise, consider the next random. For a random of the form r_j, if $j \notin I$, add j to I. If there are no more corrective terms to consider, or if c_i does not involve corrective terms, consider the involved $t_{i,j}$ in reverse order (from right to left). Add to I the first index j that is not in I.
   - if $i \notin J$, add i in J. Otherwise, if c_i involves corrective terms (i.e., randoms not in $t_{i,j}$),consider them successively (from left to right). For a random of the form $r_{j,i}$, if $j \notin J$, add j to J, otherwise, consider the next random. For a random of the form r_j, if $j \notin J$, add j to J. If there are no more corrective terms to consider, or if c_i does not involve corrective terms, consider the involved $t_{i,j}$ in reverse order (from right to left). Add to J the first index j that is not in J.
5. for any observed variable $r_{i,j}$:
   - if $i \notin I$, add i to I, otherwise add j to I.
   - if $i \notin J$, add i to J, otherwise add j to J.
6. for any observed intermediate sum t occurring during the computation of $t_{i,j}$ we distinguish two cases:
   - t is a sum of at most 3 terms[4]:
     - if $i \notin I$, add i to I, otherwise add j to I.
     - if $i \notin J$, add i to J, otherwise add j to J.
   - t is a sum of strictly more than 3 terms:
     - if $j-1 \notin I$, add $j-1$ to I. Otherwise, if $i \notin I$, add i to I, otherwise add j to I.
     - if $j-1 \notin J$, add $j-1$ to J. Otherwise, if $i \notin J$, add i to J, otherwise add j to J.

We can check that these categories cover all possible intermediate variables in our algorithm. Moreover, each observation adds at most one index in I and one index in J. With at most d probes, their cardinals hence cannot be greater than d.

Simulation phase.

Before simulating, we make the following useful observations.

(i). all variables whose expression involves $r_{i,j}$ are: $r_{i,j}$, $t_{i,j}$, c_i, c_j.

(ii). all variables whose expression involves r_{j-1} are: r_{j-1}, $t_{k,j}$, c_{j-1}, c_k, for any $k \leq j-2$.

(iii). all variables whose expression involves both $r_{i,j}$ and r_{j-1} are: c_i and $t_{i,j}$.

We now prove that every observed value can be perfectly simulated with the input shares whose indexes are among I and J.

1. any variable $\alpha_{i,i}$ is trivially simulated thanks to the fact that i is in I and in J.

[4]Note that the case where it involves only one term $r_{i,j}$ has already been treated.

2. any variable $\alpha_{i,j}$ is trivially simulated thanks to the fact that i is in I and j is in J.

3. any variable $r_{i,j}$ is assigned to a uniformly distributed random value, as it is the case in the real algorithm.

4. any variable r_j is assigned to a uniformly distributed random value, as it is the case in the real algorithm.

5. for any variable t of at most three terms manipulated during the computation of $t_{i,j}$:

   - if t is a sum of at most 3 terms (i.e., $t = r_{i,j} + \alpha_{i,j}$ or $t = r_{i,j} + \alpha_{i,j} + \alpha_{j,i}$), then necessarily we have $i \in I$ and $i \in J$. Moreover :
     - if $j \in I$ and $j \in J$, t can be perfectly simulated with $\alpha_{i,j}$ and $\alpha_{j,i}$ thanks to the indexes in I and J.
     - otherwise, we show that t can be assigned to a random value. In particular, we show that if t is non-random, we must have $i, j \in I$ and $i, j \in J$. The variable t involves $r_{i,j}$. As noted in Observation (i), this variable can only appear either alone, in c_i, in c_j, in another t' of less than three terms part of $t_{i,j}$, or in another t' of strictly more than three terms part of $t_{i,j}$.
       * $r_{i,j}$ appears alone: this probe involved $i \in I$ and $i \in J$, and hence the probe of t added j in I and j in J
       * $r_{i,j}$ appears in an observed c_i: this probe involved $i \in I$ and $i \in J$, and hence the probe of t added j in I and j in J
       * $r_{i,j}$ appears in an observed c_j: this probe involved $j \in I$ and $j \in J$ and hence the probe of t added i in I and i in J
       * $r_{i,j}$ appears in another observed t' of less than three terms: the probe of two variables t and t' of this kind leads to first $i \in I$ and $i \in J$ and then $j \in I$ and $j \in J$
       * $r_{i,j}$ appears in another observed t' of strictly more than three terms: in this case, t' also involves the random r_{j-1}. With Observation (ii), we know that r_{j-1} can either be observed alone, in c_{j-1}, in t'' of more than three terms part of $t_{k,j}$ or in c_k. Once again, considering r_{j-1} or c_{j-1}, and t and t', we get that $j-1, j, i \in I$ and $j-1, j, i \in J$. Considering t'' of more than three terms, or c_k, if $k = i$, we have already treated this case and we have $i, j \in I$ and $i, j \in J$, otherwise, the variable involves $r_{k,j}$. Once again thanks to Observation (i), we know exactly in which variables of the protocol $r_{k,j}$ can be involved. It can be checked that $i, j, k, j-1$ are in I and in J for each variables that are not part of $c_{k'}$ or $t_{k,j}$. Consequently, each other probe that does not imply $i, j \in I$ and $i, j \in J$ are variables of these kinds. However, each of these variables involves both r_{j-1} and $r_{k',j}$ for a certain k'. To summarize, t has been queried, which involves only $r_{i,j}$, and the only other possible variables involve r_{j-1} and $r_{\ell,j}$, where ℓ is the index of the line. Hence, the parity of the number of occurrences of r_{j-1} is different from the parity of the number of occurrences of $r_{\ell,j}$. This ensures that it is impossible to get rid of r_{j-1} and all variables $r_{\ell,j}$ at the same time. Therefore, in those cases t can be assigned to a random value.
   - if t is a sum of strictly more than 3 terms:
     - if $i, j, j-1 \in I$ and $i, j, j-1 \in J$, then t can be simulated from the indexes in I and J.
     - t involves $r_{i,j}$ and r_{j-1}. Observations (i) and (ii) provide us the variables in which these randoms are involved. For all but four cases, we trivially have $i, j, j-1 \in I$ and $i, j, j-1 \in J$. Those four cases are the queries of $(r_{i,j}, t')$ with t' part of $t_{k,j}$ and involving strictly more than three terms, (c, c'), where c and c' are part of c_i, (t', t'') with t' part of $t_{i,j}$ and t'' part of $t_{k,j}$, where t'' is assigned before t', both involving more than three terms, and finally, any other couple involving a part of c_k.
       * the cases $(r_{i,j}, t')$ and (t', t'') imply the involvement of $r_{k,j}$. Thanks to Observation (i), all possible cases can be exhausted, and we obtain $i, k, j-1, j \in I$ and $i, k, j-1, j \in J$.
       * the case (c, c') is particular. Indeed, we can assume that c is computed during the computation of c'. We can hence safely assign t to a random variable if this is the only case where $r_{i,j}$ and r_{j-1} have been involved.

$*$ the query of a c, part of c_k and involving r_{j-1} involves the variable $r_{k,j}$. From Observation (i), we can exhaust the possible cases. For each of these cases except five, we have $i, j, j-1, k \in I$ and $i, j, j-1, k \in J$. The five remaining cases are $(c_j, c_j), (c_j, c_k), (r_{k,j}, c_k), (c_i, c_k), (t_{i,j}, c_k)$. With the case involving c_j, by construction we have that $r_{k,j}$ and $r_{i,j}$ appear after the addition of all the terms of the form $t_{j,\ell}$. Consequently, this expression involves the term $r_{j-1,j}$ (if $i = j-1$, $t_{i,j}$ does not exist, and if $k = j-1$, the probe of c_k assures that we have $j-1$ in I and J, hence we also get $i, j \in I$ and $i, j \in J$). Using Observation (i), we find out that the only way not to have $i, j, j-1 \in I$ and $i, j, j-1 \in I$ is to make another probe to c_j. However, this case is similar to the one we just observed: it is safe to randomly assign t. For any another case, the random $r_{k,j}$ reappears, and we must hence query another variable to get rid of it. The only possibility is to query c_k once more. Hence t can be randomly assigned.

$\square$