How to Securely Compute with Noisy Leakage in Quasilinear Complexity

Dahmun Goudarzi[1,2(✉)], Antoine Joux[3], and Matthieu Rivain[1]

[1] CryptoExperts, Paris, France
{dahmun.goudarzi,matthieu.rivain}@cryptoexperts.com
[2] ENS, CNRS, Inria and PSL Research University, Paris, France
[3] Sorbonne Université, Institut de Mathématiques de Jussieu–Paris Rive Gauche, CNRS, Inria, Univ Paris Diderot, Paris, France
antoine.joux@m4x.org

Abstract. Since their introduction in the late 90's, side-channel attacks have been considered as a major threat against cryptographic implementations. This threat has raised the need for formal leakage models in which the security of implementations can be proved. At Eurocrypt 2013, Prouff and Rivain introduced the *noisy leakage model* which has been argued to soundly capture the physical reality of power and electromagnetic leakages. In their work, they also provide the first formal security proof for a masking scheme in the noisy leakage model. However their work has two important limitations: (i) the security proof relies on the existence of a leak-free component, (ii) the tolerated amount of information in the leakage (aka *leakage rate*) is of $O(1/n)$ where n is the security parameter (*i.e.* the number of shares in the underlying masking scheme). The first limitation was nicely tackled by Duc, Dziembowski and Faust one year later (Eurocrypt 2014). Their main contribution was to show a security reduction from the noisy leakage model to the conceptually simpler *random-probing model*. They were then able to prove the security of the well-known Ishai-Sahai-Wagner scheme (Crypto 2003) in the noisy leakage model. The second limitation was addressed in a paper by Andrychowicz, Dziembowski and Faust (Eurocrypt 2016) which makes use of a construction due to Ajtai (STOC 2011) to achieve security in the strong adaptive probing model with a leakage rate of $O(1/\log n)$. The authors argue that their result can be translated into the noisy leakage model with a leakage rate of $O(1)$ by using secret sharing based on algebraic geometric codes. In terms of complexity, the protected program scales from $|P|$ arithmetic instructions to $\tilde{O}(|P|\, n^2)$. According to the authors, this $\tilde{O}(n^2)$ blow-up could be reduced to $\tilde{O}(n)$ using *packed secret sharing* but no details are provided. Moreover, such an improvement would only be possible for a program of *width* at least linear in n. The issue of designing an explicit scheme achieving $\tilde{O}(n)$ complexity blow-up for *any* arithmetic program is hence left open.

In this paper, we tackle the above issue: we show how to securely compute in the presence of noisy leakage with a leakage rate $\tilde{O}(1)$ and complexity blow-up $\tilde{O}(n)$. Namely, we introduce a transform that turns any program P composed of arithmetic instructions on some filed $\mathbb{F}$ into

T. Peyrin and S. Galbraith (Eds.): ASIACRYPT 2018, LNCS 11273, pp. 547–574, 2018.
https://doi.org/10.1007/978-3-030-03329-3_19

a (functionally equivalent) program Π composed of $|\Pi| = O(|P| n \log n)$ arithmetic instructions which can tolerate some (quasi-constant) amount of noisy leakage on its internal variables (while revealing negligible information). We use a polynomial encoding allowing quasilinear multiplication based on the fast Number Theoretic Transform (NTT). We first show that our scheme is secure in the random-probing model with leakage rate $O(1/\log n)$. Using the reduction by Duc *et al.* this result can be translated in the noisy leakage model with a $O(1/|\mathbb{F}|^2 \log n)$ leakage rate. However, a straight application of this reduction is not satisfactory since our construction requires $|\mathbb{F}| = O(n)$. In order to bypass this issue (which is shared with the construction of Andrychowicz *et al.*), we provide a generic security reduction from the noisy leakage model at the logical-instruction level to the random-probing model at the arithmetic level. This reduction allows us to prove the security of our construction in the noisy leakage model with leakage rate $\tilde{O}(1)$.

1 Introduction

Side-channel attacks have been considered as a major threat against cryptographic implementations since their apparition in the late 90's. It was indeed shown that even a tiny dependence between the data processed by a device and its *side-channel leakage* (*e.g.* running time, power consumption, electromagnetic emanation) could allow devastating key-recovery attacks against the implementation of any cryptosystem secure in the *black-box* model (*i.e.* the model in which the adversary only sees the input-output behaviour of the cryptosystem) [15,18,19]. The so-called physical security of cryptographic implementations has then become a very active research area and many efficient countermeasures have been proposed to mitigate these side-channel attacks. However, most of these countermeasures are only empirically validated or they are proven secure in a weak adversarial model where, for instance, an attacker only exploits a small part of the available leakage.

An important step towards a more formal treatment of side-channel security was made by Micali and Reyzin in 2004 in their *physically observable cryptography* framework [20]. In particular, they formalized the assumptions that a cryptographic device can at least keep some secrets and that *only computation leaks information*. This framework was then specialized into the *leakage resilient cryptography* model introduced by Dziembowski and Pietrzak in [14] which gave rise to a huge amount of subsequent works. In this model, a leaking computation is divided into elementary operations that are assumed to leak some information about their inputs through a leakage function whose range is bounded (*i.e.* taking values in $\{0,1\}^\lambda$ for some parameter λ). Many new leakage-resilient cryptographic primitives were proposed as well as so-called *compilers* that can make any computation secure in this model [16].

While the leakage resilient literature has achieved considerable theoretical advances, the considered model does not fully capture the physical reality of power or electromagnetic leakages (see for instance [24]). In particular for a

leakage function $f : \{0,1\}^m \to \{0,1\}^\lambda$, the parameter λ must be (significantly) smaller than m. This means, for instance, that the leakage of an AES computation should be smaller than 128 bits, whereas in practice a power trace resulting from an AES computation can take several kilobytes (or even megabytes). On the other hand, it is fair to assume that the side-channel leakage is *noisy* in such a way that the information $f(x)$ leaked by an elementary operation on a variable x is not enough to fully recover x. This intuition was formalized in the *noisy leakage model* introduced by Prouff and Rivain in 2013 [21]. In a nutshell, this model considers that an elementary operation with some input x leaks a *noisy leakage function* $f(x)$ (with a random tape parameter which is omitted from the presentation). The noisy feature is then captured by assuming that an observation of $f(x)$ only implies a bounded bias in the probability distribution of x. Namely the statistical distance between the distribution of x and the distribution of x given the observation $f(x)$ is bounded by some parameter δ which shall be called the *leakage rate* in the following. The function f is then said to be a δ-noisy leakage function. Notably, this model does not imply any restriction on the leakage size (*i.e.* on the range of f) but only on the amount of useful information it contains.

1.1 Related Works

Probing-Secure Circuits. In a seminal paper of 2003, Ishai, Sahai and Wagner considered the problem of building Boolean circuits secure against *probing attacks* [17]. In the so-called *probing model*, an adversary is allowed to adaptively probe up to t wires of the circuit. They show how to transform any circuit C with q logic gates into a circuit C' with $O(qt^2)$ logic gates that is secure against a t-probing adversary. Their scheme consists in encoding each Boolean variable x as a random sharing $(x_1, x_2, \ldots, x_n)$ satisfying $x_1 + x_2 + \ldots + x_n = x$ over $\mathbb{F}_2$, where $n = 2t + 1$. They show how to transform each logic gate into a *gadget* that work on encoded variables. Their construction is actually secure against an adversary that can adaptively place up to t probes per such gadget. The so-called ISW construction has since then served as a building block in many practical side-channel countermeasures known as *higher-order masking schemes* (see for instance [10,11,22]). Its efficiency has also been improved in recent works [5,6] which respectively show how to optimize the randomness consumption and the number of multiplications (while conserving similar asymptotic complexity).

Towards Noisy-Leakage Security. In [21], Prouff and Rivain proposed the first formal security proof for an ISW-like masking scheme in the noisy leakage model. In particular they generalize the previous work of Chari *et al.* [7] and show that in the presence of noisy leakage on the shares x_1, x_2, ..., x_n the information on x becomes negligible as n grows. Specifically, they show that for any δ-noisy leakage function f, the mutual information between x and the leakage $(f(x_1), f(x_2), \ldots, f(x_n))$ is of order $O(\delta^n)$. They also provide a security proof for full masked computation in the noisy leakage model, however their result

has two important limitations. First they assume the existence of a *leak-free component* that can refresh a sharing without leaking any information. Second, their proof can only tolerate an δ-noisy leakage with $\delta = O(1/n)$. Namely, the leakage rate must decrease linearly with the number of shares. Note that this second limitation is inherent to masking schemes based on the ISW construction since it implies that each share leaks $O(n)$ times. Some practical attacks have been exhibited that exploit this issue [4].

Avoiding Leak-Free Components. In [13], Duc, Dziembowski and Faust tackled the first of these two limitations. Namely they show how to avoid the requirement for a leak-free component with a nice and conceptually simpler security proof. Applying the Chernoff bound, they show that the ISW scheme is secure in the *δ-random probing model* in which each operation leaks its full input with a given probability $\delta = O(1/n)$ (and leaks nothing with probability $1 - \delta$). Their main contribution is then to show that any δ'-noisy leakage $f(x)$ can be simulated from a δ-random probing leakage $\phi(x)$ with $\delta' \leq \delta \cdot |\mathcal{X}|$, where $\mathcal{X}$ denotes the definition space of x. In other words, if the δ-random probing leakage of a computation contains no significant information, then neither does any δ'-noisy leakage of this computation as long as $\delta \leq \delta' \cdot |\mathcal{X}|$. The ISW scheme is therefore secure against δ'-noisy leakage for $\delta' = O(1/n|\mathcal{X}|)$. Note that for an arithmetic program working over some field $\mathbb{F}$, each elementary operation takes up to two inputs on $\mathbb{F}$, meaning $\mathcal{X} = \mathbb{F}^2$ and $\delta' = O(1/n|\mathbb{F}|^2)$. This way, the work of Duc *et al.* avoid the strong requirement of leak-free components. However, it still requires a leakage rate of $O(1/n)$.

Towards a Constant Leakage Rate. This second limitation was addressed by Andrychowicz, Dziembowski, and Faust [3]. They propose a scheme –that we call *ADF scheme* hereafter– which is based on Shamir's secret sharing [23] and a refreshing algorithm from expander graphs due to Ajtai [1]. The number of instructions in the protected program is multiplied by a factor $O(n^3)$ which can be reduced to $O(n^2 \log n)$ using an FFT-based multiplication. They show that this construction achieves security in the strong probing model where an adversary can adaptively place up to $O(1/\log n)$ probes per elementary operation. In the random probing model, the result is improved to a constant ratio. Applying the reduction from [13] they obtain the security in the noisy model for a leakage rate $\delta = O(1/|\mathbb{F}|^2)$.[1] For the standard version of their scheme based on Shamir's secret sharing, the base field $\mathbb{F}$ must be of size $O(n)$ which implies a leakage rate $\delta = O(1/n^2)$ in the noisy leakage model. Fortunately, their scheme can be improved by using secret sharing based on algebraic geometric codes [8] (at the cost of weaker parameters). As argued in [3], these codes operate over fields of

[1] Note that they obtain a leakage rate $O(1/|\mathbb{F}|)$ in the restrictive model where input variables leak independently. In the present paper, we make the more realistic assumption that the leakage function applies to the full input of each elementary operation.

constant size and hence there basic operations can be implemented by constant size Boolean circuits, which gives a $\delta = O(1)$ noisy leakage rate with the reduction from [13]. We show in this paper that a simple reduction actually exists to achieve $\delta = \tilde{O}(1)$ noisy leakage security from a random-probing secure scheme on a field $\mathbb{F} = O(n)$. This reduction could also be used to get tight noisy-leakage security for the ADF scheme without algebraic geometric codes (*i.e.* with simple Shamir's secret sharing).

Towards a Quasilinear Complexity. The leakage-secure schemes in the current state-of-the-art imply a (quasi)quadratic blow-up of the complexity: the protected program (or circuit) scales from $|P|$ arithmetic instructions to $\tilde{O}(|P|\, n^2)$. Another challenging issue is hence to bridge the gap between this $\tilde{O}(n^2)$ blow-up and the theoretically achievable $\tilde{O}(n)$ blow-up. In [3], the authors claim that the complexity of their scheme can be improved by using *packed secret sharing* [2,12]. As explained in [2], the use of packed secret sharing allows to securely compute (in the presence of leakage) an addition or a multiplication on several encoded values in parallel at the same asymptotic cost as a single operation with a standard secret sharing. Using the transform of [12], one can improve the complexity of the ADF scheme on an arithmetic program P from $O(|P|\, n^2 \log n)$ to $O(|P| \log |P|\, n^2 \log n/w)$ where w denotes the *width* of P. Roughly speaking, the width of P is the number of operations that can be computed in parallel throughout an execution of P which satisfies $w = O(|P|/d)$ where d is the depth of P (considered as a circuit). For a circuit of width $w = \Theta(n)$, this approach hence results in a complexity blow-up quasilinear in n. For a constant-size circuit (as the AES cipher) on the other hand, only a constant factor can be saved and the complexity blow-up remains (quasi)quadratic.

1.2 Our Contribution

In this paper we show how to securely compute *any* arithmetic program (or circuit) in the noisy leakage model with a leakage rate $\tilde{O}(1)$ and with complexity blow-up $\tilde{O}(n)$. Our scheme is conceptually very simple and also practically efficient provided that the computation relies on a base field $\mathbb{F}$ with appropriate structure.

We consider an *arithmetic program* P that executes basic arithmetic instructions over some prime field $\mathbb{F}$ (additions, subtractions, and multiplications) satisfying $|\mathbb{F}| = \alpha \cdot n + 1$ for n being a power of 2 (in particular $|\mathbb{F}| = O(n)$ as in [3]). Note that we prefer the terminology of (arithmetic) program composed of instructions to the terminology of (arithmetic) circuit composed of gates but the two notions are strictly equivalent.

Each internal variable $a \in \mathbb{F}$ of the computation is encoded into a random tuple $(a_0, a_1, \ldots, a_{n-1})$ that satisfies the relation $a = \sum_{i=0}^{n-1} a_i \omega^i$ for some random element $\omega \in \mathbb{F}$. In other words, a is encoded as the coefficient of a random n-degree polynomial Q satisfying $Q(\omega) = a$. It is worth noting that the security of our scheme does not rely on the secrecy of ω but on its random distribution. We then show how to transform each arithmetic instruction of P into a

corresponding secure gadget that works on encoded variables. Using a fast Number Theoretic Transform (NTT), we then achieve a multiplication gadget with $O(n \log n)$ instructions.

We first show that our scheme is secure in the δ-random-probing model for a parameter $\delta = O(1/\log n)$. Specifically, we show that for any program P with a constant number of instructions $|P|$, the advantage of a δ-random-probing adversary can be upper bounded by $\mathsf{negl}(\lambda) + \mathsf{negl}'(n)$ where negl and negl' are some negligible functions and where λ denotes some security parameter that impact the size of $\mathbb{F}$ (specifically we have $\lambda = \log \alpha$ where $|\mathbb{F}| = \alpha \cdot n + 1$). This is shown at the level of a single NTT-based secure multiplication in a first place. Then we show how to achieve compositional security, by interleaving each gadget by a refreshing procedure that has some *input-output separability* property. Using the Chernoff bound as in [13] we can then statistically bound the number of leaking intermediate variables in each gadget. Specifically, we show that the leakage in each gadget can be expressed as linear or quadratic combinations of the input shares that do not reveal any information with overwhelming probability (over the random choice of ω).

From our result in the random probing model, the security reduction of Duc *et al.* [13] directly implies that our construction is secure in the δ'-noisy leakage model for $\delta' = O(1/|\mathbb{F}|^2 \log n)$. However, since we require $|\mathbb{F}| = O(n)$ (as in the standard ADF scheme) this reduction is not satisfactory. We then refine the granularity of our computation by considering the noisy leakage model on *logical instructions* working on constant-size machine words. In this model, we provide a generic reduction from the random-probing model over $\mathbb{F}$ to the noisy leakage model on logical instructions. Namely we show that any arithmetic program Π secure under a δ-random-probing leakage gives rise to a functionally equivalent program Π' that is secure under a δ'-noisy leakage at the logical instruction level where $\delta' = \delta / O(\log |\mathbb{F}| \log \log |\mathbb{F}|)$. Applying this reduction, our construction achieves security in the δ'-noisy leakage model with $\delta' = O\big(1/((\log n)^2 \log \log n)\big)$ for a complexity blow-up of $O(n \log n)$.

Table 1 hereafter gives a asymptotic comparison of our scheme and the previous schemes in the literature (with noisy leakage security): the ISW scheme [17] with the reduction from [13], the ADF scheme with algebraic geometric codes [3], the ADF scheme improved with packed secret sharing [2,3] (ADF-PSS). We emphasize that for the latter case no detailed description and analysis have been provided.

Table 1. Asymptotic comparison of secure schemes in the noisy leakage model.

	ISW	ADF	ADF-PSS*	Our result
Leakage rate	$O(1/n)$	$O(1)$	$O(1)$	$\tilde{O}(1)$
Complexity blow-up	$O(n^2)$	$\tilde{O}(n^2)$	$\tilde{O}(n^2/w)$	$\tilde{O}(n)$

*w stands for the width of the protected program.

The paper is organized as follows. Section 2 provides background notions on the noisy leakage model and the considered adversary. In Sect. 3 we describe our secure quasilinear multiplication scheme and we prove its security in the random probing model. Section 4 then presents the refreshing procedure used to get compositional security and provides a security proof for a full arithmetic program. In Sect. 5 we give our generic reduction from the random-probing model over $\mathbb{F}$ to the noisy leakage model on logical instructions and we apply this reduction to our scheme to get our final result. We finally discuss practical aspects of our scheme and related open problems in Sect. 6.

2 Leakage and Adversary

In the rest of the paper, we shall denote by $x \leftarrow \mathcal{X}$ the action of picking x uniformly at random over some set $\mathcal{X}$. Similarly, for a probabilistic algorithm $\mathcal{A}$, we denote by $y \leftarrow \mathcal{A}(x)$ the action of running $\mathcal{A}$ on input x with a fresh random tape and setting y to the obtained result.

2.1 Noisy Leakage Model

The noisy leakage model introduced by Prouff and Rivain in [21] follows the *only computation leaks* paradigm [20]. In this paradigm, the computation is divided into subcomputations; each works on a subpart x of the current computation state and leaks some information $f(x)$, where f is called the *leakage function*. In practice, f is a so-called *randomized function* that takes two arguments, the input variable x and a random tape ρ that is large enough to model the leakage noise. A subcomputation with input variable x hence leaks $f(x, \rho)$ for a fresh random tape ρ. For the sake of simplicity, in the sequel we shall omit the parameter ρ and see $f(x)$ as a random realization of $f(x, \rho)$. Moreover, the definition space of the input x shall be called the *domain* of f, and we shall write $f : \mathcal{X} \to \mathcal{Y}$ for a randomized function with domain $\mathcal{X}$ and image space $\mathcal{Y}$.

In the noisy leakage model [21], a *noisy leakage function* f is defined as a randomized function such that an observation $f(x)$ only implies a bounded bias in the probability distribution of x. Namely, the statistical distance between the distributions of x and $(x \mid f(x))$ is assumed to be bounded by some bias δ. Let X and X' be two random variables defined over some set $\mathcal{X}$. We recall that the *statistical distance* between X and X' is defined as:

$$\Delta(X; X') = \frac{1}{2} \sum_{x \in \mathcal{X}} |\Pr(X = x) - \Pr(X' = x)|. \tag{1}$$

The notion of noisy leakage function is then formalized as follows:

Definition 1 ([21]). *A* δ-noisy leakage function *is a randomized function* $f : \mathcal{X} \to \mathcal{Y}$ *satisfying*

$$\sum_{y \in \mathcal{Y}} \Pr(f(X) = y) \cdot \Delta(X; (X \mid f(X) = y)) \leq \delta, \tag{2}$$

where X *is a uniform random variable over* $\mathcal{X}$.

In practice, the leaking input x might not be uniformly distributed but one must specify a distribution to have a consistent definition, and as argued in [21], the uniform distribution is a natural choice. Also note that in the original paper [21], the L_2 norm was used for the definition of the statistical distance while, as argued in [13], the L_1 norm is a more standard choice (that we also adopt in this paper).

A conceptually simpler model, known as the *random probing model*, was first used in [17] and formalized in the work of Duc, Dziembowski, and Faust [13]. Informally speaking, this model restricts the noisy leakage model to leakage functions that leak their entire input with a given probability. These *random-probing leakage functions* are formalized in the following definition.[2]

Definition 2. *A* δ-random-probing leakage function *is a randomized function* $\phi : \mathcal{X} \to \mathcal{X} \cup \{\perp\}$ *satisfying*

$$\phi(x) = \begin{cases} \perp & \text{with probability } 1-\delta \\ x & \text{with probability } \delta \end{cases} \tag{3}$$

It can be checked that such a function is a special case of δ-noisy leakage function.[3] Moreover, it has been shown by Duc, Dziembowski, and Faust [13] that every noisy leakage function f can be expressed as a composition $f = f' \circ \phi$ where ϕ is a random-probing leakage function. This important result enables to reduce noisy-leakage security to random-probing security. It is recalled hereafter:

Lemma 1 ([13]). *Let* $f : \mathcal{X} \to \mathcal{Y}$ *be a* δ*-noisy leakage function with* $\delta < \frac{1}{|\mathcal{X}|}$. *There exists a* δ'*-random-probing leakage function* $\phi : \mathcal{X} \to \mathcal{X} \cup \{\perp\}$ *and a randomized function* $f' \colon \mathcal{X} \cup \{\perp\} \to \mathcal{Y}$ *such that for every* $x \in \mathcal{X}$ *we have*

$$f(x) = f'(\phi(x)) \quad \text{and} \quad \delta' \leq \delta \cdot |\mathcal{X}|. \tag{4}$$

In the random-probing model, the total number of leaking operations can be statistically bounded using the Chernoff bound as suggested in [13,17]. We shall follow this approach in the present paper by using the following corollary.

Corollary 1 (Chernoff bound [9]). *The* δ*-random probing leakage of a computation composed of* N *elementary operations reveals the input of* $\ell > \delta N$ *of these elementary operations with probability lower than*

$$\psi(\ell, N) = \exp\Big(-\frac{(\ell - \delta N)^2}{\ell + \delta N}\Big). \tag{5}$$

If $\ell \leq \alpha n$ *and* $N = \beta n$, *for some* α, β *and* n *with* $\alpha/\beta > \delta$, *the above gives*

$$\psi(\alpha n, \beta n) = \exp\Big(-\frac{(\alpha - \delta\beta)^2}{\alpha + \delta\beta}\ n\Big). \tag{6}$$

[2] Note that we use a different terminology from [13] where these are called δ-*identity functions*.

[3] To be tighter, a δ-random-probing leakage function is a $\delta\,(1 - \frac{1}{|\mathcal{X}|})$-noisy function. This can be simply checked by evaluating (2).

2.2 Leakage Adversary

We consider computation schemes that *encode* the data of a program in order to make the leakage on the encoded data useless. An *encoding* Enc is a randomized function that maps an element $x \in \mathbb{F}$ to a n-tuple $\mathsf{Enc}(x) \in \mathbb{F}^n$, where n is called the encoding length, and for which a deterministic function $\mathsf{Dec} : \mathbb{F}^n \to \mathbb{F}$ exists that satisfies $\Pr(\mathsf{Dec}(\mathsf{Enc}(x)) = x) = 1$ for every $x \in \mathbb{F}$ (where the latter probability is taken over the encoding randomness).

Consider an *arithmetic program* P taking a string $\boldsymbol{x} \in \mathbb{F}^s$ as input and executing a sequence of instructions of the form $\mu_i \leftarrow \mu_j * \mu_k$, where $*$ denotes some operations over $\mathbb{F}$ (addition, subtraction, or multiplication) and where $[\mu_0, \mu_1, \ldots, \mu_T]$ denotes the memory of the program which is initialized with $\boldsymbol{x}$ (and some constants). To achieve leakage security, the program P is transformed into a functionally equivalent arithmetic program Π taking as input an encoded string $\mathsf{Enc}(\boldsymbol{x})$ (where the encoding simply applies to each coordinate of $\boldsymbol{x}$). According to the defined leakage model, each executed instruction of Π is then assumed to leak some noisy function $f(\mu_j, \mu_k)$ of its pair of inputs. It is further assumed that Π includes random sampling instructions $\mu_i \leftarrow \mathbb{F}$ that each leaks a noisy function of the generated random element $f(\mu_i)$. We denote the overall leakage by $\mathcal{L}(\Pi, \boldsymbol{x})$. The compiler is then said to be *leakage secure* if an observation of $\mathcal{L}(\Pi, \boldsymbol{x})$ does not reveal significant information about $\boldsymbol{x}$. More specifically, the leakage $\mathcal{L}(\Pi, \boldsymbol{x})$ must be indistinguishable from the leakage $\mathcal{L}(\Pi, \boldsymbol{x}')$ for every $\boldsymbol{x}' \in \mathbb{F}^s$. This security notion is formalized as follows:

Definition 3 (Leakage Security). *The program Π is* ε-leakage secure *(w.r.t. leakage functions $\mathcal{L}$) if every adversary $\mathcal{A}$ has advantage at most ε of distinguishing $\mathcal{L}(\Pi, \boldsymbol{x}_0)$ from $\mathcal{L}(\Pi, \boldsymbol{x}_1)$ for chosen $\boldsymbol{x}_0$ and $\boldsymbol{x}_1$, i.e. we have:*

$$\mathrm{Adv}_{\mathcal{A}}^{\Pi,\mathcal{L}} := \left| \mathrm{Succ}_{\mathcal{A}}^{\Pi,\mathcal{L}} - \frac{1}{2} \right| \leq \varepsilon \tag{7}$$

where

$$\mathrm{Succ}_{\mathcal{A}}^{\Pi,\mathcal{L}} = \Pr\left(\begin{array}{c} (\boldsymbol{x}_0, \boldsymbol{x}_1, \mu) \leftarrow \mathcal{A}(\perp) \\ b \leftarrow \{0,1\} \\ \boldsymbol{\ell} \leftarrow \mathcal{L}(\Pi, \boldsymbol{x}_b) \end{array} : \mathcal{A}(\boldsymbol{x}_0, \boldsymbol{x}_1, \mu, \boldsymbol{\ell}) = b \right). \tag{8}$$

In the above definition, $\mu \in \{0,1\}^*$ denotes any auxiliary information computed by the adversary during the first round when she chooses the inputs $\boldsymbol{x}_0$ and $\boldsymbol{x}_1$. Note that for the definition to be sound, we only consider adversaries $\mathcal{A}$ such that $\mathcal{A}(\perp)$ takes values over $\mathbb{F}^s \times \mathbb{F}^s \times \{0,1\}^*$ and $\mathcal{A}(\boldsymbol{x}_0, \boldsymbol{x}_1, \mu, \boldsymbol{\ell})$ takes values over $\{0,1\}$ for every input $(\boldsymbol{x}_0, \boldsymbol{x}_1, \mu, \boldsymbol{\ell}) \in \mathbb{F}^s \times \mathbb{F}^s \times \{0,1\}^* \times \mathrm{span}(\mathcal{L})$.

Lemma 1 provides a security reduction from the noisy leakage model to the random probing model. This is formalized in the following corollary:

Corollary 2. *Let Π be an arithmetic program that is ε-leakage secure w.r.t δ-random-probing leakage functions. Then Π is ε-leakage secure w.r.t δ'-noisy leakage functions, where $\delta' = \delta\,|\mathbb{F}|^2$.*

Note that in the original version of Lemma 1 (see [13]), the authors need the additional requirement that f' is efficiently *decidable* so that $f'(\phi(x))$ is computable in polynomial time in $|\mathcal{X}|$. We ignore this property in the present paper since our security statements consider adversaries with unlimited computational power.

3 Secure Multiplication in Quasilinear Complexity

In this section, we describe our encoding scheme and the associated secure multiplication. An important requirement of our construction is that the size n of the underlying encoding must divide $\frac{p-1}{2}$ where p is the characteristic of $\mathbb{F}$, that is $\mathbb{F}$ must contain the $2n$-th roots of unity. This implies that the size of the elements of $\mathbb{F}$ is in $\Omega(\log n)$. Without loss of generality, we further assume that n is a power of 2.

3.1 Our Encoding

Let ξ denote a primitive $2n$th root of unity in $\mathbb{F}$. Our encoding is based on a random element $\omega \in \mathbb{F}^*$ and is defined as follows:

Definition 4. *Let $\omega \in \mathbb{F}^*$ and $a \in \mathbb{F}$. An* ω-encoding *of a is a tuple $(a_i)_{i=0}^{n-1} \in \mathbb{F}^n$ satisfying $\sum_{i=0}^{n-1} a_i\omega^i = a$.*

Our encoding function Enc maps an element $a \in \mathbb{F}$ to a random element $\omega \in \mathbb{F}^*$ and a random uniform ω-encoding of a:

$$\mathsf{Enc}(a) = \langle \omega, (a_0, a_1, \ldots, a_{n-1}) \rangle. \tag{9}$$

The corresponding decoding function Dec is defined as:

$$\mathsf{Dec}\big(\langle \omega, (a_0, a_1, \ldots, a_{n-1}) \rangle\big) := \mathsf{Dec}_\omega(a_0, a_1, \ldots, a_{n-1}) := \sum_{i=0}^{n-1} a_i\omega^i \tag{10}$$

It is easy to check that we have $\Pr(\mathsf{Dec}(\mathsf{Enc}(a)) = a) = 1$ for every $a \in \mathbb{F}$. It is worth noting that the security of our scheme does not rely on the secrecy of ω but on its uniformity. Besides, we will consider that ω is systematically leaked to the adversary.

3.2 Multiplication of Encoded Variables

Let $(a_i)_{i=0}^{n-1}$ be an ω-encoding of a and $(b_i)_{i=0}^{n-1}$ be an ω-encoding of b. To compute an ω-encoding $(c_i)_{i=0}^{n-1}$ of $c = a \cdot b$ we use the NTT-based polynomial multiplication.

Specifically, we first apply the NTT on $(a_i)_i$ and $(b_i)_i$ to obtain the polynomial evaluations $u_j = \sum_{i=0}^{n-1} a_i(\xi^j)^i$ and $v_j = \sum_{i=0}^{n-1} b_i(\xi^j)^i$ for $j \in [\![0, 2n-1]\!]$. These evaluations are then pairwisely multiplied to get evaluations of the

product $s_j = (2n)^{-1} u_j \cdot v_j$ for $j \in [\![0, 2n-1]\!]$ (with a multiplicative factor $(2n)^{-1}$). Afterwards, we apply the inverse NTT to get coefficients t_i that satisfy $\sum_{i=0}^{2n-1} t_i \omega^i = (\sum_{i=0}^{n-1} a_i \omega^i) \cdot (\sum_{i=0}^{n-1} b_i \omega^i)$. Eventually, we apply a compression procedure to recover an n-size ω-encoding from the $2n$-size ω-encoding $(t_i)_i$. Due to the particular form of roots of unity, an NTT can be evaluated with a divide and conquer strategy in $3\, n \log n$ arithmetic instructions (a detailed description is given in Appendix A).

The overall process is summarized as follows:

$$\begin{aligned}
(u_0, u_1, \ldots, u_{2n-1}) &\leftarrow \mathsf{NTT}_\xi(a_0, a_1, \ldots, a_{n-1}, 0, \ldots, 0) \\
(r_0, r_1, \ldots, r_{2n-1}) &\leftarrow \mathsf{NTT}_\xi(b_0, b_1, \ldots, b_{n-1}, 0, \ldots, 0) \\
(s_0, s_1, \ldots, s_{2n-1}) &\leftarrow (2n)^{-1}(u_0 \cdot r_0, u_1 \cdot r_1, \ldots, u_{2n-1} \cdot r_{2n-1})) \\
(t_0, t_1, \ldots, t_{2n-1}) &\leftarrow \mathsf{NTT}_{\xi^{-1}}(s_0, s_1, \ldots, s_{2n-1}) \\
(c_0, c_1, \ldots, c_{n-1}) &\leftarrow \mathsf{compress}(t_0, t_1, \ldots, t_{2n-1})
\end{aligned}$$

Compression Procedure. After computing the inverse NTT, we get a double-size encoding $(t_i)_{i=0}^{2n-1}$ satisfying $\sum_{i=0}^{2n-1} t_i\, \omega^i = a \cdot b$. In order to obtain a standard encoding with n shares, we simply set $c_i = t_i + t_{n+i}\omega^n$ for $i \in [\![0, n-1]\!]$. It is not hard to see that the result is consistent.

3.3 Security in the Random Probing Model

We first focus on the NTT leakage security as it is the most complex part of our scheme, and then provide a security proof for the whole multiplication.

Security of the NTT. We have the following result:

Theorem 1. *Let ω be a uniform random element of $\mathbb{F}^*$, let $(a_i)_{i=0}^{n-1}$ be a uniform ω-encoding of some variable a and let $\delta < 1/(6 \log n)$. The NTT_ξ procedure on input $(a_i)_{i=0}^{n-1}$ is ε-leakage secure in the δ-random-probing leakage model, where*

$$\varepsilon = \frac{n}{|\mathbb{F}|} + \exp\Big(- \frac{(1 - 6\delta \log n)^2}{4}\ n\Big). \tag{11}$$

The rest of the section gives a proof of Theorem 1. During the computation of the NTT on an ω-encoding $(a_i)_{i=0}^{n-1}$ of a, all the leaking intermediate variables (*i.e.* the inputs of arithmetic instructions) are linear combinations of the a_i's. Specifically, every intermediate variable v occurring in the NTT computation can be expressed as

$$v = \sum_{i=0}^{n-1} \alpha_i a_i \tag{12}$$

where the α_i's are constant coefficients over $\mathbb{F}$. In the following, we shall use the notation

$$[v] = (\alpha_0, \alpha_1, \ldots, \alpha_{n-1})^{\mathsf{t}} \tag{13}$$

for the column vector of coefficients of such an intermediate variable. Similarly, we shall denote $[a] = (1, \omega, \omega^2, \ldots, \omega^{n-1})^{\mathsf{t}}$ since we have $a = \sum_{i=0}^{n-1} \omega^i a_i$ by definition. Moreover, we will denote by $[v_0, v_1, \ldots, v_\ell]$ the matrix with column vectors $[v_0], [v_1], \ldots, [v_\ell]$. In particular, we have $[a_0, a_1, \ldots, a_{n-1}] = I_n$ (where I_n stands for the identity matrix of dimension n over $\mathbb{F}$) and for $u_i = \sum_{j=0}^{n-1} a_j (\xi^i)^j$ (the output elements of the NTT), the matrix $[u_0, u_1, \ldots, u_{n-1}]$ is a Vandermonde matrix.

First consider an adversary that recovers $\ell < n$ intermediate variables in the computation of the NTT, denoted $v_1, v_2, \ldots, v_\ell$. Without loss of generality, we assume that these intermediate variables are linearly independent (otherwise the adversary equivalently gets less than ℓ intermediate variables), which means that the matrix $[v_1, v_2, \ldots, v_\ell]$ has full rank. The following lemma gives a necessary and sufficient condition for such a leakage to be statistically independent of a.

Lemma 2. *Let $v_1, v_2, \ldots, v_\ell$ be a set of $\ell < n$ intermediate variables of the NTT on input a uniform ω-encoding of a variable a. The distribution of the tuple $(v_1, v_2, \ldots, v_\ell)$ is statistically independent of a iff*

$$[a] \notin \mathrm{span}([v_1, \ldots, v_\ell]) \ , \tag{14}$$

where span$(\cdot)$ *refers to the linear span of the input matrix.*

Proof. If $[a] \in \mathrm{span}([v_1, \ldots, v_\ell])$ then there exists constants $\gamma_1, \gamma_2, \ldots, \gamma_\ell$ such that $[a] = \sum_i \gamma_i [v_i]$ implying $a = \sum_i \gamma_i v_i$, and the distribution $(v_1, v_2, \ldots, v_\ell)$ is hence statistically dependent on a. On the other hand, if $[a] \notin \mathrm{span}([v_1, \ldots, v_\ell])$, then the system

$$\begin{cases} a = \sum_{j=0}^{n-1} \omega^j a_j & = \gamma_0 \\ v_1 = \sum_{j=0}^{n-1} \alpha_{1,j} a_j & = \gamma_1 \\ v_2 = \sum_{j=0}^{n-1} \alpha_{2,j} a_j & = \gamma_2 \\ \vdots & \\ v_\ell = \sum_{j=0}^{n-1} \alpha_{t,j} a_j & = \gamma_\ell \end{cases}$$

has $|\mathbb{F}|^{n-(\ell+1)}$ solutions $(a_0, a_1, \ldots, a_{n-1})$ for every $(\gamma_0, \gamma_1, \ldots, \gamma_\ell) \in \mathbb{F}^{\ell+1}$. This implies the statistical independence between a and $(v_1, v_2, \ldots, v_\ell)$. □

The following lemma gives an upper bound on the probability that the above condition is not fulfilled.

Lemma 3. *Let ω be a uniform random element in $\mathbb{F}^*$ and let $v_1, v_2, \ldots, v_\ell$ be a set of $\ell < n$ linearly independent intermediate variables of the NTT on input an ω-encoding of a variable a. We have:*

$$\Pr\left([a] \in \mathrm{span}([v_1, \ldots, v_\ell])\right) \leq \frac{\ell}{|\mathbb{F}| - 1} < \frac{n}{|\mathbb{F}|} \ , \tag{15}$$

where the above probability is taken over a uniform random choice of ω.

Proof. Let us denote $A(x) = \sum_{i=0}^{n-1} a_i x^i$ so that $[A(\alpha)] = (1, \alpha, \alpha^2, \ldots, \alpha^{n-1})^{\mathrm{t}}$ for every $\alpha \in \mathbb{F}$, and in particular $[a] = [A(\omega)]$. For any distinct $\ell + 1$ elements $\alpha_1, \alpha_2, \cdots \alpha_{\ell+1} \in \mathbb{F}^*$, the matrix $[A(\alpha_1), A(\alpha_2), \ldots, A(\alpha_{\ell+1})]$ has full rank since it is a Vandermonde matrix with distinct input entries. This directly implies:

$$\underbrace{\mathrm{span}([A(\alpha_1), A(\alpha_2), \ldots, A(\alpha_{\ell+1})])}_{\dim\ \ell+1} \nsubseteq \underbrace{\mathrm{span}([v_1, \ldots, v_\ell])}_{\dim\ \ell}, \tag{16}$$

hence the set $\Omega = \{\alpha \mid [A(\alpha)] \in \mathrm{span}([v_0, v_1, \ldots, v_\ell])\}$ has cardinality at most ℓ. By the uniform distribution of ω, we then have a probability at most $\ell/(|\mathbb{F}|-1) \leq n/|\mathbb{F}|$ to have $\omega \in \Omega$ that is to have $[a] \in \mathrm{span}([v_1, \ldots, v_\ell])$. □

We now have all the ingredients to prove Theorem 1.

Proof. (Theorem 1) We will show that for any adversary $\mathcal{A}$, the advantage $\mathrm{Adv}_{\mathcal{A}}^{\mathsf{NTT},\mathcal{L}}$ in distinguishing $\mathcal{L}(\mathsf{NTT}, \mathsf{Enc}(a^{(0)}))$ from $\mathcal{L}(\mathsf{NTT}, \mathsf{Enc}(a^{(1)}))$ for any chosen elements $a^{(0)}, a^{(1)} \in \mathbb{F}$ is lower than ε, where $\mathcal{L}(\mathsf{NTT}, \mathsf{Enc}(a))$ denotes the δ-random-probing leakage of the procedure NTT_ξ on input $\mathsf{Enc}(a) = \langle \omega, (a_i)_{i=0}^{n-1} \rangle$. Note that this leakage is a tuple in which each coordinate corresponds to an arithmetic instruction in the computation of NTT_ξ that either equals $\perp$ (with probability $1 - \delta$) or the input of the instruction. We recall that the advantage is defined as $\mathrm{Adv}_{\mathcal{A}}^{\mathsf{NTT},\mathcal{L}} = \left|\mathrm{Succ}_{\mathcal{A}}^{\mathsf{NTT},\mathcal{L}} - \frac{1}{2}\right|$ where

$$\mathrm{Succ}_{\mathcal{A}}^{\mathsf{NTT},\mathcal{L}} = \Pr\left(\begin{array}{c} (a^{(0)}, a^{(1)}, \mu) \leftarrow \mathcal{A}(\perp) \\ b \leftarrow \{0,1\} \\ \boldsymbol{\ell} \leftarrow \mathcal{L}(\mathsf{NTT}, a^{(b)}) \end{array} : \ \mathcal{A}(a^{(0)}, a^{(1)}, \mu, \boldsymbol{\ell}) = b \right) \tag{17}$$

Without loss of generality, we assume $\mathrm{Succ}_{\mathcal{A}}^{\mathsf{NTT},\mathcal{L}} \geq \frac{1}{2}$. Indeed, for any adversary with success probability $\frac{1}{2} - \mathrm{Adv}_{\mathcal{A}}^{\mathsf{NTT},\mathcal{L}}$, there exists an adversary $\mathcal{A}'$ with success probability $\frac{1}{2} + \mathrm{Adv}_{\mathcal{A}}^{\mathsf{NTT},\mathcal{L}}$ (defined as $\mathcal{A}'(a^{(0)}, a^{(1)}, \boldsymbol{\ell}) = 1 - \mathcal{A}(a^{(0)}, a^{(1)}, \mu, \boldsymbol{\ell})$).

The procedure NTT_ξ is composed of $N = 3n \log n$ arithmetic instructions. In the δ-random-probing model, each of these instructions leaks its input(s) with probability δ. The number of instructions that leak hence follows a binomial distribution with parameters N and δ. Let us denote by max_ℓ the event that ℓ or less instructions leak in the random-probing leakage $\mathcal{L}(\mathsf{NTT}, \mathsf{Enc}(a))$. Since each instruction takes at most two inputs over $\mathbb{F}$, the adversary gets the values of at most 2ℓ intermediate variables whenever max_ℓ occurs. By the Chernoff bound (see Corollary 1), the probability that more than $\ell > N\delta$ arithmetic instructions leak, namely the probability that $\neg\mathsf{max}_\ell$ occurs, satisfies:

$$\Pr(\neg\mathsf{max}_\ell) \leq \psi(\ell, N). \tag{18}$$

From $N = 3n \log n$ and $\ell < \frac{n}{2}$, we get that:

$$\Pr(\neg\mathsf{max}_\ell) \leq \exp\left(-\frac{(1 - 6\delta \log n)^2}{2 + 12\delta \log n}\ n \right) \leq \exp\left(-\frac{(1 - 6\delta \log n)^2}{4}\ n \right). \tag{19}$$

Now let assume that max_ℓ occurs for some $\ell < \frac{n}{2}$ and let denote v_1, v_2, ..., $v_{2\ell}$ the recovered intermediate variables. Without loss of generality, we assume that the recovered intermediate variables are linearly independent. Let us then denote by free the event that $[a] \notin \mathrm{span}([v_1, \ldots, v_\ell])$. By Lemma 3, we have

$$\Pr(\neg\mathsf{free}) < \frac{n}{|\mathbb{F}|}. \tag{20}$$

And let finally denote by succ the event that $\mathcal{A}$ outputs the right bit b on input $(a^{(0)}, a^{(1)}, \mu, \boldsymbol{\ell})$ so that $\mathrm{Succ}_{\mathcal{A}}^{\mathsf{NTT},\mathcal{L}} = \Pr(\mathsf{succ})$. We can then write:

$$\begin{aligned}\mathrm{Succ}_{\mathcal{A}}^{\mathsf{NTT},\mathcal{L}} = \Pr(\mathsf{max}_\ell)\Pr(\mathsf{succ} \mid \mathsf{max}_\ell) \; &+ \Pr(\neg\mathsf{max}_\ell)\Pr(\mathsf{succ} \mid \neg\mathsf{max}_\ell) \\ &\leq \Pr(\mathsf{succ} \mid \mathsf{max}_\ell) + \Pr(\neg\mathsf{max}_\ell). \end{aligned} \tag{21}$$

In the same way, we have

$$\Pr(\mathsf{succ} \mid \mathsf{max}_\ell) \leq \Pr(\mathsf{succ} \mid \mathsf{max}_\ell \cap \mathsf{free}) + \Pr(\neg\mathsf{free}). \tag{22}$$

By Lemma 2, we have that the leakage $\boldsymbol{\ell}$ is statistically independent of $a^{(b)}$ in (17) whenever $\mathsf{max}_\ell \cap \mathsf{free}$ occurs. This directly implies $\Pr(\mathsf{succ} \mid \mathsf{max}_\ell \cap \mathsf{free}) = \frac{1}{2}$, which gives

$$\mathrm{Succ}_{\mathcal{A}}^{\mathsf{NTT},\mathcal{L}} < \frac{1}{2} + \Pr(\neg\mathsf{max}_\ell) + \Pr(\neg\mathsf{free}). \tag{23}$$

Hence, we finally get

$$\mathrm{Adv}_{\mathcal{A}}^{\mathsf{NTT},\mathcal{L}} < \Pr(\neg\mathsf{max}_\ell) + \Pr(\neg\mathsf{free}) = \frac{n}{|\mathbb{F}|} + \exp\Big(-\frac{(1-6\delta\log n)^2}{4}\, n\Big), \tag{24}$$

which concludes the proof. □

Security of the Full Multiplication. We now prove the security of the full multiplication. We have the following result:

Theorem 2. *Let ω be a uniform random element of $\mathbb{F}^*$, let $(a_i)_{i=0}^{n-1}$ and $(b_i)_{i=0}^{n-1}$ be uniform ω-encodings of some variables a and b, and let $\delta < 1/(21\log n)$. The above NTT-based multiplication procedure on input $(a_i)_{i=0}^{n-1}$ and $(b_i)_{i=0}^{n-1}$ is ε-leakage secure in the δ-random-probing leakage model, where*

$$\varepsilon = \frac{2\,n}{|\mathbb{F}|} + 5\exp\Big(-\frac{(1-21\delta\log n)^2}{14}\, n\Big). \tag{25}$$

Proof. The full multiplication is composed of five successive steps:

1. the NTT on input $(a_i)_i$,
2. the NTT on input $(b_i)_i$,
3. the pairwise multiplications $(2n)^{-1} \cdot u_i \cdot r_i$,
4. the NTT on input $(s_i)_i$,
5. the final compression on input $(t_i)_i$.

Let us denote by ℓ_1, ℓ_2, ..., ℓ_5 the number of operations that leak at each of these steps. Since each operation takes up to 2 input variables, the adversary then gets:

- up to $2\ell_1$ variables from the first NTT, each variable providing a linear equation in the a_i's;
- up to $2\ell_2$ variables from the second NTT, each variable providing a linear equation in the b_i's;
- up to ℓ_3 pairs (u_i, r_i);[4] each pair providing a linear equation in the a_i's and a linear equation in the b_i's;
- up to $2\ell_4$ variables in the third NTT (the inverse NTT), each variable providing a linear equation in the s_j's;
- up to ℓ_5 pairs (t_i, t_{i+n}),[5] each pair providing two linear equations in the s_j's.

To sum up, the adversary gets a system composed of

- up to $\ell_1^* = 2\ell_1 + \ell_3$ linear equations of the form

$$\sum_{i=1}^{n} \alpha_{k,i} \cdot a_i = \eta_k \quad \text{for } k = 1, \ldots, \ell_1^* \tag{26}$$

- up to $\ell_2^* = 2\ell_2 + \ell_3$ linear equations of the form

$$\sum_{i=1}^{n} \beta_{k,i} \cdot b_i = \nu_k \quad \text{for } k = 1, \ldots, \ell_2^* \tag{27}$$

- up to $\ell_3^* = 2\ell_4 + 2\ell_5$ linear equations of the form

$$\sum_{j=1}^{2n} \gamma_{k,j} \cdot s_j = \chi_k \quad \text{for } k = 1, \ldots, \ell_3^* \tag{28}$$

we have $s_j = (2n)^{-1} u_j r_j$ for every j, and since u_j and r_j can be expressed as linear combinations of $(a_i)_i$ and of $(b_i)_i$ respectively, for every j, the last ℓ_3^* equations can be rewritten as:

$$\sum_{i=1}^{n} \gamma'_{k,i} \cdot b_i = \chi_k \quad \text{for } k = 1, \ldots, \ell_3^* \tag{29}$$

where the $\gamma'_{k,i}$'s are coefficients that depend on the a_i's.

From these equations, the attacker gains the knowledge that:

[4] Either a multiplication of the form $(2n)^{-1} \cdot u_i$ or a multiplication of the form $(2n)^{-1} u_i \cdot r_i$ leaks. In both cases we consider that the pair (u_i, r_i) is revealed to the adversary.

[5] Either a multiplication $\omega^n \cdot t_{i+n}$ or an addition $t_i + \omega^n t_{i+n}$ leaks. In both cases we consider that the pair (t_i, t_{i+n}) is revealed to the adversary.

1. the encoding $(a_i)_{i=0}^{n-1}$ belongs to some vectorial space
$$\mathcal{S}_1 = \{\boldsymbol{x} \in \mathbb{F}^n \ ; \ M_1 \cdot \boldsymbol{x} = \boldsymbol{\eta}\} \tag{30}$$
of dimension at least $n - \ell_1^*$ where M_1 is the matrix with coefficients $\alpha_{k,i}$'s, and $\boldsymbol{\eta}$ is the vector with coordinates η_k,
2. the encoding $(b_i)_{i=0}^{n-1}$ then belongs to some vectorial space
$$\mathcal{S}_2 = \{\boldsymbol{x} \in \mathbb{F}^n \ ; \ M_2 \cdot \boldsymbol{x} = (\boldsymbol{\nu}, \boldsymbol{\chi})\} \tag{31}$$
of dimension at least $n - \ell_2^* - \ell_3^*$ where M_2 is the matrix with coefficients $\beta_{k,i}$'s and $\gamma'_{k,i}$'s and $(\boldsymbol{\nu}, \boldsymbol{\chi})$ is the vector with coordinates ν_k and χ_k.

Following the demonstration of Lemma 2, it can be checked that if
$$(1, \omega, \ldots, \omega^{n-1}) \notin \mathrm{span}(M_1) \quad \text{and} \quad (1, \omega, \ldots, \omega^{n-1}) \notin \mathrm{span}(M_2),$$
then the full leakage of the multiplication is statistically independent of a and b, namely the leakage security holds. These two events are denoted free_1 and free_2 hereafter.

Then, following the demonstration of Lemma 3, free_1 occurs with probability at least $1 - \frac{n}{|\mathbb{F}|}$ over a random choice of ω, provided that we have $\mathrm{rank}(M_1) < n$. Then, since the vectorial space $\mathcal{S}_1$ is independent of ω, any possible choice of $(a_i)_{i=0}^{n-1} \in \mathcal{S}_1$ gives rise to some coefficients $\gamma'_{k,i}$'s independent of ω and we have that free_2 occurs with probability at least $1 - \frac{n}{|\mathbb{F}|}$ over a random choice of ω as long as we have $\mathrm{rank}(M_2) < n$. The two conditions on the ranks of M_1 and M_2 are then fulfilled whenever we have
$$\ell_1^* = 2\ell_1 + \ell_3 < n, \tag{32}$$
and
$$\ell_2^* + \ell_3^* = 2\ell_2 + \ell_3 + 2\ell_4 + 2\ell_5 < n. \tag{33}$$
Let us denote max_i the event that the number of leaking operations ℓ_i at step i is lower than $n/7$, for every i. If max_i occurs for every $i \in \{1, 2, 3, 4, 5\}$, then two above inequalities are well satisfied.

By applying the Chernoff bound, we hence get:
$$\Pr(\neg\mathsf{max}_i) \leq \psi\big(\frac{n}{7}, N_i\big), \tag{34}$$
where N_i is the number of operations at step i, which satisfies $N_i \leq 3n \log n$, which gives
$$\Pr(\neg\mathsf{max}_i) \leq \psi\big(\frac{n}{7}, 3n \log n\big) \leq \exp\Big(-\frac{(1 - 21\delta \log n)^2}{14}\ n\Big). \tag{35}$$
We finally get that the multiplication is ε-leakage secure with
$$\begin{aligned}\varepsilon < \ & \Pr(\neg\mathsf{max}_1) + \Pr(\neg\mathsf{max}_2) + \cdots + \Pr(\neg\mathsf{max}_5) \\ & + \underbrace{\Pr(\neg\mathsf{free}_1 \mid \mathsf{max}_1 \wedge \ldots \wedge \mathsf{max}_5)}_{<n/|\mathbb{F}|} + \underbrace{\Pr(\neg\mathsf{free}_2 \mid \mathsf{max}_1 \wedge \ldots \wedge \mathsf{max}_5)}_{<n/|\mathbb{F}|}.\end{aligned} \tag{36}$$

□

4 Compositional Security for Arithmetic Programs

In this section we show how to obtain leakage security for a full arithmetic program, composed of several multiplications, additions and subtractions. Since computing addition and subtraction on encoded variables is quite simple, our main contribution is to describe a refreshing procedure which allows us to achieve compositional security.

We first describe our refreshing procedure before explaining how to transform an arithmetic program into a leakage-secure equivalent arithmetic program. Then we provide our compositional security proof.

4.1 Refreshing Procedure

Our refreshing procedure is based on the common approach of adding an encoding of 0. Let $(a_i)_{i=0}^{n-1}$ be an ω-encoding of a variable a. We refresh it into an ω-encoding $(a'_i)_{i=0}^{n-1}$ of a as follows:

1. sample a random ω-encoding $(r_0, r_1, \ldots, r_{n-1}) \leftarrow \mathsf{Enc}_\omega(0)$
2. set $a'_i = a_i + r_i$ for $i = 0$ to $n-1$

The main issue with such an approach is the design of a scheme to sample an encoding of 0 which has the right features for the compositional security. As detailed later, we can prove the compositional security as long as our construction satisfies the two following properties:

- **Uniformity:** it outputs a uniform ω-encoding of 0;
- **Output linearity:** its intermediate variables (*i.e.* the input of elementary operations in the sampler) can each be expressed as a linear combination of the output shares $(r_i)_i$.

We now describe an $\mathsf{Enc}_\omega(0)$ sampler which satisfies these two properties.

Sampling Encodings of 0. At the beginning of the computation of Π, a random ω-encoding of 0 is generated. This is simply done by randomly picking $n-1$ of the n shares and computing the last one accordingly. We will denote by $(e_i)_{i=0}^{n-1}$ this encoding. Note that just as for ω, this encoding can be fully leaked to the adversary. Our sampler then works as follows:

1. pick $n-1$ random values $u_0, u_1, \ldots, u_{n-2}$ over $\mathbb{F}$,
2. output $(r_i)_{i=0}^{n-1} = \mathsf{NTTMult}((u_0, u_1, \ldots, u_{n-2}, 0), (e_0, e_1, \ldots, e_{n-1}))$

where $\mathsf{NTTMult}$ is the NTT-based multiplication described in Sect. 3.

It is not hard to see that the result is indeed an encoding of 0: since the $(e_i)_i$ encode a 0, then the encoded product is also a 0. The uniformity is slightly more tricky to see. We claim that with overwhelming probability (over the random choice of $(e_i)_i$), the function:

$$(u_0, u_1, \ldots, u_{n-2}) \mapsto \mathsf{NTTMult}((u_0, u_1, \ldots, u_{n-2}, 0), (e_0, e_1, \ldots, e_{n-1})), \quad (37)$$

is invertible. This function is indeed linear and it can be seen as a multiplication by an $(n-1)\times n$ matrix. We empirically validated that this matrix is of rank $n-1$ with overwhelming probability.[6] By discarding one column we can get a full-rank square matrix of dimension $n-1$, allowing the recovery of the $(u_0, u_1, \ldots, u_{n-2})$ from output encoding. Therefore, we have a one-to-one mapping between the vectors $(u_0, u_1, \ldots, u_{n-2}) \in \mathbb{F}^{n-1}$ and the ω-encodings of 0, $(r_i)_{i=0}^{n-1} \in \mathbb{F}^n$ with $\mathsf{Dec}_\omega((r_i)_{i=0}^{n-1}) = 0$.

The output linearity is a direct consequence of the above. Since the u_i's can be expressed as linear combinations of the r_i's, then all the intermediate variables of the sampling procedure can be expressed as such linear combinations as well.

4.2 Arithmetic Program Compiler

We consider an arithmetic program P processing variables defined over a prime field $\mathbb{F}$. We show how to transform such a program into a leakage-secure arithmetic program Π. Each arithmetic instruction of P gives rise to a corresponding *gadget* in Π that works on encodings. We describe these different gadgets hereafter.

Copy Gadget. The copy gadget simply consists in applying a refreshing procedure to copy an encoded variable into the same freshly encoded variable. Let $(a_i)_{i=0}^{n-1}$ be an ω-encoding of a. The copy gadget compute an ω-encoding $(a'_i)_{i=0}^{n-1}$ of a as:

$$(a'_0, a'_1, \ldots, a'_{n-1}) \leftarrow \mathsf{refresh}(a_0, a_1, \ldots, a_{n-1})$$

The copy gadget is used whenever an output ω-encoding $(a_i)_{i=0}^{n-1}$ from some previous gadget is used as an input of several following gadgets. If $(a_i)_{i=0}^{n-1}$ is to be used in input of N following gadgets, one makes $N-1$ extra copies (in such a way that each new copy enters the next copy gadget):

$$(a_i)_{i=0}^{n-1} \rightarrow (a_i^{(2)})_{i=0}^{n-1} \rightarrow \cdots \rightarrow (a_i^{(N)})_{i=0}^{n-1}$$

This way, each fresh encoding $(a_i^{(j)})_{i=0}^{n-1}$ enters at most two different gadgets: the copy gadget and one of the N computation gadgets.

Addition Gadget. Let $(a_i)_{i=0}^{n-1}$ be an ω-encoding of a and $(b_i)_{i=0}^{n-1}$ be an ω-encoding of b. To compute an ω-encoding $(c_i)_{i=0}^{n-1}$ of $c = a + b$, we simply compute:

$$(c_0, c_1, \ldots, c_{n-1}) \leftarrow \mathsf{refresh}(a_0 + b_0, a_1 + b_1, \ldots, a_{n-1} + b_{n-1})$$

[6] To avoid to rely on an empirical assumption, one could easily check whether the generated encoding $(e_i)_i$ gives rise to a full-rank linear transformation.

Subtraction Gadget. Let $(a_i)_{i=0}^{n-1}$ be an ω-encoding of a and $(b_i)_{i=0}^{n-1}$ be an ω-encoding of b. To compute an ω-encoding $(c_i)_{i=0}^{n-1}$ of $c = a + b$, we simply compute:

$$(c_0, c_1, \ldots, c_{n-1}) \leftarrow \mathsf{refresh}(a_0 - b_0, a_1 - b_1, \ldots, a_{n-1} - b_{n-1})$$

Multiplication Gadget. Let $(a_i)_{i=0}^{n-1}$ be an ω-encoding of a and $(b_i)_{i=0}^{n-1}$ be an ω-encoding of b. To compute an ω-encoding $(c_i)_{i=0}^{n-1}$ of $c = a \cdot b$, we simply compute:

$$(c_0, c_1, \ldots, c_{n-1}) \leftarrow \mathsf{refresh}\big(\mathsf{NTTMult}((a_0, a_1, \ldots, a_{n-1}), (b_0, b_1, \ldots, b_{n-1}))\big)$$

where $\mathsf{NTTMult}$ denotes the NTT-based multiplication described in Sect. 3.

4.3 Compositional Security

The compositional security of our construction is based on the two following properties of the refreshing procedure:

- **Uniformity:** for a given $\omega \in \mathbb{F}^*$ and a given value $a \in \mathbb{F}$, the ω-encoding $(a'_i)_{i=0}^{n-1}$ in output of the refreshing procedure is uniformly distributed and independent of the input ω-encoding $(a_i)_{i=0}^{n-1}$;
- **I/O linear separability:** the intermediate variables of the refreshing procedure can each be expressed as a deterministic function of a linear combination of the $(a_i)_i$ and a linear combination of the $(a'_i)_i$.

The uniformity property is a direct consequence of the uniformity of the $\mathsf{Enc}_\omega(0)$ sampler. The I/O linear separability holds from the output linearity of the $\mathsf{Enc}_\omega(0)$ sampler since the shares $(r_i)_i$ output by the sampler satisfy $r_i = a'_i - a_i$ for every i, implying that any linear combination $\sum_i \gamma_i r_i$ equals $\sum_i \gamma_i a'_i - \sum_i \gamma_i a_i$ and is hence a deterministic function of a linear combination $\sum_i \gamma_i a'_i$ and a linear combination $\sum_i \gamma_i a_i$.

The I/O linear separability of the refreshing procedure implies that its leakage can be split into some leakage depending only on its input encoding, which is the output (before refreshing) from a previous gadget, and some leakage depending only on its output encoding, which is the input of a next gadget. This way, the full leakage can be split into subleakages each depending on the input/output of one gadget. Moreover, the uniformity property implies that all these subleakages are mutually independent. They can hence be analyzed separately: if none of them reveal information, then the full leakage does not reveal information either.

The compositional security of our construction is formalized in the following theorem.

Theorem 3. *Let P be an arithmetic program taking some input $\boldsymbol{x} \in \mathbb{F}^s$ and let Π denotes the corresponding program protected with n-size encodings as described*

above. For every $\delta < 1/(33 \log n)$, Π is ε-leakage secure in the δ-random-probing model where

$$\varepsilon = 3|P| \cdot \left(2 \exp\Big(- \frac{(1 - 33\delta \log n)^2}{22} \; n \Big) + \frac{2n}{|\mathbb{F}|} \right), \tag{38}$$

where $|P|$ denotes the size of P i.e. its number of arithmetic instructions.

Proof. Let $|\Pi|$ denotes the number of gadgets in Π. Since the output of each gadget is refreshed (and nothing more), the number of call to the refreshing procedure is also $|\Pi|$. Each arithmetic instruction in P gives rise to one associated gadget, plus up to 2 copy gadgets if necessary. We hence deduce $|\Pi| \leq 3|P|$.

Let us denote by rmax the event that at most $\frac{n}{11}$ operations leak in each refreshing. By applying the Chernoff bound (see Corollary 1), we have

$$\Pr(\neg \mathsf{rmax}) \leq |\Pi| \cdot \psi\big(\frac{n}{11}, N_{\mathrm{ref}}\big), \tag{39}$$

where N_{ref} denotes the number of elementary operations in the refreshing procedure. Let us further denote by gmax the event that at most $\frac{n}{11}$ operations leak in each gadget (without refreshing). In the same way as above, we have

$$\Pr(\neg \mathsf{gmax}) \leq \sum_{i=1}^{|\Pi|} \psi\big(\frac{n}{11}, N^{(i)}\big) \leq |\Pi| \cdot \psi\big(\frac{n}{11}, N_{\mathrm{gad}}\big), \tag{40}$$

where $N^{(i)}$ denotes the number of elementary operations in the ith gadget and where N_{gad} denotes the max (which is reached by the multiplication gadget).

In the following, we shall denote by $(a_j^{(i)})_j$ and $(b_j^{(i)})_j$ the input encodings of the ith gadget of Π and by $(c_j^{(i)})_j$ the output encoding (before refreshing) of the ith gadget of Π. Let us further denote by $\mathcal{L}$ the full δ-random-probing leakage of Π, so that we have:

$$\mathcal{L} = \bigcup_{i=1}^{|\Pi|} \mathcal{G}^{(i)} \; \cup \; \bigcup_{i=1}^{|\Pi|} \mathcal{R}^{(i)} \tag{41}$$

where $\mathcal{G}^{(i)}$ denotes the leakage from the ith gadget (without refreshing) and where $\mathcal{R}^{(i)}$ denotes the leakage of the ith refresh. Specifically, $\mathcal{G}^{(i)}$ and $\mathcal{R}^{(i)}$ are families of intermediate variables (inputs of elementary operations) that are revealed by the δ-random-probing leakage. If rmax and gmax occurs, we have $|\mathcal{G}^{(i)}| \leq \frac{2n}{11}$ and $|\mathcal{R}^{(i)}| \leq \frac{2n}{11}$.

According the the I/O linear separability property of the refreshing procedure, we can define a *separated leakage* $\mathcal{L}'$ as

$$\mathcal{L}' = \bigcup_{i=1}^{|\Pi|} \big(\mathcal{G}^{(i)} \cup \mathcal{A}^{(i)} \cup \mathcal{B}^{(i)} \cup \mathcal{C}^{(i)} \big) \tag{42}$$

where $\mathcal{A}^{(i)}$ is a set of linear combinations of $(a_j^{(i)})_j$, $\mathcal{B}^{(i)}$ is a set of linear combinations of $(b_j^{(i)})_j$, $\mathcal{C}^{(i)}$ is a set of linear combinations of $(c_j^{(i)})_j$, such that $\mathcal{L}$ is a deterministic function of $\mathcal{L}'$. This implies that if $\mathcal{L}'$ is statistically independent of the program input $\boldsymbol{x}$, then so is $\mathcal{L}$. The remaining of the proof consists in showing that the former occurs with overwhelming probability (for a sound choice of the parameters).

We shall bound the probability (over the distribution of ω) that the family $\mathcal{L}'$ is statistically dependent on $\boldsymbol{x}$, hereafter denoted $\boldsymbol{x} \triangledown \mathcal{L}'$. We have

$$\boldsymbol{x} \triangledown \mathcal{L}' = \bigvee_{i=1}^{|\Pi|} \left(\boldsymbol{x} \triangledown \mathcal{G}^{(i)} \cup \mathcal{A}^{(i)} \cup \mathcal{B}^{(i)} \cup \mathcal{C}^{(i)}\right). \tag{43}$$

By the uniformity property of the refreshing, we have that, given the program input $\boldsymbol{x}$, the different families of input/output shares $\left\{(a_j^{(i)})_{j=0}^{n-1}, (b_j^{(i)})_{j=0}^{n-1}, (c_j^{(i)})_{j=0}^{n-1}\right\}$ are mutually independent. We hence get

$$\Pr(\boldsymbol{x} \triangledown \mathcal{L}') \leq \sum_{i=1}^{|\Pi|} \Pr\left(\boldsymbol{x} \triangledown \mathcal{G}^{(i)} \cup \mathcal{A}^{(i)} \cup \mathcal{B}^{(i)} \cup \mathcal{C}^{(i)}\right). \tag{44}$$

We can then upper bound the probability $\Pr\left(\boldsymbol{x} \triangledown \mathcal{G}^{(i)} \cup \mathcal{A}^{(i)} \cup \mathcal{B}^{(i)} \cup \mathcal{C}^{(i)}\right)$ when the ith gadget is a secure multiplication by following the proof of Theorem 2. The only difference is that the attacker gets additional linear combinations of the input/output shares from the refreshing procedures. Specifically, we would have

- up to $\ell_1^* = 2\ell_1 + \ell_3 + 2\ell_1'$ linear combinations of the form $\sum_i \alpha_{k,i} a_i$, for $1 \leq k \leq \ell_1^*$;
- up to $\ell_2^* = 2\ell_2 + \ell_3 + 2\ell_2'$ linear combinations of the form $\sum_i \beta_{k,i} b_i$, for $1 \leq k \leq \ell_2^*$;
- up to $\ell_3^* = 2\ell_4 + 2\ell_5 + 2\ell_3'$ linear combinations of the form $\sum_{i,j} \gamma_{k,j} s_j$, for $1 \leq k \leq \ell_3^*$;

where ℓ_1', ℓ_2' and ℓ_3', are the number of leaking operations in the input/output refreshing procedures. Taking the constraint $\ell_i < \frac{n}{11}$ and $\ell_i' < \frac{n}{11}$ for every i, we still get $\ell_1^* + \ell_3^* < n$ and $\ell_2^* + \ell_3^* < n$. That is, if rmax and gmax occurs, we get

$$\Pr\left(\boldsymbol{x} \triangledown \mathcal{G}^{(i)} \cup \mathcal{A}^{(i)} \cup \mathcal{B}^{(i)} \cup \mathcal{C}^{(i)} \mid \mathsf{rmax} \wedge \mathsf{gmax}\right) \leq \frac{2n}{|\mathbb{F}|}. \tag{45}$$

For copy, addition and subtraction gadgets, the proof is quite simple. When an operation leaks in such a gadget, it reveals one shares from each input encoding. We hence get less that $\frac{n}{11}$ linear combinations on each input encoding (from the gadget leakage), plus $\frac{2n}{11}$ linear combinations on each input encoding (from their respective refreshing), plus $\frac{2n}{11}$ linear combinations on the output encoding, which can be split into independent linear combinations on the two input encodings. We clearly get less than n linear combinations on each encoding, which allows us to apply Lemma 3 and to obtain (45) for every kind of gadget.

We finally get

$$\begin{aligned}\Pr(\boldsymbol{x} \triangledown \mathcal{L}') &\leq \Pr(\neg\mathsf{rmax}) + \Pr(\neg\mathsf{gmax}) \\ &\quad + \sum_{i=1}^{|\Pi|} \Pr\left(\boldsymbol{x} \triangledown \mathcal{G}^{(i)} \cup \mathcal{A}^{(i)} \cup \mathcal{B}^{(i)} \cup \mathcal{C}^{(i)} \mid \mathsf{rmax} \wedge \mathsf{gmax}\right) \\ &\leq |\Pi| \cdot \Big(\psi\big(\frac{n}{11}, N_{\mathrm{ref}}\big) + \psi\big(\frac{n}{11}, N_{\mathrm{gad}}\big) + \frac{2n}{|\mathbb{F}|}\Big),\end{aligned}$$

which together with $N_{\mathrm{ref}}, N_{\mathrm{gad}} < 3n \log n$ concludes the proof. □

5 From Arithmetic Random Probing to Noisy Leakage

5.1 Logical Programs

The definition of a *logical program* is analogous to the definition of an arithmetic program but it is composed of logical instructions over $\{0,1\}^w$ such as the bitwise AND, OR, XOR, logical shifts and rotations, as well as the addition, subtraction, and multiplication modulo 2^w (namely typical instructions of a w-bit processor). In the ε-noisy leakage model, a logical program leaks an ε-noisy leakage function $f(\mu_j, \mu_k)$ of the pair of inputs of each logical instruction $\mu_i \leftarrow \mu_j * \mu_k$.

The security reduction of Duc *et al.* (Lemma 1) then implies that a logical program Π that is secure against δ-random-probing leakage is also secure against δ'-noisy leakage with $\delta' = \delta/2^{2w}$.

5.2 A Generic Reduction

We then have the following reduction of random-probing model for a logical programs, to the random-probing model for an arithmetic programs:

Lemma 4. *Let Π be a ε-leakage secure arithmetic program in the δ-random-probing model, then there exists a functionally equivalent logical program Π' that is ε-leakage secure in the δ'-random-probing model for some δ' satisfying*

$$\delta' = 1 - (1-\delta)^{1/N} \geq \frac{\delta}{N} \quad \textit{with} \quad N = O\Big(\frac{1}{w} \log|\mathbb{F}| \log\big(\frac{1}{w}\log|\mathbb{F}|\big)\Big). \tag{46}$$

Proof. The logical program Π' is simply the program Π where arithmetic instructions are built from several w-bit logical instructions. It is well known that the addition and subtraction on $\mathbb{F}$ can be computed in $N = O\big(\frac{1}{w}\log|\mathbb{F}|\big)$ elementary (w-bit) operations, and that the multiplication on $\mathbb{F}$ can be computed from $N = O\big(\frac{1}{w}\log|\mathbb{F}|\log\big(\frac{1}{w}\log|\mathbb{F}|\big)\big)$ elementary (w-bit) operations.

Assume that there exists an adversary $\mathcal{A}'$ with advantage ε that makes use of a δ'-random-probing leakage on Π', then we show that there exists an adversary $\mathcal{A}$ with advantage ε that makes use of a δ-random-probing leakage on Π. Since by assumption no such adversary $\mathcal{A}$ exists, then by contraposition neither does

such adversary $\mathcal{A}'$, meaning that Π' is indeed ε-leakage secure in the δ'-random-probing model.

We construct an adversary $\mathcal{A}$ that is given the full input to an arithmetic instruction of Π whenever at least one of the corresponding logical instruction leaks in Π'. Informally, it is clear that this can only increase the success probability. To make this reasoning formal, we need to construct an adversary $\mathcal{A}$ that receives the strengthened leakage, resamples it to make its distribution identical to that of the δ'-random-probing leakage on Π' and then call $\mathcal{A}'$. When $\mathcal{A}$ receives $\perp$ as leakage for an arithmetic instruction, it simply sends $\perp$ to $\mathcal{A}'$ for all the corresponding logical instructions. When it receives the full input of the arithmetic instruction (meaning that at least one corresponding logical instruction of Π' must leak), it can compute all the inputs of the corresponding logical instructions in Π', and reveal each of them to $\mathcal{A}'$ with some (biased) given probability. Since we do not consider the computational complexity of the adversaries, the easiest way to achieve a perfect simulation is to use rejection sampling. Namely, for every logical instruction in the group, the input is revealed with probability δ'. If at the end of the group, no input was revealed, simply restart the revealing process for the same group. This way, we have constructed an adversary $\mathcal{A}$ using a δ-random-probing leakage on Π where

$$\delta = 1 - (1 - \delta')^N,$$

for $N = O\big(\log|\mathbb{F}|\log\log|\mathbb{F}|\big)$. Since by assumption no such adversary exists, this means that no adversary $\mathcal{A}'$ exists with advantage ε that makes use of a δ'-random-probing leakage on Π'. □

Combining the above lemma with Lemma 1, and considering a constant word-size w, we get a tight reduction of the security in the noisy leakage model for logical program to the security in the random-probing model for arithmetic program:

Lemma 5. *Let Π be a ε-leakage secure arithmetic program in the δ-random-probing model, then there exists a functionally equivalent logical program Π' that is ε-leakage secure in the δ'-noisy leakage model for some δ' satisfying*

$$\delta' = \frac{\delta}{O(\log|\mathbb{F}|\log\log|\mathbb{F}|)}. \tag{47}$$

5.3 Application to Our Scheme

In the previous section we have shown that for $\delta = O(1/\log n)$ our construction is ε-leakage secure in the δ-random-probing model with

$$\varepsilon = \mathsf{negl}(\lambda) + \mathsf{negl}'(n) \tag{48}$$

where negl and negl' are some negligible functions and where λ is some security parameter such that $\log|\mathbb{F}| = \lambda + \log n$.

By applying the above reduction to our construction (and recalling that we have $|\mathbb{F}| = O(n)$), we obtain the following corollary of Theorem 3:

Corollary 3. *Let Π' denotes the secure logical program corresponding to our construction (see Sect. 4). Π' is ε-leakage secure in the δ-noisy leakage model where $\varepsilon = \mathsf{negl}(\lambda) + \mathsf{negl}'(n)$ and $\delta' = O\big(1/((\log n)^2 \log\log n)\big)$.*

6 Practical Aspects and Open Problems

Securing Arbitrary Computation. Although our scheme is described to work on a finite field $\mathbb{F}$ with specific structure, it can be used to secure any arbitrary computation represented as a Boolean circuit. Indeed, it is possible to embed a Boolean circuit into an arithmetic program over $\mathbb{F}$. Each bit is simply represented by an element $a \in \{0,1\} \subseteq \mathbb{F}$. The binary multiplication then matches with the $\mathbb{F}$-multiplication over this subset. Regarding the binary addition $\oplus$, it can be implemented with operations over $\mathbb{F}$ as:

$$a \oplus b = a + b - 2ab, \tag{49}$$

for every $a, b \in \{0,1\} \subseteq \mathbb{F}$. Of course such an embedding comes at a high cost in practice and our scheme would not be efficient to protect *e.g.* an AES computation. However, our scheme is asymptotically more efficient than previous ISW-based schemes meaning that there exists some masking order n for which an implementation of our scheme would be more efficient than an implementation of a previous scheme. Moreover and as discussed hereafter, we think that our scheme could be practically improved in many ways.

Practical Efficiency. For any cryptographic computation on a base field $\mathbb{F}$ with appropriate structure, our scheme should be very efficient in practice. We recall that the field should be such that $|\mathbb{F}| = \alpha \cdot n + 1$, for n being a power of 2 and α being large enough so that n/α is negligible. A 256-bit prime field such as those used in Elliptic Curve Cryptography could for instance satisfy these criteria. An interesting open issue would be to extend our scheme to work on other algebraic structures and in particular on binary fields (*e.g.* to efficiently secure the AES) or on rings used in lattice-based cryptography.

On the Size of the Field. We note that we need a 'big' field (typically of size $128 + 2\log n$) in order to have enough randomness when picking ω. However this might be a proof artefact and the scheme could be secure for some constant ω and/or using smaller fields. Another direction of improvement would be to mitigate or remove this constraint with an improved construction and/or proof technique.

Packing Encodings. Finally our scheme could also probably be improved by using the principle of packed secret sharing as suggested in [2,3] since our encoding is a kind of randomized Shamir's secret sharing.

A Number Theoretic Transform

The Number Theoretic Transform (NTT) is essentially a (Fast) Fourier Transform defined in a finite field (or ring) where inaccurate floating point or complex arithmetic can be avoided. The NTT can be used to multiply two polynomials over a finite field in quasilinear complexity. Let $\mathbb{F}_p$ be a prime finite field such that $d \mid p-1$ for some integer d ($\mathbb{F}_p$ contains d-th roots of unity) and let A be a $(d-1)$-degree polynomial over $\mathbb{F}_p[x]$ such that $A(x) = a_0 + a_1 x + a_2 x^2 + \cdots + a_{d-1} x^{d-1}$. For a given primitive d-th root of unity ξ, the NTT maps the coefficients of A to the evaluations $A(\xi^i)$ with $1 \leq i \leq d$:

$$\mathsf{NTT}_\xi : (a_0, a_1, \ldots, a_{d-1}) \mapsto (A(\xi^1), A(\xi^2), \ldots, A(\xi^d)). \tag{50}$$

For d being a power of two, the NTT can be computed in time complexity $O(d \log d)$. To show this, let us define A_0 and A_1, the two $(\frac{d}{2} - 1)$-degree polynomials

$$A_0(x) = a_0 + a_2 x + a_4 x^2 + \cdots + a_{d-2} x^{\frac{d}{2}-1}$$
$$A_1(x) = a_1 + a_3 x + a_5 x^2 + \cdots + a_{d-1} x^{\frac{d}{2}-1}$$

which satisfy

$$A(x) = A_0(x^2) + x A_1(x^2).$$

The problem of evaluating $A(x)$ at each d-th root of unity ξ^i, for $1 \leq i \leq d$, is reduced to the problem of evaluating $A_0(x)$ and $A_1(x)$ at the points ξ^{2i}, for $1 \leq i \leq \frac{d}{2}$, and we can combine the results with $A(\xi) = A_0(\xi^2) + \xi A_1(\xi^2)$. The polynomials $A_0(x)$ and $A_1(x)$ can also be evaluated at the points ξ^{2i} with the same divide and conquer strategy, using the polynomials A_{00}, A_{01}, A_{10}, A_{11} satisfying

$$A_0(x) = A_{00}(x^2) + x A_{01}(x^2) \quad \text{and} \quad A_1(x) = A_{01}(x^2) + x A_{11}(x^2).$$

This divide and conquer strategy can be iterated $\log_2(d)$ times. At the t-th step we have 2^t polynomials $A_{\boldsymbol{u}}$ of degree $\frac{d}{2^t}$ for $\boldsymbol{u} \in \{0,1\}^t$ that must be evaluated in ξ^j for $j = 2^t, 2 \cdot 2^t, \ldots, \frac{d}{2^t} \cdot 2^t$, which makes a total of $2^t \cdot \frac{d}{2^t} = d$ evaluations. Moreover, from $\xi^{j+\frac{d}{2}} = -\xi^j$ we have

$$A_{\boldsymbol{u}}(\xi^j) = A_{\boldsymbol{u}|0}(\xi^{2j}) + \xi^j A_{\boldsymbol{u}|1}(\xi^{2j}) \text{ and } A_{\boldsymbol{u}}(\xi^{j+\frac{d}{2}}) = A_{\boldsymbol{u}|0}(\xi^{2j}) - \xi^j A_{\boldsymbol{u}|1}(\xi^{2j}) \tag{51}$$

implying that the number of evaluations can be merely divided by two.

In practice, we start with $t = \log_2(d)$, where we have $2^t = d$ constant polynomials $A_{\boldsymbol{u}} = a_{\varphi(\boldsymbol{u})}$ with $\varphi(\boldsymbol{u})$ denoting the integer corresponding to the binary expansion $\boldsymbol{u} \in \{0,1\}^{\log_2(d)}$. Then we iterate (51) for t from $\log_2(d)$ down to 1 where we have our d evaluations of A. The overall process is summarized hereafter:

1. $(c_0, c_1, \ldots, c_{d-1}) \leftarrow (a_0, a_1, \ldots, a_{d-1})$
2. for $t = \log_2(d) - 1$ down to 1:
3. $\quad j = 2^t$; $k = 2^{d-t-1}$
4. $\quad$ for $i \in \bigcup_{\ell=0}^{k-1} U_{j,\ell}$
5. $\quad\quad (c_i, c_{i+j}) \leftarrow (c_i + \xi^j c_{i+j}, c_i - \xi^j c_{i+j})$

where $U_{j,\ell} = \{(2\ell j, \ldots, (2\ell+1)j - 1)\}$ and where the index shiftings of c are done modulo d, *i.e.* $c_{i+j} = c_{i+j \mod d}$. It can be checked that the above evaluation of NTT_ξ takes a total of $\frac{d \log d}{2}$ multiplications, $\frac{d \log d}{2}$ additions and $\frac{d \log d}{2}$ subtractions.

Using the NTT with a dth root of unity, we can efficiently compute the product $C(x) = A(x) \cdot B(x)$ for any two polynomials $A, B \in \mathbb{F}_p[x]$ of degree up to $n-1$ with $d = 2n$. We first apply the NTT to get d evaluations of both polynomials:

$$(A(\xi^1), A(\xi^2), \ldots, A(\xi^d)) = \mathsf{NTT}_\xi(a_0, a_1, \ldots, a_{n-1}, 0, \ldots, 0)$$
$$(B(\xi^1), B(\xi^2), \ldots, B(\xi^d)) = \mathsf{NTT}_\xi(b_0, b_1, \ldots, b_{n-1}, 0, \ldots, 0)$$

from which we get d evaluations of C by $C(\xi^i) = A(\xi^i) \cdot B(\xi^i)$ for $1 \le i \le d$. Finally, we can recover the coefficients of the output polynomial C by computing the inverse NTT on $(C(\xi), C(\xi^1), \ldots, C(\xi^d))$, which satisfies

$$\begin{aligned}(c_0, c_1, \ldots, c_d) &= \mathsf{NTT}_\xi^{-1}(C(\xi^1), C(\xi^2), \ldots, C(\xi^d)) \\ &= \mathsf{NTT}_{\xi^{-1}}\big(\frac{1}{d}C(\xi^1), \frac{1}{d}C(\xi^2), \ldots, \frac{1}{d}C(\xi^d)\big).\end{aligned}$$

References

1. Ajtai, M.: Secure computation with information leaking to an adversary. In: Fortnow, L., Vadhan, S.P. (eds.) 43rd ACM STOC, pp. 715–724. ACM Press, June 2011
2. Andrychowicz, M., Damgård, I., Dziembowski, S., Faust, S., Polychroniadou, A.: Efficient leakage resilient circuit compilers. In: Nyberg, K. (ed.) CT-RSA 2015. LNCS, vol. 9048, pp. 311–329. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-16715-2_17
3. Andrychowicz, M., Dziembowski, S., Faust, S.: Circuit compilers with $O(1/\log(n))$ leakage rate. In: Fischlin, M., Coron, J.-S. (eds.) EUROCRYPT 2016, Part II. LNCS, vol. 9666, pp. 586–615. Springer, Heidelberg (2016). https://doi.org/10.1007/978-3-662-49896-5_21
4. Battistello, A., Coron, J.-S., Prouff, E., Zeitoun, R.: Horizontal side-channel attacks and countermeasures on the ISW masking scheme. In: Gierlichs, B., Poschmann, A.Y. (eds.) CHES 2016. LNCS, vol. 9813, pp. 23–39. Springer, Heidelberg (2016). https://doi.org/10.1007/978-3-662-53140-2_2
5. Belaïd, S., Benhamouda, F., Passelègue, A., Prouff, E., Thillard, A., Vergnaud, D.: Randomness complexity of private circuits for multiplication. In: Fischlin, M., Coron, J.-S. (eds.) EUROCRYPT 2016, Part II. LNCS, vol. 9666, pp. 616–648. Springer, Heidelberg (2016). https://doi.org/10.1007/978-3-662-49896-5_22

6. Belaïd, S., Benhamouda, F., Passelègue, A., Prouff, E., Thillard, A., Vergnaud, D.: Private multiplication over finite fields. In: Katz, J., Shacham, H. (eds.) CRYPTO 2017, Part III. LNCS, vol. 10403, pp. 397–426. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-63697-9_14
7. Chari, S., Jutla, C.S., Rao, J.R., Rohatgi, P.: Towards sound approaches to counteract power-analysis attacks. In: Wiener, M. (ed.) CRYPTO 1999. LNCS, vol. 1666, pp. 398–412. Springer, Heidelberg (1999). https://doi.org/10.1007/3-540-48405-1_26
8. Chen, H., Cramer, R.: Algebraic geometric secret sharing schemes and secure multi-party computations over small fields. In: Dwork, C. (ed.) CRYPTO 2006. LNCS, vol. 4117, pp. 521–536. Springer, Heidelberg (2006). https://doi.org/10.1007/11818175_31
9. Chernoff, H.: A measure of asymptotic efficiency for tests of a hypothesis based on the sum of observations. Ann. Math. Stat. **23**(4), 493–507 (1952)
10. Coron, J.-S., Prouff, E., Rivain, M., Roche, T.: Higher-order side channel security and mask refreshing. In: Moriai, S. (ed.) FSE 2013. LNCS, vol. 8424, pp. 410–424. Springer, Heidelberg (2014). https://doi.org/10.1007/978-3-662-43933-3_21
11. Coron, J.-S., Roy, A., Vivek, S.: Fast evaluation of polynomials over binary finite fields and application to side-channel countermeasures. In: Batina, L., Robshaw, M. (eds.) CHES 2014. LNCS, vol. 8731, pp. 170–187. Springer, Heidelberg (2014). https://doi.org/10.1007/978-3-662-44709-3_10
12. Damgård, I., Ishai, Y., Krøigaard, M.: Perfectly secure multiparty computation and the computational overhead of cryptography. In: Gilbert, H. (ed.) EUROCRYPT 2010. LNCS, vol. 6110, pp. 445–465. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-13190-5_23
13. Duc, A., Dziembowski, S., Faust, S.: Unifying leakage models: from probing attacks to noisy leakage. In: Nguyen, P.Q., Oswald, E. (eds.) EUROCRYPT 2014. LNCS, vol. 8441, pp. 423–440. Springer, Heidelberg (2014). https://doi.org/10.1007/978-3-642-55220-5_24
14. Dziembowski, S., Pietrzak, K.: Leakage-resilient cryptography. In: 49th FOCS, pp. 293–302. IEEE Computer Society Press, October 2008
15. Gandolfi, K., Mourtel, C., Olivier, F.: Electromagnetic analysis: concrete results. In: Koç, Ç.K., Naccache, D., Paar, C. (eds.) CHES 2001. LNCS, vol. 2162, pp. 251–261. Springer, Heidelberg (2001). https://doi.org/10.1007/3-540-44709-1_21
16. Goldwasser, S., Rothblum, G.N.: How to compute in the presence of leakage. In: 53rd FOCS, pp. 31–40. IEEE Computer Society Press, October 2012
17. Ishai, Y., Sahai, A., Wagner, D.: Private circuits: securing hardware against probing attacks. In: Boneh, D. (ed.) CRYPTO 2003. LNCS, vol. 2729, pp. 463–481. Springer, Heidelberg (2003). https://doi.org/10.1007/978-3-540-45146-4_27
18. Kocher, P.C.: Timing attacks on implementations of Diffie-Hellman, RSA, DSS, and other systems. In: Koblitz, N. (ed.) CRYPTO 1996. LNCS, vol. 1109, pp. 104–113. Springer, Heidelberg (1996). https://doi.org/10.1007/3-540-68697-5_9
19. Kocher, P., Jaffe, J., Jun, B.: Differential power analysis. In: Wiener, M. (ed.) CRYPTO 1999. LNCS, vol. 1666, pp. 388–397. Springer, Heidelberg (1999). https://doi.org/10.1007/3-540-48405-1_25
20. Micali, S., Reyzin, L.: Physically observable cryptography (extended abstract). In: Naor, M. (ed.) TCC 2004. LNCS, vol. 2951, pp. 278–296. Springer, Heidelberg (2004). https://doi.org/10.1007/978-3-540-24638-1_16

21. Prouff, E., Rivain, M.: Masking against side-channel attacks: a formal security proof. In: Johansson, T., Nguyen, P.Q. (eds.) EUROCRYPT 2013. LNCS, vol. 7881, pp. 142–159. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-38348-9_9
22. Rivain, M., Prouff, E.: Provably secure higher-order masking of AES. In: Mangard, S., Standaert, F.-X. (eds.) CHES 2010. LNCS, vol. 6225, pp. 413–427. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-15031-9_28
23. Shamir, A.: How to share a secret. Commun. Assoc. Comput. Mach. **22**(11), 612–613 (1979)
24. Standaert, F.-X., Pereira, O., Yu, Y., Quisquater, J.-J., Yung, M., Oswald, E.: Leakage resilient cryptography in practice. Cryptology ePrint Archive, Report 2009/341 (2009). http://eprint.iacr.org/2009/341