Leakage-Resilient Cryptography over Large Finite Fields: Theory and Practice

Marcin Andrychowicz*, Daniel Masny†, Edoardo Persichetti‡

*University of Warsaw, †HGI, Ruhr-Universität Bochum, ‡Dakota State University

Abstract

Information leakage is a major concern in modern day IT-security. In fact, a malicious user is often able to extract information about private values from the computation performed on the devices. In specific settings, such as RFID, where a low computational complexity is required, it is hard to apply standard techniques to achieve resilience against this kind of attacks. In this paper, we present a framework to make cryptographic primitives based on large finite fields robust against information leakage with a bounded computational cost. The approach makes use of the inner product extractor and guarantees security in the presence of leakage in a widely accepted model. Furthermore, we show how to apply the proposed techniques to the authentication protocol Lapin, and we compare it to existing solutions.

1 Introduction

A major concern for the implementation of secure cryptographic protocols is resistance to *side-channel attacks* (SCA). This class of attacks makes use of information obtained by the observation of physical phenomena that may occur in the device used to implement the scheme. These include measurements of timings, power consumption level, running machine's sound or an electromagnetic radiation (cf. for instance [ISW03, MR04, DP08, FKPR10, GR10, DHLAW10, BKKV10, DF11, DF12, GR12, GST13]).

The technique called *masking* is a very efficient way to protect sensitive data. The idea behind masking is to split the sensitive values into d (the *masking order*) random shares and to compute every intermediate value of the algorithm on these shares. The security requirement is that each subset of $d-1$ shares is independent from the original value. In this way, in fact, an adversary would need to combine leakage samples obtained by several separate shares in order to recover useful information about the sensitive data. Multiple candidates for d-th order masking schemes have been proposed, such as *Boolean* masking [RP10] and *polynomial* masking [PR11].

Recently, an efficient way to mask the LPN-based authentication protocol Lapin [HKL$^+$12] with Boolean masking was proposed by Gaspar et al. [GLS14]. The proposal takes advantage of the linearity of the Learning Parity with Noise (LPN) assumption, on which Lapin is based. This makes it easy and therefore very efficient to apply Boolean masking to Lapin. While Boolean masking decreases the efficiency of AES quadratically in the number of shares, it decreases the efficiency only linearly in case of Lapin.

The above mentioned masking schemes, however, lack a strong formal security proof. A way to deal with this issue from a theoretical point of view was suggested by Ishai et al. [ISW03],

who proposed to use a *leakage resilient circuit compiler* based on Boolean masking. Such a compiler takes as input a certain circuit Γ and returns a modified circuit $\hat{\Gamma}$ that computes the same functionality but is designed to be resilient against a restricted class of leakage attacks. This was subsequently extended to a broader class of attacks in [FRR+10]. Solutions based on more complicated algebraic frameworks have been also proposed, for example Juma and Vahlis [JV10] and Goldwasser and Rothblum [GR10]. These solutions achieve leakage resilience against *polynomial-time computable* functions, but require a very heavy and inefficient machinery that involves public-key encryption to protect the shares.

In two independent works by Dziembowski and Faust [DF12] and again Goldwasser and Rothblum [GR12], it was shown how to achieve the same results without relying on secure encryption schemes. Both papers describe leakage-resilient compilers, which encode values on the internal wires using an inner product. The leakage resilience follows from the extractor property of the inner product as a strong extractor which builds a strong theoretical security basis. The framework has been adjusted and optimized in terms of efficiency for AES in a work by Balasch et al. [BFGV12], along with a sample implementation and an analysis of performance results. Unfortunately, the authors lose the strong theoretical security basis in favor of efficiency by using the inner product as a masking scheme but not as an extractor. Furthermore, Prouff et al. [PRR14] showed that some of their proposed algorithms to compute operations in finite fields can be attacked in theory. It is unclear yet, if these attacks can be exploited by real world SCAs.

Our Contribution. We use inner product extractor based techniques to gain leakage resilience while preserving the efficiency such that our techniques are applicable in practice. Compared to the algorithms proposed by [DF12, BFGV12, GR12] in order to perform operations on the encoded values we use non-interactive algorithms which do not use any refresh subroutine, thus improving the efficiency. Furthermore, the security of these procedures is easy to verify and does not need any leakage-free components or oracles. The drawback is that the size of the secret state will grow when using our proposed algorithms. To overcome this issue, we propose a procedure to shrink down the secret internal state. This is an interactive algorithm which uses a refresh algorithm as a subroutine. We emphasize that this shrinking procedure is optional and in many applications not necessary. A refreshing algorithm is required when a computed value is retrieved from the encodings.

The generation of leak-free randomness is a serious issue in many concrete scenarios. While [DF12, BFGV12] access leakage-free components in almost all procedures to perform operations in a finite field, we only access leakage-free components to retrieve a final value and, depending on the application, to shrink down the internal state. We also give a complete security analysis for every proposed algorithm, while, in particular for low dimension encodings together with large finite fields, the security of some of the algorithms given by [DF12, BFGV12] is not clear.

We emphasize that an inner product extractor based leakage-resilient storage is very attractive when using a finite field of an exponential size. Since even encodings with a low dimension preserve strong statistical extractor properties of the inner product. This is shown by the analyses of inner product based leakage-resilient storage of [DDV10, DF11]. Further, we improve the analysis of the inner product based leakage-resilient storage to get even stronger results.

A suitable application of our techniques are LPN- or LWE-based protocols over large fields. We will show how to perform a leakage-resilient computation of the LPN-based protocol Lapin and give implementation results. The results show that our implementation is efficient enough such that it can be considered for applications in practice.

2 Preliminaries

We write $[n]$ to indicate the set $\{1, \ldots, n\}$. We denote with $\mathbb{F}$ the finite field $\mathbb{Z}_2[x]/(g(x))$, where $g(x)$ is a degree m polynomial irreducible over $\mathbb{Z}_2[x]$ and $\mathbb{F}^* := \mathbb{F} \setminus \{0\}$. Let $A = (A_1, \ldots, A_n)$ and $B = (B_1, \ldots, B_n)$ be two vectors with elements in $\mathbb{F}$. The notation $A||B$ indicates the concatenation of the two vectors. Moreover, we denote with $A \otimes B$ the following vector of length n^2:

$$A \otimes B := (A_1B_1, \ldots, A_1B_n, A_2B_1, \ldots, A_2B_n, \ldots, A_nB_1, \ldots, A_nB_n).$$

The inner product between A and B is defined in the usual way as

$$\langle A, B \rangle := \sum_{i=1}^{n} A_i \cdot B_i.$$

If an algorithm A has oracle access to a distribution $\mathcal{D}$, we write $\mathsf{A}^{\mathcal{D}}$. A probabilistic polynomial time algorithm is called PPT.
The *statistical distance* between two random variables A and B with values in a finite set $\mathcal{X}$ is defined as $\Delta(A, B) = \frac{1}{2} \sum_{x \in \mathcal{X}} \left| \Pr[A = x] - \Pr[B = x] \right|$. If this distance is negligible, we say that the two variables are *statistically indistinguishable*. The *min-entropy* of a random variable A is defined as $H_\infty(A) = -\log(\max_{x \in \mathcal{X}} \Pr[A = x])$.

Two-Source Extractors. Two-source extractors, introduced in 1988 by Chor and Goldreich [CG88], are an important and powerful tool in cryptography.

Definition 2.1. *Let $\mathcal{L}$, $\mathcal{R}$ and $\mathcal{C}$ be finite sets, and let U be the uniform distribution over $\mathcal{C}$. A function $\mathsf{ext} : \mathcal{L} \times \mathcal{R} \to \mathcal{C}$ is a* weak (m, ϵ) two-source extractor *if for all distributions of independent random variables $L \in \mathcal{L}$ and $R \in \mathcal{R}$ such that $H_\infty(L) \geq m$ and $H_\infty(R) \geq m$ we have $\Delta(\mathsf{ext}(L, R), U) \leq \epsilon$.*

If we change the condition on the min-entropy to $H_\infty(L) + H_\infty(R) \geq k$, the extractor is called *flexible*. Note that if $k = 2m$ this requirement is weaker than the original, hence flexibility is a stronger notion.

The fact that the inner product is a strong extractor is well known in the literature ([Vaz85], [CG88]). The security results in this work are based on the following lemma regarding the inner product extractor over finite fields.

Lemma 2.1. *[Rao07, Proof of Theorem 3.1] The inner product function $\langle .,. \rangle : \mathbb{F}^n \times \mathbb{F}^n \to \mathbb{F}$ is a weak flexible (k, ϵ) two-source extractor for $\epsilon \leq 2^{((n+1)\log|\mathbb{F}|-k)/2}$.*

Limited Adversaries and Leakage-Resilient Storage. There have been several proposals to model SCA in theory [DF11, DF12, GR12]. In the so-called *split-state model*, we assume that the memory of a physical device can be split in two distinct parts, called respectively P_L and P_R. These could be, for instance, two separate processors, or also a single processor operating at distinct and separate times.

All the computation carried out on the device (for computing, for example, a cryptographic primitive or an algorithm) is performed as a two-party protocol Π between the two parties P_L and P_R. More precisely, each of the two parties has an internal state (initially just some input) and at each step communicates with the other party by sending some messages. These

messages depend on the initial state, the local randomness, and the messages received earlier in the protocol. At the end of the execution of Π, each party outputs a new state.

The main reason to adopt this setting is that we assume that the two parties operate independently, and hence are subject to completely independent leakage. In our model, we consider an adversary A that is able to interact with both memory parts. After each execution of Π, the adversary is allowed to query a *leakage oracle* $\Omega(\mathsf{view}_L, \mathsf{view}_R)$, where $(\mathsf{view}_L, \mathsf{view}_R)$ are the respective *views* of the players. The view of a player consists of all the information that was available to him during the execution of the protocol, i.e. his initial state, his local randomness and all the messages sent and/or received. The adversary submits functions f_L and f_R and after submission, he gets back $f_L(\mathsf{view}_L)$ and $f_R(\mathsf{view}_R)$. The only restriction is that the total amount of bits output by the function f_L during *one execution of the protocol* is limited to a certain constant λ, and the same holds for f_R. An adversary is called λ-*limited* with respect to the limited amount of leakage during a single execution, but an arbitrary amount of leakage over all executions of the protocol. A more formal description of the model may be found in [DF12] or [GR12].

An important primitive used to achieve leakage resilience in this model is a *leakage-resilient storage* (LRS) [DDV10, DF11, DF12]. An LRS for a set of values $\mathbb{S}$ consists of two PPT algorithms LRS := $(\mathsf{Encode}, \mathsf{Decode}, \mathsf{Refresh})$:

- $\mathsf{Encode}(1^\kappa, S) \to (L, R)$: Outputs an encoding (L, R) of a value $S \in \mathbb{S}$.
- $\mathsf{Decode}(L, R) = S$: Outputs the private value S corresponding to the encoding (L, R).

For *correctness* it is required that $\mathsf{Decode}(\mathsf{Encode}(S)) = S$ for all $S \in \mathbb{S}$.

Definition 2.2. *We say an LRS is* (λ, ϵ)-secure *if for every private value* S *and any* λ*-limited adversary* $\mathsf{A}^{\Omega(L,R)}$ *querying the functions* $f_L(L)$ *to* P_L *and* $f_R(R)$ *to* P_R *we have*

$$\Delta([f_L(L), f_R(R) \mid \mathsf{Decode}(L, R) = S], [f_L(L'), f_R(R')]) \leq \epsilon$$

where (L', R') *is an encoding of a uniformly chosen value.*

With this security notion, a λ-limited adversary cannot distinguish whether the leakage is obtained from a specific value S or a uniformly sampled value S'.

The protocol Π computes operations on encoded values and outputs encodings of the final values. These can be later retrieved with a dedicated procedure.

Remark 2.1. In our leakage model, the total amount of leakage obtained from each memory part in a single round is bounded by λ. However, after a few observations, an adversary could recover the shares completely, and trivially break the security of the scheme. The first procedure we need to define, then, is a *refreshing* procedure that allows to inject new randomness in the protocol. Namely the procedure $\mathsf{Refresh}$ takes as input an encoding (L, R) of a value S and outputs a new encoding (L', R') for S. Due to space limitations, we will leave the details and issues of the $\mathsf{Refresh}$ procedure to the appendix. We will mention, however, that all known provably-secure refreshing algorithms for two parties need a leakage-free sampling of the randomness [1]. We will discuss leakage-free oracles in Section 5.

[1] The construction of a compiler from [GR12] implies a refreshing procedure, which does not need any leak-free gates. However, it assumes that a number of parties executing the protocol is much bigger than 2 and is rather unefficient.

3 A Leakage-Resilient Storage Based on the Inner Product

An LRS based on the inner product was first proposed by [DDV10]. Given a field $\mathbb{F}$ and an integer n (the dimension of the encodings), the LRS Φ^n based on the inner product for values in $\mathbb{F}$ is given by:

- $\mathsf{Encode}(1^\kappa, S) \to (L, R)$: Sample values $(L_1, \ldots, L_n, R_1, \ldots, R_{n-1}) \xleftarrow{\$} (\mathbb{F}^*)^{2n-1}$ and set $R_n = L_n^{-1}(S - \langle L_1 \| \ldots \| L_{n-1}, R_1 \| \ldots \| R_{n-1} \rangle)$. If $R_n = 0$, resample. Finally, output $(L := L_1 \| \ldots \| L_n, R := R_1 \| \ldots \| R_n)$.
- $\mathsf{Decode}(L, R) = S$: Output $S = \langle L, R \rangle$.

Correctness and security were proved in [DF11]. However, we manage to improve the bounds for which security holds. We will present our result in the next theorem.

Theorem 3.1. *For separated P_L and P_R and a finite field $\mathbb{F}$, Φ^n is a (λ, ϵ)-secure LRS for*

$$\epsilon \leq 2^{-\frac{2n \log |\mathbb{F}^*| - (n+3) \log |\mathbb{F}| - 2\lambda}{2}}$$

Proof. Let A be a λ-limited adversary with access to oracle $\Omega(\mathsf{view}_L, \mathsf{view}_R)$. He is allowed to query $f_L(\mathsf{view}_L)$ and $f_R(\mathsf{view}_R)$ since P_L and P_R are separated. The functions f_L and f_R have joint output size 2λ. These functions define a mapping f from $(\mathbb{F}^*)^{2n}$ to $\{0,1\}^{2\lambda}$. For simplicity we will write $f(L, R)$ instead of $f_L(\mathsf{view}_L)$ and $f_R(\mathsf{view}_R)$. Let $\mathbb{P}_x$ be the set of all preimages of $x \in \{0,1\}^{2\lambda}$. Then the min-entropy of L and R given a certain leakage $x \in \{0,1\}^{2\lambda}$ is $\forall f : (\mathbb{F}^*)^{2n} \to \{0,1\}^{2\lambda}$:

$$\begin{aligned}
&H_{\infty,x}((L, R) \mid f(L, R) = x) \\
&= -\log \left(\max_{(L',R') \in (\mathbb{F}^*)^{2n}} \left(\Pr_{(L,R) \xleftarrow{\$} (\mathbb{F}^*)^{2n}} [(L, R) = (L', R') \mid f(L, R) = x] \right) \right) \\
&= -\log \left(\max_{(L',R') \in \mathbb{P}_x} \left(\Pr_{(L,R) \xleftarrow{\$} \mathbb{P}_x} [(L, R) = (L', R')] \right) \right) = \log |\mathbb{P}_x|
\end{aligned}$$

Since $f_L(\mathsf{view}_L)$ depends only on L and $f_R(\mathsf{view}_R)$ only on R, L and R are independent given f. Hence Lemma 2.1 implies the following bounds on the statistical distances for the elements of $\{0,1\}^{2\lambda}$:

$$\epsilon_x = \Delta_x([\langle L, R \rangle \mid f(L, R) = x], \langle L', R' \rangle) \leq \sqrt{|\mathbb{F}|^{n+1}} \sqrt{|\mathbb{P}_x|^{-1}}$$

for a uniform $\langle L', R' \rangle \in \mathbb{F}$. Notice that the statistical distance ϵ_x is not necessarily negligible. For instance an adversary could choose a function f such that the function is 1 if all entries of L and R are $1 \in \mathbb{F}$ and otherwise 0. In this case if a leakage $f(L, R) = x = 1$ appears, L and R are statistically fixed and $\epsilon_x = \epsilon_1 = 1$. Even if an adversary will choose such a function f, a $x = 1$ will appear only with a negligible probability then. A straight forward but a lossy technique to prove the Theorem would be: Either x appears with negligible probability or ϵ_x is negligible. We are not using this approach which is also a reason why we get better bounds.

We get the Theorem by bounding the final advantage of A: For all $S \in \mathbb{F}$

$$
\begin{aligned}
\epsilon &= \Delta([f(L,R) \mid \langle L,R\rangle = S], f(L',R')) \\
&= \frac{1}{2} \sum_{x\in\{0,1\}^{2\lambda}} |\Pr[f(L,R) = x \mid \langle L,R\rangle = S] - \Pr[f(L',R') = x]| \\
&= \frac{1}{2} \sum_{x\in\{0,1\}^{2\lambda}} \left| \frac{\Pr[\langle L,R\rangle = S \mid f(L,R) = x] \cdot \Pr[f(L',R') = x]}{\Pr[\langle L,R\rangle = S]} - \Pr[f(L',R') = x] \right| \\
&\leq \frac{1}{2}|\mathbb{F}| \sum_{x\in\{0,1\}^{2\lambda}} \Pr[f(L',R') = x] \left| \Pr[\langle L,R\rangle = S \mid f(L,R) = x] - \frac{1}{|\mathbb{F}|} \right| \\
&\leq |\mathbb{F}| \sum_{x\in\{0,1\}^{2\lambda}} \Pr[f(L',R') = x] \left(\frac{1}{2} \sum_{S'\in\mathbb{F}} \left| \Pr[\langle L,R\rangle = S' \mid f(L,R) = x] - \Pr[\langle L',R'\rangle = S'] \right| \right) \\
&= |\mathbb{F}| \sum_{x\in\{0,1\}^{2\lambda}} \Pr[f(L',R') = x] \left(\Delta_x([\langle L,R\rangle \mid f(L,R) = x], \langle L',R'\rangle) \right) \\
&\leq \frac{|\mathbb{F}|\sqrt{|\mathbb{F}|^{n+1}}}{|\mathbb{F}^*|^{2n}} \sum_{x\in\{0,1\}^{2\lambda}} \sqrt{|\mathbb{P}_x|} \leq \frac{\sqrt{|\mathbb{F}|^{n+3}} \cdot 2^{\lambda}}{|\mathbb{F}^*|^n} = 2^{-\frac{2n\log|\mathbb{F}^*| - (n+3)\log|\mathbb{F}| - 2\lambda}{2}}
\end{aligned}
$$

The first steps are straight forward. Then for the first inequality, we use a probably lossy bound. In the second last line, we sum over the probability, that a leakage x appears multiplied with the statistical distance ϵ_x implied by x. Finally we plugin the probabilities and apply the bounds on ϵ_x for all $x \in \{0,1\}^{2\lambda}$ and use Jensen's Inequality. □

Flexibility and Graceful Degradation. The LRS Φ^n satisfies two additional, very useful properties. It is *flexible*, since an adversary could query 2λ bits on a single party instead of querying λ bits on each of them, without decreasing the statistical distance. More generally, an adversary is allowed to arbitrary split the amount of leakage among the two parties, as long as the sum is equal to the total amount of tolerated leakage.
Even more interesting is the *graceful degradation* achieved by an LRS in general. If an adversary queries $2\lambda+2k$ bits instead of 2λ bits, the security will not entirely break down. In case of Φ^n, it will only increase the statistical distance from uniform by a factor of 2^k. If the statistical distance is 2^{κ} for security parameter κ, then the security parameter will be decreased to $\kappa' = \kappa - k$.

Remark 3.1. For seeing the improvement compared to previous results, we use the parameters of Lemma 1 in [DF11] which is also used in [DF12]. We set $m = 1$ and the given leakage and statistical distance is $\lambda = (1/2 - \delta)n\log|\mathbb{F}| - \log\gamma^{-1}$ and $\epsilon' = 2(|\mathbb{F}|^{3/2-n\delta} + |\mathbb{F}|\gamma)$ for $\gamma > 0$ and $1/2 > \delta > 0$. If we plug in λ in Theorem 3.1, our bound yields $\epsilon = |\mathbb{F}^*|^{-n}|\mathbb{F}|^{n+3/2-n\delta}\gamma \approx |\mathbb{F}|^{3/2-n\delta}\gamma$ for large fields. Hence $\epsilon' > \epsilon$.

Remark 3.2. Further, for a total leakage 2λ of $1/2$ of the bits of the encodings or more, security is not guranateed anymore. This follows from the fact that $(n+3)\log|\mathbb{F}|$ is larger than $n\log|\mathbb{F}^*|$ which is the entropy of one of the encodings.

4 Computation and Retrieving Computed Values

To begin, we show how to perform non-interactive operations on the encoded values. Non-interactivity guarantees that the computation doesn't contradict the split-state model's assumptions, thus ensuring to achieve security. After describing the non-interactive operations, we give a more formal description of a set of leakage-resilient operations based on the LRS Φ^n.

Addition of a Constant and an Encoded Value. Let $X = \langle L, R\rangle$ be the input secret value and $c \in \mathbb{F}$ be a constant. To compute $c + X$, we set $L' = L||c$ and $R' = R||1$. Then

$$\langle L', R'\rangle = \sum_{i=1}^{n}(L_i \cdot R_i) + c = X + c.$$

Addition of two Encoded Values. Let $X = \langle L, R\rangle$ and $Y = \langle K, Q\rangle$ be the input secret values, and (L', R') the encoding for $Z = X + Y$. The simplest addition procedure is to set $L' = L||K$ and $R' = R||Q$. It is trivial to verify that

$$\langle L', R'\rangle = \sum_{i=1}^{n}(L_i \cdot R_i + K_i \cdot Q_i) = \sum_{i=1}^{n}(L_i \cdot R_i) + \sum_{i=1}^{n}(K_i \cdot Q_i) = \langle L, R\rangle + \langle K, Q\rangle.$$

Multiplication of an Encoded Value by a Constant. Let c be a public constant and let $X = \langle L, R\rangle$ be the input secret value. We would like to obtain shares (L', R') for $c \cdot X$. It is then enough to set $L' = L$ and $R'_i = c \cdot R_i$ for $i \in [n]$. It is immediate to verify that

$$\langle L', R'\rangle = \sum_{i=1}^{n}(L_i \cdot c \cdot R_i) = c \cdot \langle L, R\rangle = c \cdot X.$$

Multiplication of two Encoded Values. Let $X = \langle L, R\rangle$ and $Y = \langle K, Q\rangle$ be the input secret values and (L', R') the encoding for $Z = X \cdot Y$. The simplest multiplication procedure is to set $L' = L \otimes K$ and $R' = R \otimes Q$. It is now easy to verify that

$$\langle L', R'\rangle = \sum_{i=1}^{n}\sum_{j=1}^{n}(L_i \cdot K_j \cdot R_i \cdot Q_j) = \sum_{i=1}^{n}(L_i \cdot R_i) \cdot \sum_{i=1}^{n}(R_i \cdot Q_i) = \langle L, R\rangle \cdot \langle K, Q\rangle.$$

We emphasize that this operation is too costly for large dimensions. If a multiplication between two encoded values is necessary, using the algorithm given by [DF12] should be considered.

A Set of Leakage-Resilient Operations. To describe the set of leakage-resilient operations, we use again the algorithms of Φ^n. More precisely, the set of leakage-resilient operations Ψ^n consists of nine PPT algorithms for two parties P_L and P_R:

- $\mathsf{Initialize}(S_1, \ldots, S_s)$: For all $i \in [s]$ compute $\mathsf{Encode}_{\Phi^n}(1^\kappa, S_i) \to (L_i, R_i)$. Start P_L with input $L_1, \ldots L_s$ and P_R with input $R_1, \ldots, R_s$.
- $\mathsf{Refresh}(i)$: P_L and P_R replace (L_i, R_i) by $(L'_i, R'_i) \leftarrow \mathsf{Refresh}(L_i, R_i)$.
- $\mathsf{cAdd}(i, j, c)$: P_L sets $L_i := L_j \| c$ and P_R sets $R_i := R_j \| 1$.
- $\mathsf{Add}(i, j, k)$: P_L sets $L_i := L_j \| L_k$ and P_R sets $R_i := R_j \| R_k$.
- $\mathsf{cMult}(i, j, c)$: P_L sets $L_i := (cL_{j,1} \| cL_{j,2} \| \ldots)$ for $L_j = (L_{j,1} \| L_{j,2} \| \ldots)$ and P_R sets $R_i := R_j$.
- $\mathsf{Mult}(i, j, k)$: P_L sets $L_i := L_j \otimes L_k$ and P_R sets $R_i := R_j \otimes R_k$.
- $\mathsf{RetrieveValue}(i) \to (L', R')$: Invoke $\mathsf{Refresh}(i)$, P_L outputs L_i and P_R outputs R_i.
- $\mathsf{ShrinkDown}(i)$: Shrinks down L_i and R_i to dimension $n + 1$. For more details and the security analysis, we refer to Appendix B.

Remark 4.1. Note that, apart from cMult, the length of the encodings increases in all the other operations. This can influence the performance of the following operations. Thus, we have designed a Shrink procedure that allows to reduce an arbitrary length of encodings down to $n+1$ field elements.
It turns out that, in the protocols we considered, using this operation does not improve the overall efficiency. This is because it requires a call to the Refresh procedure, which is quite costly. For completeness, we present the Shrink operation in Appendix B. We remark that this operation is still useful in many situations, because it does improve the performance for more complicated patterns of operations (indeed, even for just two consecutive multiplications on encoded values).

The main property of Ψ^n is that functions computable by two parties P_L and P_R with the operations described above can be made leakage resilient in a straightforward way. The procedure Initialize, which receives as input all sensitive values, is called at the beginning of the computation. This process has to be free of leakage. Once encodings for the sensitive values are created and shared among P_L and P_R, arbitrary functions can be computed and retrieved and the leakage during the computation will not leak any information about the sensitive values, even if the computed function may reveal them.

After the computation, P_L and P_R can refresh their encodings by using Refresh to compute another function without leaking information about the sensitive values during the computation. If Refresh is used, the amount of tolerated leakage is as large as during the first computation. This follows directly from the property of Refresh. We prove the general statement about Ψ^n in the next theorem.

Theorem 4.1. *Let F be an arbitrary function computable by two parties P_L, P_R using Ψ^n. Let the encodings used by P_L, P_R for computing a value be fresh and independent. Let $S_1, \ldots, S_s \in \mathbb{F}$ be a set of input values for F among additional inputs that may be chosen uniformly or by an adversary. Then for any λ-limited adversary A and any $q \in \mathbb{N}$:*

$$\Delta(\mathsf{A}^{\Omega(\mathbb{P}_L,\mathbb{P}_R)}(x_1,\ldots x_q), \mathsf{A}^{\Omega(\mathbb{P}_U,\mathbb{P}_U)}(x_1,\ldots x_q)) \leq q2^{-\frac{2n\log|\mathbb{F}^*|-(n+3)\log|\mathbb{F}|-2\lambda}{2}}$$

where x_i is an output of F on input $S_1, \ldots, S_s$. Furthermore, for every $i \in [q]$, $\Omega(\mathbb{P}_L, \mathbb{P}_R)$ gives access to λ bits of leakage on each of the views of P_L and P_R during the computation of x_i, whereas $\Omega(\mathbb{P}_U, \mathbb{P}_U)$ indicates leakage obtained from the computation of x_i for uniform $S'_1, \ldots, S'_s \in \mathbb{F}$.

Proof. We start with $q = 1$. Without loss of generality we set $x_1 = \{S_1, \ldots S_s\}$ and assume that A sends queries $f_{L,1}(L_{S_1,1}), \ldots, f_{L,s}(L_{S_s,1})$ to P_L and $f_{R,1}(R_{S_1,1}), \ldots, f_{R,s}(R_{S_s,1})$ to P_R with a total ouput size of 2λ bits. Let λ_i be the output size of $f_{L,1}(L_{S_i,1})$ and $f_{R,1}(R_{S_i,1})$ for $i \in [s]$. Then according to Theorem 3.1:

$$\begin{aligned}
\epsilon &= \Delta(\mathsf{A}^{\Omega(\mathbb{P}_L,\mathbb{P}_R)}(x_1), \mathsf{A}^{\Omega(\mathbb{P}_U,\mathbb{P}_U)}(x_1)) \\
&= \Delta(\mathsf{A}^{\Omega(\mathbb{P}_L,\mathbb{P}_R)}(S_1,\ldots,S_s), \mathsf{A}^{\Omega(\mathbb{P}_U,\mathbb{P}_U)}(S_1,\ldots,S_s)) \\
&\leq \sum_{i=1}^{s} 2^{-\frac{2n\log|\mathbb{F}^*|-(n+3)\log|\mathbb{F}|-\lambda_i}{2}} \\
&= 2^{-\frac{2n\log|\mathbb{F}^*|-(n+3)\log|\mathbb{F}|}{2}} \sum_{i=1}^{s} 2^{\frac{\lambda_i}{2}} \\
&\leq 2^{-\frac{2n\log|\mathbb{F}^*|-(n+3)\log|\mathbb{F}|-2\lambda}{2}}
\end{aligned}$$

This is because Theorem 3.1 holds for any private value $S \in \mathbb{F}$, which is harder to achieve than if S is known or even chosen by A. To extend the result to q outputs of F, we use a simple hybrid argument. For x_1, we showed that A can not distinguish if the leakage is received from encodings of $S_1, \dots S_s$ or from some uniform $S'_1, \dots S'_s$ with probability more than ϵ. Since we use fresh and independent encodings of $S_1, \dots S_s$ for the computation of x_2 to x_q, we can apply Theorem 3.1 again. So for every single x_i, A will notice with at most probability ϵ, if the leakage is based on $S'_1, \dots S'_s$ instead of $S_1, \dots S_s$. Summing up over q we get:

$$\Delta(\mathsf{A}^{\Omega(\mathbb{P}_L,\mathbb{P}_R)}(x_1, \dots x_q), \mathsf{A}^{\Omega(\mathbb{P}_U,\mathbb{P}_U)}(x_1, \dots x_q)) \leq q\epsilon.$$

□

Note that Theorem 4.1 provides leakage resilience for any function F with private values $\mathbb{S}$ and computable by two parties P_L, P_R using Ψ^n. More precisely, given q outputs of F and leakage retrieved during the computation of F, an adversary cannot distinguish if the leakage comes from the computation of F on input $\mathbb{S}$ or a uniformly sampled input in $\mathbb{F}$.

Corollary 4.1. *Let F be a function with private input $\mathbb{S}$ and additional input that may be chosen at uniform or by an adversary. Suppose that, for any PPT algorithm, q outputs of F are distinguishable from uniform with probability at most ϵ. Then q outputs of F computed by two parties P_L, P_R using Ψ^n are distinguishable from uniform with probability at most ϵ' by any PPT λ-limited adversary, where*

$$\epsilon' \leq \epsilon + q2^{-\frac{2n \log |\mathbb{F}^*| - (n+3) \log |\mathbb{F}| - 2\lambda}{2}}.$$

5 Leakage-Resilient Computation of Lapin

Even though the techniques presented above can be easily applied to other primitives or protocols (for example [LM13]), we set our focus on Lapin. The instantiation of Lapin with a large field fits perfectly the proposed techniques. We use the parameters given in [HKL$^+$12]. The authors propose to use the field $\mathbb{F} = \mathbb{F}_2[X]/(X^{532}+X+1)$, which results in a size $|\mathbb{F}| = 2^{532}$. Lapin uses two private key elements $s_1, s_2 \in \mathbb{F}$ and for every protocol execution, a sensitive noise term e is sampled from the distribution $\mathcal{B}^{\mathbb{F}}_\tau$, i.e. the distribution over the polynomials of $\mathbb{F}$ where each of the coefficients is chosen from the binary Bernoulli distribution. While s_1 and s_2 could be stored in encoded form on two separated parts P_L and P_R on the device, e has to be resampled after every computation and not just refreshed. During the protocol a term $z = r(cs_1 + s_2) + e$ for uniform field elements r, c is computed. Due to space constraintments, we refer for details to [HKL$^+$12]. A leakage-resilient computation of z would imply a leakage-resilient variant of Lapin.

On Leak-Free Oracles. For sampling and encoding e, we use a *leak-free* oracle $\mathcal{O}_e$. The reason for using $\mathcal{O}_e$ to generate an encoding for e is that it is fundamental to securely sample the randomness. In fact, even leaking a single bit of the sampled noise is enough to undermine security, since revealing the noise from a LPN sample provides a linear equation from which the secret can be recovered. Hence we assume that an encoding of the random noise is computed in a leak-free way. This may be not reasonable to assume in some situations. On the other side, the $\mathcal{O}_e$ oracle does not have any input, and the noise e is independent from any interaction between the parties of the authentication protocol, this makes it harder to attack such an oracle with a SCA.

One strategy to deal with this issue (that also concerns refreshing procedures), is to sample the vectors L_e and R_e in advance, i.e. even before the challenge c is known. One can therefore

compute a number of pairs $(L_{e_1}, R_{e_1}), (L_{e_2}, R_{e_2}), \ldots$ and pick one of them (possibly at random) whenever a fresh pair is needed. Storing these pairs on the Tag even for a long time is completely safe under the assumption that only computation leaks information. Even if an adversary got access to a stored pair, the scheme would still be secure as long as the adversary did not learn more than what he could have learned via leakage queries during a single execution of the protocol. Whenever a Tag is running out of (L_e, R_e) pairs, it could sample a few new pairs from $\mathcal{O}_e$ and store them in the memory or sample a new pair after every protocol execution. Even if the oracle $\mathcal{O}_e$ was not completely leakage-free, it would still be hard to attack the system, since the (L_e, R_e) pairs are sampled in a different moment from the actual execution of the protocol and it is probably not easy for an adversary to figure out which pair is used next time.[2]

Describing the Leakage-Resilient Computation. At the core of Lapin, there is the function $F(r, c, s_1, s_2, e) = z = r(s_1c + s_2) + e = rcs_1 + rs_2 + e$. In Figure 1 we give the details of its implementation using the set of leakage-resilient operations Ψ^n from Section 4.

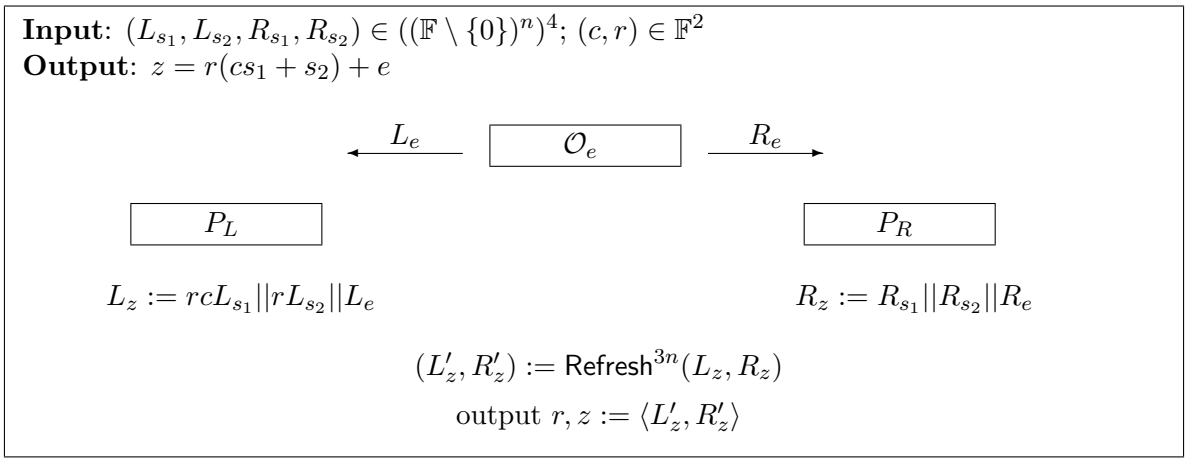

Figure 1: **Leakage Resilient Computation for a Lapin Tag.** To see which instructions of Ψ^n are used, see Section 4. For the encodings hold $\langle L_{s_1}, R_{s_1} \rangle = s_1$, $\langle L_{s_1}, R_{s_1} \rangle = s_2$ and $\langle L_e, R_e \rangle = e$. Before perfoming the next computation, the encodings of s_1 and s_2 need to be refreshed.

The encodings $L_{s_1}, L_{s_2}, R_{s_1}, R_{s_2}$ for s_1 and s_2 are stored on the device and e is obtained from $\mathcal{O}_e$. The two parties P_L and P_R perform non-interactive additions of shares and multiplications by constants to create an encoding of the response z. The retrieving procedure is used to get an encoding of z in a secure way. Finally, z itself can be obtained by computing the inner product of the encodings. Before starting the next protocol execution, the encodings of s_1 and s_2 need to be refreshed using the refreshing operation of Ψ^n.
The security of the scheme and robustness against leakage can be easily obtained from Corollary 4.1. Let ϵ_L be the winning probability against Lapin. This is essentially the probability of distinguishing, for q outputs, the function $F(r, c, s_1, s_2, e) = z$ from uniform, where r is uniform and c is chosen by an adversary. The values s_1, s_2 and e are the sensitive values and hence they are encoded. The winning probability ϵ_p against the proposed leakage-resilient protocol for q executions is $\epsilon_p = \epsilon_L + \epsilon_{\Psi^n}$, where ϵ_{Ψ^n} is the distinguishing probability stated in Theorem 4.1.

Sampling the Randomness and Refreshing. As we already mentioned, it is necessary that both the on-chip randomness sampling and the refreshing procedure be secure against continual leakage. In particular, if the refreshing procedure accesses a sensitive value in order to generate

[2]Because the pair to be used can be picked at random from the set of available pairs.

new encodings for it, the overall security of the protocol could be critically harmed. The sensitive value could in fact be easily retrieved during refresh executions. In Appendix A we describe two existing refreshing algorithms for inner product shares. Neither of them directly accesses a sensitive value so both perform much better, in the presence of leakage, than simply executing an Decode operation followed by a new Encode operation. While the weaker refreshing algorithm is not provably secure in a theoretical sense, the stronger, leakage-resilient refreshing procedure comes at a cost of a less efficient computation and requires a larger amount of randomness. Note that even the leakage-resilient refreshing requires that the randomness is drawn from a leakage-free oracle.

Efficiency. The efficiency of the scheme is calculated in terms of inversions and multiplications over $\mathbb{F}$. In Table 1 we report our efficiency analysis of Lapin when instantiated with the stronger (second row) and the weaker (third row) refreshing procedures. In our analysis, we do not include the computation of a refreshing procedure between two protocol executions.

Protocol	Refresh	n	Efficiency		Security	
			Multiplications & Invertions	8 bit AVR	λ	ϵ_p
Lapin	-	-	2 & 0	0.3 mio cycles	0	ϵ_L
Lapin	Leakage-Resilient	4	$19n$ & $6n+1$	43 mio cycles	141	$\epsilon_L + 2^{-81}$
Lapin	Leakage-Free	4	$11n+1$ & 1	9 mio cycles	141	$\epsilon_L + 2^{-81}$

Table 1: **Efficiency of the Framework and Robustness Against Leakage**. In the table above, n is the dimension of the encodings, ϵ_L is the winning probability against Lapin and ϵ_p is the winning probability against the leakage-resilient protocol with λ bits of leakage on each of the two parties per protocol execution. The refresh procedure in between two protocol executions is not covered in the presented computational costs. The 8 bit AVR implementation for multiplication and division is a straight forward implementation of the algorithms given in [HVM04] and for Lapin a uniform challenge c in $\mathbb{F}$ is used instead of a sparse element in $\mathbb{F}$.

Even though the protocol is quite simple, the computation is perhaps more expensive than one would expect, due to the expensive refreshing operation (which we describe in Appendix A). Compared to standard Lapin, the efficiency decreases by at least a factor of 30. Lapin performs better over a ring with a reducible multiplication, but in order to apply the proposed techniques, the extractor properties of a field are necessary. Furthermore, Lapin takes advantage of a multiplication with sparse field elements. In our framework, only a few field elements are sparse and hence the optimization does not have a big effect on the overall efficiency.

The 8 bit AVR implementation is based on a shift and add based division and multiplication. Even the most costly implementation with 43 million cycles has a running time of 1.34 seconds on a 32 Mhz architecture. The cycle amount would drastically decrease on an implementation on a 32 bit architecture, since shifts and additions can be carried out four times faster. We emphasize, that the cost of sampling the randomness is not covered here.

Leakage Resilience. Our proposal accomplishes leakage resilience in a model which allows continuous and arbitrarily chosen leakage functions as long as leakage-free components are not addressed. A choice of $n = 4$ results in a leakage-resilient protocol for chosen leakage functions of 141 bits output size per round for each of the two parties. To get these results, we first set the statistical distance gained by the inner product to 2^{-81}. For meaningful results, Theorem 4.1 requires $n \geq 4$. Finally we set the amount of protocol executions to be at most $q = 2^{40}$.

6 Conclusions and Future Work

This work provides techniques to perform leakage-resilient operations which perfectly fits cryptographic primitives or protocols running over large finite fields. It achieves strong provable security results thanks to the improved results for the underlying LRS based on the inner product extractor and the large size of the field. This framework could be very helpful to make other primitives leakage-resilient without using heavy machinery. Since the known refresh algorithms are still costly, more efficient alternatives would greatly increase the overall efficiency.

An issue from which our techniques suffer is the generation of on-chip randomness. Furthermore, it is required to use leakage-free oracles to sample randomness without leaking information.

Applying the proposed techniques to Lapin, we obtain a very high level of leakage resilience. In terms of efficiency, it is still very expensive, decreasing the efficiency compared to standard Lapin by at least a factor of 30. This is also a drawback for leakage resilience, since additional computation will cause additional leakage. Therefore, in settings in which performance is very important and leakage resilience plays a minor role, the Boolean masking of Lapin seems to be a better choice. On the other hand, in applications in which a high leakage resilience is necessary, the proposed techniques applied to Lapin provides an interesting option while still having reasonable responding times during a protocol interaction.

7 Acknowledgements

The authors would like to thank Krzysztof Pietrzak and Eike Kiltz for the helpful discussions on the leakage resilience of LPN and Tim Güneysu, Thomas Pöppelmann and Ingo von Maurich for helping with the implementation on the avr microcontroller.

References

[And12] Marcin Andrychowicz. Efficient refreshing protocol for leakage-resilient storage based on the inner-product extractor. *CoRR*, abs/1209.4820, 2012. 15, 16

[BFGV12] Josep Balasch, Sebastian Faust, Benedikt Gierlichs, and Ingrid Verbauwhede. Theory and practice of a leakage resilient masking scheme. In Xiaoyun Wang and Kazue Sako, editors, *ASIACRYPT*, volume 7658 of *Lecture Notes in Computer Science*, pages 758–775. Springer, 2012. 2, 14

[BKKV10] Zvika Brakerski, Yael Tauman Kalai, Jonathan Katz, and Vinod Vaikuntanathan. Overcoming the hole in the bucket: Public-key cryptography resilient to continual memory leakage. In *FOCS*, pages 501–510. IEEE Computer Society, 2010. 1

[CG88] Benny Chor and Oded Goldreich. Unbiased bits from sources of weak randomness and probabilistic communication complexity. *SIAM Journal on Computing*, 17(2):230–261, 1988. 3

[DDV10] Francesco Davì, Stefan Dziembowski, and Daniele Venturi. Leakage-resilient storage. In Juan A. Garay and Roberto De Prisco, editors, *SCN*, volume 6280 of *Lecture Notes in Computer Science*, pages 121–137. Springer, 2010. 2, 4, 5

[DF11] Stefan Dziembowski and Sebastian Faust. Leakage-resilient cryptography from the inner-product extractor. In Dong Hoon Lee and Xiaoyun Wang, editors, *ASIACRYPT*, volume 7073 of *Lecture Notes in Computer Science*, pages 702–721. Springer, 2011. 1, 2, 3, 4, 5, 6, 14

[DF12] Stefan Dziembowski and Sebastian Faust. Leakage-resilient circuits without computational assumptions. In Ronald Cramer, editor, *TCC*, volume 7194 of *Lecture Notes in Computer Science*, pages 230–247. Springer, 2012. 1, 2, 3, 4, 6, 7, 14, 16

[DHLAW10] Yevgeniy Dodis, Kristiyan Haralambiev, Adriana López-Alt, and Daniel Wichs. Cryptography against continuous memory attacks. In *FOCS*, pages 511–520. IEEE Computer Society, 2010. 1

[DP08] Stefan Dziembowski and Krzysztof Pietrzak. Leakage-resilient cryptography. In *FOCS '08: Proceedings of the 49th Annual IEEE Symposium on Foundations of Computer Science*, Washington, DC, USA, 2008. IEEE Computer Society. 1

[FKPR10] Sebastian Faust, Eike Kiltz, Krzysztof Pietrzak, and Guy N. Rothblum. Leakage-resilient signatures. In Daniele Micciancio, editor, *Theory of Cryptography, 7th Theory of Cryptography Conference, TCC 2010, Zurich, Switzerland, February 9-11, 2010. Proceedings*, volume 5978 of *Lecture Notes in Computer Science*, pages 343–360. Springer, 2010. 1

[FRR+10] Sebastian Faust, Tal Rabin, Leonid Reyzin, Eran Tromer, and Vinod Vaikuntanathan. Protecting circuits from leakage: the computationally-bounded and noisy cases. In Henri Gilbert, editor, *EUROCRYPT*, volume 6110 of *Lecture Notes in Computer Science*, pages 135–156. Springer, 2010. 2, 14

[GLS14] Lubos Gaspar, Gaëtan Leurent, and François-Xavier Standaert. Hardware Implementation and Side-Channel Analysis of Lapin. In Josh Benaloh, editor, *CT-RSA 2014*, San Francisco, États-Unis, February 2014. 1

[GR10] Shafi Goldwasser and Guy N. Rothblum. Securing computation against continuous leakage. In Tal Rabin, editor, *Advances in Cryptology - CRYPTO 2010, 30th Annual Cryptology Conference, Santa Barbara, CA, USA, August 15-19, 2010. Proceedings*, volume 6223 of *Lecture Notes in Computer Science*, pages 59–79. Springer, 2010. 1, 2

[GR12] Shafi Goldwasser and Guy N. Rothblum. How to compute in the presence of leakage. In *FOCS*, pages 31–40. IEEE Computer Society, 2012. 1, 2, 3, 4

[GST13] Daniel Genkin, Adi Shamir, and Eran Tromer. Rsa key extraction via low-bandwidth acoustic cryptanalysis. Cryptology ePrint Archive, Report 2013/857, 2013. `http://eprint.iacr.org/`. 1

[HKL+12] S. Heyse, E. Kiltz, V. Lyubashevsky, C. Paar, and K. Pietrzak. Lapin: An efficient authentication protocol based on ring-lpn. In *Fast Software Encryption*, pages 346–365. Springer, 2012. 1, 9

[HVM04] Darrel Hankerson, Scott Vanstone, and Alfred J Menezes. *Guide to elliptic curve cryptography*. Springer, 2004. 11

[ISW03] Yuval Ishai, Amit Sahai, and David Wagner. Private circuits: Securing hardware against probing attacks. In Dan Boneh, editor, *CRYPTO*, volume 2729 of *Lecture Notes in Computer Science*, pages 463–481. Springer, 2003. 1

[JV10] Ali Juma and Yevgeniy Vahlis. Protecting cryptographic keys against continual leakage. In Tal Rabin, editor, *CRYPTO*, volume 6223 of *Lecture Notes in Computer Science*, pages 41–58. Springer, 2010. 2

[LM13] Vadim Lyubashevsky and Daniel Masny. Man-in-the-middle secure authentication schemes from lpn and weak prfs. In Ran Canetti and JuanA. Garay, editors, *Advances in Cryptology CRYPTO 2013*, volume 8043 of *Lecture Notes in Computer Science*, pages 308–325. Springer Berlin Heidelberg, 2013. 9

[MR04] Silvio Micali and Leonid Reyzin. Physically observable cryptography (extended abstract). In Moni Naor, editor, *TCC*, volume 2951 of *Lecture Notes in Computer Science*, pages 278–296. Springer, 2004. 1

[PR11] Emmanuel Prouff and Thomas Roche. Higher-order glitches free implementation of the aes using secure multi-party computation protocols. In Bart Preneel and Tsuyoshi Takagi, editors, *CHES*, volume 6917 of *Lecture Notes in Computer Science*, pages 63–78. Springer, 2011. 1

[PRR14] Emmanuel Prouff, Matthieu Rivain, and Thomas Roche. On the practical security of a leakage resilient masking scheme. pages 169–182, 2014. 2

[Rao07] Anup Rao. An exposition of bourgains 2-source extractor. In *Electronic Colloquium on Computational Complexity (ECCC)*, volume 14, 2007. 3

[RP10] Matthieu Rivain and Emmanuel Prouff. Provably secure higher-order masking of aes. In Stefan Mangard and François-Xavier Standaert, editors, *CHES*, volume 6225 of *Lecture Notes in Computer Science*, pages 413–427. Springer, 2010. 1

[Vaz85] Umesh V Vazirani. Towards a strong communication complexity theory or generating quasi-random sequences from two communicating slightly-random sources. In *Proceedings of the seventeenth annual ACM symposium on Theory of computing*, pages 366–378. ACM, 1985. 3

A Refreshing Procedures for the Inner Product LRS

As a first security requirement, a refreshing procedure needs to be *rerandomizing.*

Definition A.1 (Rerandomizing)**.** *The refreshed encodings are uniformly distributed over the set of encodings of the encoded value.*

Dziembowski and Faust in [DF11] describe two possible refreshing procedures, starting from an intuitive, but flawed, one, and then providing a secure one. The latter makes use of a leak-free component $\mathcal{O}_R$ that samples uniformly random pairs of orthogonal vectors, and has a complexity of $O(n^2)$ field operations. An improved version appears in [DF12]. The procedure was then revisited and adapted to the AES case in [BFGV12]. We report it in Figure 2.

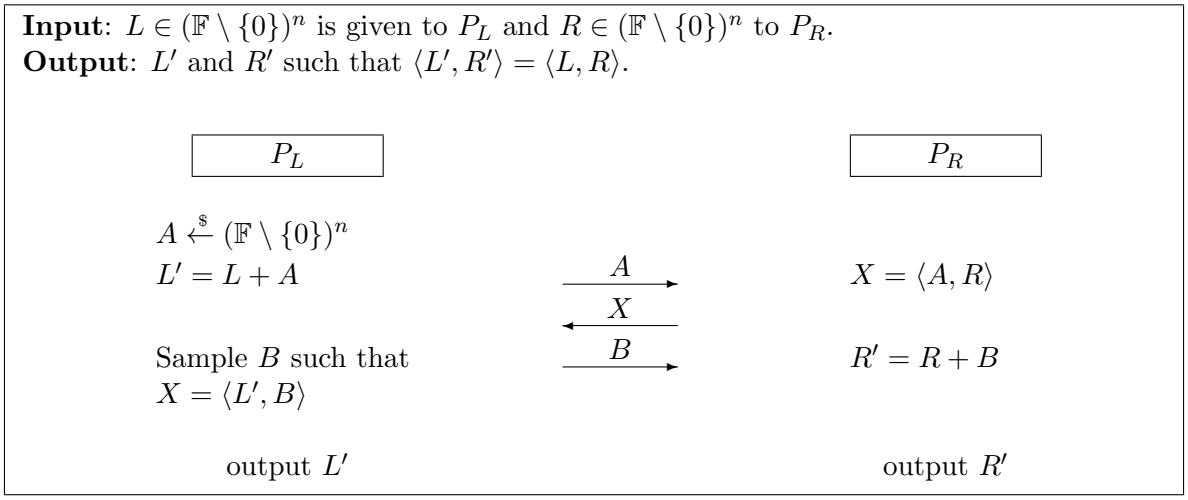

Figure 2: **Refreshing Procedure.** The refreshing procedure proposed in [BFGV12].

This formulation of a refreshing procedure is very simple but, as the authors incidentally mention, security is based on the (rather unrealistic) assumption that the whole procedure is leakage-free. The reason for this is that, during the interaction between P_L and P_R, one of the parties might learn additional information about the secret state of the other one. While leakage on input and output does not cause any problem, an adversary could use this additional knowledge of one of the parties during the procedure to query a leakage function which depends partially on both the encodings. This might reveal information about the inner product of the encodings and hence of the encoded value. Even though in practice, it is not known yet, how to exploit this by a SCA.

To deal with this issue, a property called *reconstructability* was introduced in [FRR+10]. Let Op be a masked operation with input (L, R), and output (L', R'). We call *reconstructor* a simulator algorithm Rec that is able to recreate the views that both parties would have after executing Op, without actually executing it. More specifically, Rec takes as input (L, R) and (L', R'), and returns $(\mathsf{view}_L, \mathsf{view}_R)$. In addition, it is important that the execution of Rec does not require any interaction between the parties after they are given the input.[3]

[3]Therefore, the parties can jointly draw some common randomness in advance. This will be referred to as *offline sampling* later in this paper.

Definition A.2 (Reconstructability)**.** *A masked operation* Op *is said to be* ϵ*-reconstructable if there exists a reconstructor* Rec *such that, for every* $X \in \mathbb{F}$*, it holds that*

$$\Delta((L', R', \mathsf{view}_L, \mathsf{view}_R), (L', R', \mathsf{view}'_L, \mathsf{view}'_R)) \leq \epsilon,$$

where $(L, R) = \mathsf{Encode}(X)$*,* view_L *and* view_R *are the views of the two parties after the execution of* $\mathsf{Op}(L, R) = (L', R')$ *and* $(\mathsf{view}'_L, \mathsf{view}'_R) = \mathsf{Rec}((L, R), (L', R'))$*.*

This property guarantees that leaking from the internal states during the operation on the encodings does not reveal more than just leaking from the input and output of the operation.

A reconstructable refreshing procedure was suggested by Andrychowicz in [And12] and we present it in Figure 3.

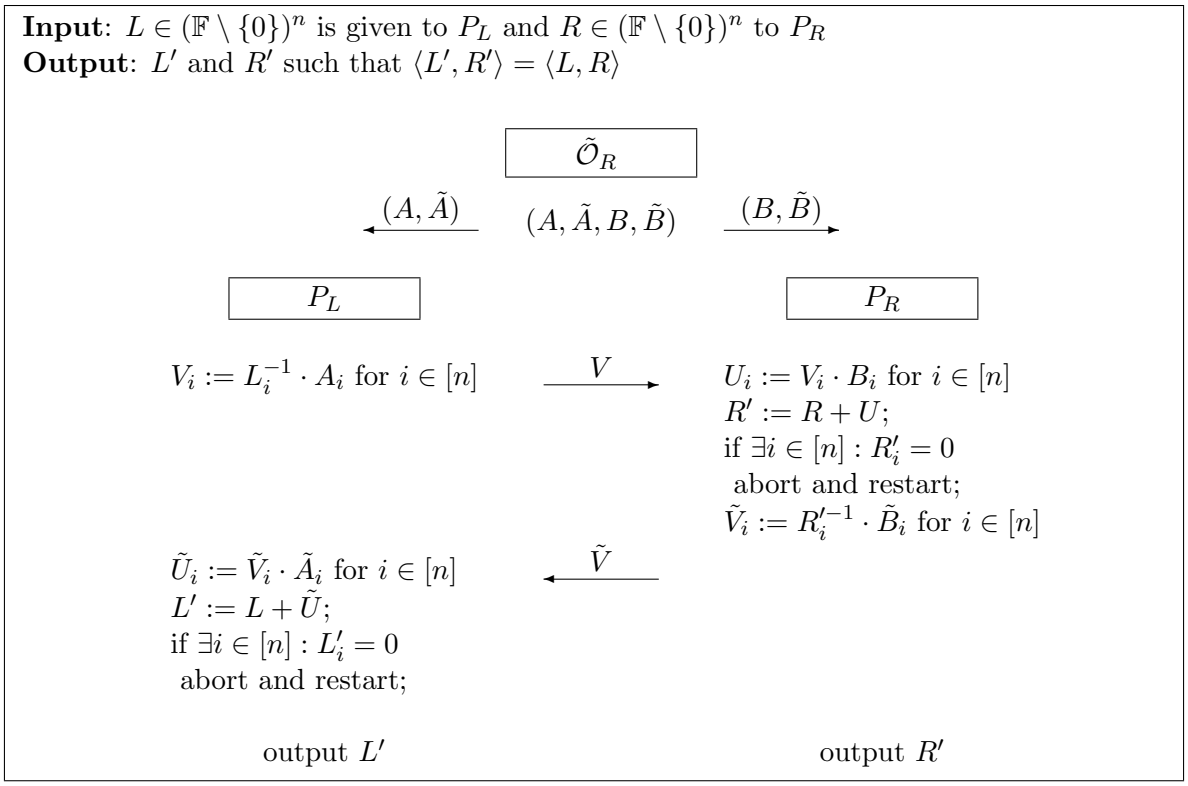

Figure 3: **Refreshing Procedure.** The procedure $\mathsf{Refresh}^n$ is used to refresh the shares of a secret. The values $A, \tilde{A}, B, \tilde{B}$ are such that $\langle A, B \rangle = -\langle \tilde{A}, \tilde{B} \rangle$ and $A_i \neq 0$ and $\tilde{B}_i \neq 0$ for $1 \leq i \leq n$.

As opposed to previous proposals, this procedure is more efficient, having a complexity of $O(n)$ operations: it requires $2n$ inversions, $4n$ multiplications and $2n$ additions in the finite field. The procedure makes use of a modified leak-free component $\tilde{\mathcal{O}}_R$ that generates quadruples of vectors $(A, \tilde{A}, B, \tilde{B})$ such that $\langle A, B \rangle = -\langle \tilde{A}, \tilde{B} \rangle$ and for $1 \leq i \leq n$ it holds that $A_i \neq 0$ and $\tilde{B}_i \neq 0$. It is easy to see that this oracle can be simulated by players in possession of $\mathcal{O}_R$.
Note that his refreshing algorithm assumes that the shares have all non-zero coordinates. In practice, we will use very big fields (at least $|\mathbb{F}| \geq 2^{256}$), so a random vector would have all non-zero coordinates with overwhelming probability.

It is easy to verify that the procedure $\mathsf{Refresh}^n$ of Figure 3 verifies the rerandomizing property. First of all, it is evident that the two shares output by $\mathsf{Refresh}^n$ are indeed a correct masking for the input secret, since

$$
\begin{aligned}
&\langle L', R' \rangle = \\
&= \langle L, R' \rangle + \langle \tilde{U}, R' \rangle = \langle L, R' \rangle + \textstyle\sum_{i=0}^{n} \tilde{U}_i \cdot R'_i = \\
&= \langle L, R' \rangle + \textstyle\sum_{i=0}^{n} \tilde{A}_i \cdot \tilde{B}_i \cdot (R')_i^{-1} \cdot R'_i = \langle L, R' \rangle + \langle \tilde{A}, \tilde{B} \rangle = \\
&= \langle L, R \rangle + \langle L, U \rangle + \langle \tilde{A}, \tilde{B} \rangle = \langle L, R \rangle + \textstyle\sum_{i=0}^{n} L_i \cdot U_i + \langle \tilde{A}, \tilde{B} \rangle = \\
&= \langle L, R \rangle + \textstyle\sum_{i=0}^{n} L_i \cdot L_i^{-1} \cdot A_i \cdot B_i + \langle \tilde{A}, \tilde{B} \rangle = \langle L, R \rangle + \langle A, B \rangle + \langle \tilde{A}, \tilde{B} \rangle = \\
&= \langle L, R \rangle.
\end{aligned}
$$

To see that L' are R' are independent from the input, we set $U = R' - R$ and $\tilde{U} = L' - L$. From the condition $\langle L, R \rangle = \langle L', R' \rangle$ follows $\langle L, U \rangle = -\langle \tilde{U}, R' \rangle$ which is the constraint of $\mathcal{O}_R$. Therefore $\mathcal{O}_R$ outputs samples of the correct distribution to make L', R' independent of L, R.

A reconstructor for $\mathsf{Refresh}^n$ was given in [And12]. We present it in Figure 4.

Input: $(L, L') \in ((\mathbb{F} \setminus \{0\})^n)^2$ is given to P_L and
$(R, R') \in ((\mathbb{F} \setminus \{0\})^n)^2$ is given to P_R
Output: view_L and view_R

Offline:

$\xleftarrow{(V, \tilde{V})}$ $V, \tilde{V} \xleftarrow{\$} \mathbb{F}^n$ $\xrightarrow{(V, \tilde{V})}$

P_L	P_R
$A_i := L_i \cdot V_i$ for $i \in [n]$	
	$U := R' - R$; $B_i := V_i^{-1} \cdot U_i$ for $i \in [n]$ $\tilde{B}_i := R'_i \cdot \tilde{V}_i$ for $i \in [n]$
$\tilde{U} := L' - L$; $\tilde{A}_i := \tilde{V}_i^{-1} \cdot \tilde{U}_i$ for $i \in [n]$	
output $(L, L', V, \tilde{V}, A, \tilde{A})$	output $(R, R', V, \tilde{V}, B, \tilde{B})$

Figure 4: **Reconstructor.** The above algorithm describes a reconstructor for the procedure $\mathsf{Refresh}^n$. The only communication between the parties is the sampling of random vectors V and $\tilde{V}$, which can be done offline.

The author provides a proof that the above procedure is an ϵ-reconstructor for $\mathsf{Refresh}^n$ with $\epsilon = 0$.

B A shrinking procedure for the Inner Product LRS

The Shrink operation is presented in Figure 5. It transforms an encoding of length m into an encoding of length $n+1$. It is based on the implicit shrinking procedure used in the multiplication gadget in [DF12].

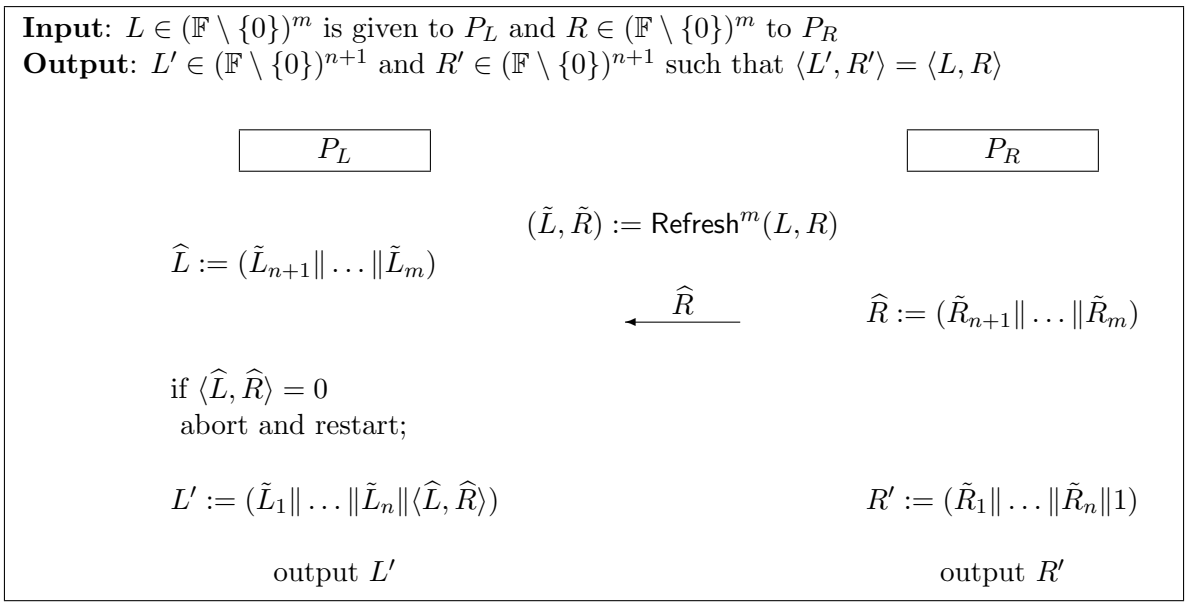

Figure 5: **Shrinking Procedure.** The procedure Shrink described in this figure is used to reduce the size of the shares of a secret.

The algorithm Shrink is interactive, so we need to analyze its security carefully. The reason for this is that for example P_L learns during the execution the value of $\widehat{R}$, which reveals some partial information about the secret state of P_R. An adversary can use this fact and query a leakage function, which depends partially on both of the encodings, and thus break the security of LRS.

We already introduced reconstruct ability in Appendix A. Reconstructability implies that the interaction between two parties does not contradict the leakage resilience. Since the views of P_L and P_R during a reconstructable procedure can be simulated by a non-interactive reconstructor. This reconstructor only uses Oracles which sample randomness which is independent of sensitive values and he does not require any interaction between P_L and P_R.

Theorem B.1. Shrink *is* 0*-reconstructable.*

Proof. The reconstructor for the Shrink operation is presented on Fig. 6. We need to show that reconstructed views $(L, \tilde{L}, L', \widehat{L}, \widehat{R})$ and $(R, \tilde{R}, R', \widehat{R})$ have the same distribution as in the shrink down procedure. This is already clear for L, R and L', R' since the input is identical. In the shrink procedure $\widehat{L}$ and $\widehat{R}$ are uniform elements in $(\mathbb{F} \setminus \{0\})^{m-n}$ and their inner product is $\langle \widehat{L}, \widehat{R} \rangle = \tilde{L}_{n+1}$. The presented reconstructor samples $\widehat{L}$ such that this is the case. The correct distribution of $\tilde{L}$, $\tilde{R}$ follows from the correct distribution of L', R' and $\widehat{L}$, $\widehat{R}$: The first n field elements of $\tilde{L}$, $\tilde{R}$ are identical to the first n field elements of L', R' and the last $m - n$ field elements are identical to $\widehat{L}$, $\widehat{R}$. The reconstructability of the view during the refresh procedure follows from the reconstructability of the refresh procedure. □

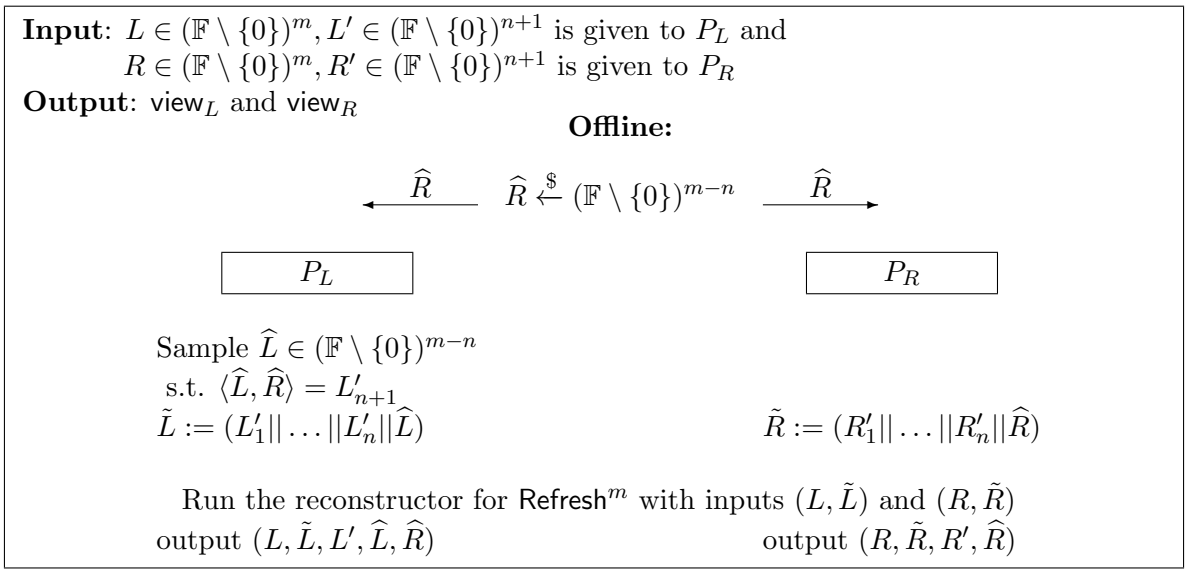

Figure 6: **Reconstructor.** The above algorithm describes a reconstructor for the procedure Shrink. The views created by the reconstructor for Refresh are treated as part of the output.