Efficient Leakage Resilient Circuit Compilers

Marcin Andrychowicz[3], Ivan Damgård[1], Stefan Dziembowski[3], Sebastian Faust[2], and Antigoni Polychroniadou[1(✉)]

[1] Aarhus University, Aarhus, Denmark
{ivan,antigoni}@cs.au.dk
[2] EPFL, Lausanne, Switzerland
sebastian.faust@gmail.com
[3] Warsaw University, Warsaw, Poland
{marcin.andrychowicz,stefan.dziembowski}@crypto.edu.pl

Abstract. In this paper, we revisit the problem of constructing general leakage resilient compilers that can transform any (Boolean) circuit C into a protected circuit C' computing the same functionality as C, which additionally is resilient to certain classes of leakage functions. An important problem that has been neglected in most works on leakage resilient circuits is to minimize the overhead induced by the compiler. In particular, in earlier works for a circuit C of size s, the transformed circuit C' has size at least $\mathcal{O}(sk^2)$, where k is the security parameter. In this work, using techniques from secure Multi-Party Computation, we show that in important leakage models such as bounded independent leakage and leakage from weak complexity classes the size of the transformed circuit can be reduced to $\mathcal{O}(sk)$.

Keywords: Leakage resilience · Multi-party computation · Split-state model · AC^0 · Side channel attacks

1 Introduction

Side channel attacks (SCA) that exploit leakage emitting from a device are among the most severe threats for cryptographic implementations. Since the introduction of timing attacks to the research community in the late 1990s [22],

This is an extended abstract. Further details can be found in the full version.

M. Andrychowicz and S. Dziembowski — Supported by the WELCOME/2010-4/2 grant founded within the framework of the EU Innovative Economy (National Cohesion Strategy) Operational Programme.

S. Faust — Received funding from the Marie Curie IEF/FP7 project GAPS, grant number: 626467.

I. Damgård and A. Polychroniadou — Research supported by the Danish National Research Foundation and the National Science Foundation of China (under the grant 61061130540) for the Sino-Danish Center for the Theory of Interactive Computation and from the Center for Research in Foundations of Electronic Markets (CFEM), supported by the Danish Strategic Research Council.

K. Nyberg (ed.): CT-RSA 2015, LNCS 9048, pp. 311–329, 2015.
DOI: 10.1007/978-3-319-16715-2_17

more sources of side channel leakage have been discovered [14,15,23,28]. To protect cryptographic implementations against these attacks various types of countermeasures have been proposed. One important and particular effective countermeasure already suggested in the early works of Kocher [22] is masking. In a masking scheme the sensitive intermediate data that occurs during the computation of the cryptographic device is encoded with a randomized encoding thereby making leakage of the intermediate values independent of the sensitive values.

The effectiveness of masking as a countermeasure has first been formally studied in the work of Chari *et al.* [3]. While [3] only considered a single masked secret, the concept of *leakage resilient circuit compilers* – pioneered by Ishai *et al.* [19] – studies security guarantees for complicated masked circuits, e.g., a masked AES circuitry. More specifically, a circuit compiler takes as input an arbitrary circuit C computing over some finite field and outputs a protected circuit $\widehat{C}$ that has the same functionality as C but comes with built-in security against certain classes of leakages. It is then shown that even given the leakage from the computation of the transformed circuit $\widehat{C}$ the adversary learns nothing beyond black-box access. Ishai *et al.* [19] consider an adversary that can learn up to t intermediate values that appear during the computation – so-called t-probing adversaries. A large body of recent work has been conducted on extending the leakage classes beyond t-probing adversaries. This has led to great progress and by now we have developed feasibility results in surprisingly strong leakage models (we review the related work in Section 1.1). Since naturally in feasibility results efficiency plays a secondary role, only little progress has been made in improving the efficiency of circuit compilers.

In this work, we make a step towards closing this gap and propose new leakage resilient circuit compilers for broad classes of leakages that come with significantly improved efficiency. Based on techniques from multiparty computation and new techniques for inner-product based transformations, we propose compilers with provable security for global and computationally weak leakages as introduced in the work of Faust *et al.* [12] and for polynomial-time computable leakages in the split-state model [11,24]. As in earlier works and the ones we improve upon [10,12,16,26], we assume that certain simple parts of the computation are leak-free.

1.1 Previous Works

As already mentioned the circuit compiler of Ishai *et al.* [19] considers an adversary that can learn up to t intermediate values of the computation. Various works [2,8,26,27,27,29] consider extensions of the probing model by either proposing more efficient constructions or developing more practice-oriented models. We notice that for a circuit of size s all the above works result into circuits of size $\mathcal{O}(sk^2)$, where s is the size of the original circuit. In [19] Ishai *et al.* also propose an alternative circuit compiler that asymptotically requires only $\tilde{O}(k)$ blow-up, however, in contrast to the other works mentioned above it only achieves statistical security against non-adaptively chosen probing attacks.

We next review some broader classes of leakage functions that go beyond the probing attacks and will be the main focus of this work.

Computationally Weak Leakages. A severe restriction of the probing model is the fact that the leakage is oblivious to large parts of the circuitry. Faust *et al.* [12] eliminate this restriction by considering *global* leakage functions, i.e., the leakage can depend on all the values that are carried on wires during the computation, but the leakage function is assumed to be computationally bounded (i.e., it cannot evaluate certain decoding functions). One concrete example given by the authors is when the shares are k random bits and the decoding function is the parity. For this setting, [12] shows security with respect to arbitrary global AC^0 leakages. The results for the AC^0 setting were recently improved in [25,30]. Similar to the probing case the size of the transformed circuit increases *at least* by a factor $\mathcal{O}(k^2)$ where k is the security parameter.

Circuit Compilers in the Split State Model. The most prominent leakage model of leakage resilient cryptography is the so-called bounded leakage model [11]. In the bounded leakage model the adversary can pick a leakage function $f : \{0,1\}^n \rightarrow \{0,1\}^\lambda$ and obtains $f(state)$, where f is efficiently computable and $\lambda \ll n$. The first work that builds circuits resistant to bounded leakages are the works of Juma and Vahlis [21], and Goldwasser and Rothblum [16]. They consider a model where the algorithm is executed by multiple "processors" that leak independently – so-called split-state leakage. The works of [16,21] use homomorphic encryption and hence rely on computational assumptions. The use of computational assumptions has been eliminated in two recent works [1,10,17] using techniques from the randomness extractor and require an overhead of at least $O(k^2)$.

1.2 Our Contribution

Our main contribution is to show how to construct leakage resilient circuit compilers that asymptotically increase the size of the circuit only by a factor of $\tilde{O}(k)$. We give an overview of our main results below.

Efficient Compilers Against Computationally Weak Leakage. An important building-block of leakage resilient circuit compilers are leakage resilient encoding schemes. Our main observation to improve the efficiency of previous compilers for the setting of computationally weak leakages is to use a linear packed secret sharing scheme to encode the computation. In contrast to the standard Shamir secret sharing where for a random polynomial $p(\cdot)$ of degree t we hide the secret at $p(0)$ while $p(1), \ldots, p(t)$ are viewed as shares of $p(0)$, we use some of the points on the polynomial to hide additional secrets. Notice that this technique has also been used in a series of works starting with Franklin and Yung [13] to improve the asymptotic efficiency of information theoretically secure multi-party computation. In particular, our circuit transformation is heavily inspired by the work of Damgård *et al.* [5], and applies two sequentially executed transformations, namely, TR_1 and TR_2 to produce the protected circuit.

The first transformation TR_1 takes as input the circuit C and produces $C' \leftarrow \mathsf{TR}_1(C)$. Its sole use is to make the circuit ready to be encoded with packed secret sharing, and it does not contribute to the actual security. The second transformation $\widehat{C} \leftarrow \mathsf{TR}_2(C')$ takes as input the so prepared C' and protects it by applying packed secret sharing. The transformation uses the same type of gates as in C', and as in several earlier works [10,12,26] a number of leakage-free gates. We notice that the leak-free components that we use enjoy the same properties as the leak-free components used in earlier works [10,12,26]: they are small (linear in the security parameter), stateless and do not take any inputs. Note that we require two different types of leak-free gates.

We show that the compiler is secure against computationally bounded leakages – so-called AC^0 leakages. To this end, we use the framework introduced by Faust et al. [12] to argue about computationally bounded leakages. As a first step, we show that the above encoding based on packed secret sharing is hard to "break" for AC^0 leakages. This requires that the underlying field is of constant size (independent of the security parameter). We use a recent result of Cramer et al. [4] which presents a linear secret sharing scheme that works for constant field size. As a next step, we prove that all our transformed gadgets in $\widehat{C}$ are reconstructible by constant depth circuits, which by applying the composition lemma of [12] can be extended to composed circuits made from the transformed gadgets. The final transformed circuit has size $\mathcal{O}(s \log(s)k)$.

We also show that the above construction is secure in the probing model of Ishai et al. [19]. When we allow t probes per transformed gadget then our security proof relies on the fact that certain parts of the computation are leak-free. If we aim for security of $\mathcal{O}(s \cdot \mathsf{polylog}(\mathsf{t})/n^2)$ probes in the entire circuit, then we eliminate the leak-free assumption for the stateless circuit case. The transformed circuit we obtain has size $\mathcal{O}(s\mathsf{log}(s) \cdot \mathsf{polylog}(\mathsf{k}))$. Further details are provided in the full version of this extended abstract. We note that a similar construction using packed-secret sharing has been recently considered in [18].

Efficient Compilers for the Split-State Setting. A second contribution is an efficient compiler for the split-state bounded leakage model. We show that the complexity of the compiler of Dziembowski and Faust [10] can be reduced to $\mathcal{O}(k \log k \log\log k)$, where k is the security parameter. This improves earlier works by at least a *quadratic factor* in the security parameter. We achieve this goal by improving the refreshing scheme of [10] which save a linear factor in complexity compared to earlier work. While this would give us only complexity of $\mathcal{O}(k^2)$, we use the fact that the underlying encoding scheme is secure even if the encoding uses only a small constant number of shares, while we increase the size of the underlying field to sub-exponential size. As multiplication in such fields can be done in complexity $\mathcal{O}(k \log k \log\log k)$, we obtain our result.

1.3 Notation

Across the paper, we use a capital letter C to label a circuit. A circuit C carries values from a finite field $\mathbb{F}$ on its wires and is composed of addition and multiplication gates which compute sums and products in $\mathbb{F}$. The size of C is defined

as the number of gates in C and denoted by s. We write $C(x, k)$ for a result of evaluating C on a given input x and the security parameter k. A vector $\mathbf{x}$ is a row vector, and we denote by $\mathbf{x}^\mathsf{T}$ its transposition. We let $\mathbb{F}$ be a finite field and for $m, n \in \mathbb{N}$, let $\mathbb{F}^{m\times n}$ denote the set of $m \times n$-matrices over $\mathbb{F}$. For a matrix $M \in \mathbb{F}^{m\times n}$ and an m bit vector $\mathbf{x} \in \mathbb{F}^m$ we denote by $\mathbf{x} \cdot M$ the n-element vector that results from the matrix vector multiplication of $\mathbf{x}$ and M. For a natural number n let $(0)^n = (0, \ldots, 0)$. We use $\mathbf{x}[i]$ to denote the ith element of a vector $\mathbf{x}$ and $\mathbf{x}[i, \ldots, j]$ to denote the elements $i, i + 1, ..., j$ of $\mathbf{x}$. In addition, let $[\mathbf{x}]$ denote an encoding which secret shares a block $\mathbf{x}$ of ℓ elements and write the k shares as $[\mathbf{x}]_=(x_1, \ldots, x_k)$ where k is the security parameter. Let $[\mathbf{x}]_c$ denote an encoded block secret shared under a linear code c specified by a generator matrix G. A secret sharing scheme is homomorphic if $[\mathbf{x}] + [\mathbf{y}]$ and $[\mathbf{x}][\mathbf{y}]$ are shares of the blocks $\mathbf{x}+\mathbf{y}$ and $\mathbf{xy}$. Moreover, let $\pi(\mathbf{x})$ be a random permutation of the vector $\mathbf{x}$. For two random variables X_0, X_1 over $\mathcal{X}$ we define the statistical distance between X_0 and X_1 as $\Delta(X_0; X_1) = \sum_{x\in\mathcal{X}} 1/2|\Pr[X_0 = x] - \Pr[X_1 = x]|$.

2 Defining a Circuit Transformation

We consider circuits C with secret state m that operate over some finite field $\mathbb{F}$, take some public input x and produce an output y. We assume that C consists of two types of elementary gates. First, addition and multiplication gates that both input two field elements and compute the corresponding operation in $\mathbb{F}$. Second, the so-called copy-gates that take as input a field element and output two copies of it to handle fan-out of the circuit. One may think of C as an implementation of a block cipher where the state is the key and the public input the plaintext.

A circuit transformation TR compiles any circuit C and the associated initial secret state m_0 into a functionally equivalent circuit $\widehat{C}$ and transformed secret state $\hat{m}_0$ that is resistant to certain leakage attacks characterized by a family of functions $\mathcal{L}$. To model the leakage from $\widehat{C}$ we introduce a leakage game that is executed by an adversary $\mathcal{A}$. In the leakage game, the adversary can submit tuples of the form (x_i, f_i) where x_i denotes an input to the circuit and $f_i \in \mathcal{L}$ is a leakage function. For each query, $\mathcal{A}$ receives the output y_i when using current state m_{i-1} and the corresponding leakage. The exact definition of the leakage depends on the leakage model and will be specified later in this paper. We denote by $\left(\mathcal{A}_\mathcal{L} \rightleftarrows \widehat{C}[\hat{m}_0]\right)$ the output of $\mathcal{A}$ after interacting with the transformed circuit $\widehat{C}$ with initial state $\hat{m}_0$. Moreover, we consider a simulated world where a simulator $\mathcal{S}$ only obtains black-box access to $C[m_0]$, which we denote by $(\mathcal{S} \rightleftarrows C[m_0])$. Security of a circuit transformation guarantees that the output of the adversary in the leakage game is indistinguishable from the output of the simulator in the ideal world. We define the notion of an $\mathcal{L}$-secure circuit transformation below.

Definition 1. *A circuit transformation* TR *is secure with respect to leakages from a family of functions* $\mathcal{L}$ *if the following two properties hold:*

1. Soundness: *For any circuit* C *and any initial state* m_0 *and any input* x_i *we have* $C[m_{i-1}](x_i) = \widehat{C}[\hat{m}_{i-1}](x_i)$.

2. Security: *For any PPT adversary* $\mathcal{A}_{\mathcal{L}}$ *and any circuit* C *with initial state* m_0*, there exists a simulator* $\mathcal{S}$ *such that for all circuits* C *with initial state* m_0 *the following holds:* $\Delta\left((\mathcal{S} \rightleftarrows C[m_0]) ; (\mathcal{A}_{\mathcal{L}} \rightleftarrows \widehat{C}[\hat{m}_0])\right) \leq \mathsf{negl}(k)$.

The transformed circuit $\widehat{C}$ shall use the same types of operations as the underlying circuit C, i.e., if C operates over the binary field, then the elementary operations used in $\widehat{C}$ are Boolean operations. Moreover, for some of our security proofs, we will require so-called *opaque gates*. Similarly, to earlier works [10,12,16,21] on leakage-resilient circuit compilers our leak-free gates do not leak from their internals, but can leak on their outputs. All our leak-free gates do not take any inputs, but merely sample from some efficiently sampable distribution. We will later in this section precisely characterize what operation is carried out by our leak-free gates.

All leakage-resilient circuit transformations follow the same paradigm to transform C into a protected circuit $\widehat{C}$. First, they use a leakage resilient encoding scheme $\Pi = (\mathsf{Enc}, \mathsf{Dec})$ to encode the values carried on the wires of C. More precisely, each wire w in C is represented in $\widehat{C}$ by a wire-bundle carrying the encoding $\mathsf{Enc}(w)$. Notice that also the content of the memory that, e.g., stores the secret key is stored in encoded form in $\widehat{C}$. Clearly, to show that the transformed circuit $\widehat{C}$ is secure against leakage functions from $\mathcal{L}$ our encoding scheme has to be resilient for functions from $\mathcal{L}$.

The next (and more challenging) step is to develop a transformation for the elementary operations of C. For instance, if C is a Boolean circuit, then we need to give secure implementations for NAND gates. Following earlier works, we call these transformed elementary operations *gadgets*. A gadget takes as input encodings and outputs the encoded result, e.g., if the gadget implements a multiplication of two encodings $\mathsf{Enc}(a)$ and $\mathsf{Enc}(b)$, then its output is $\mathsf{Enc}(a \cdot b)$. The difficulty is to design the gadgets in such a way that they guarantee *correctness*, i.e., the output encodes the correct result, while at the same time leakage from the internals of the operations do not reveal any information about the encoded secrets. To this end, we need to ensure that our gadgets operate on encodings, by exploiting some homomorphic property of the underlying encoding scheme. Finally, since our gadgets in the transformed circuit $\widehat{C}$ work on encoded values, we need two additional types of gates: an encoder gate that takes as input a field element x and outputs $\mathsf{Enc}(x)$, and a decoding gadget that takes as input $\mathsf{Enc}(x)$ and returns x. These gates may leak but as shown in earlier works they do not influence the security of transformed circuit and will be ignored for the remainder of this extended abstract.

The above approach of transforming circuits is called a *gate-by-gate transformation*, which allows us to explain circuit transformations in a modular way. That is, given some leakage resilient encoding, we present basic transformations of the gates used in the original circuit C, which by composing these transformed gates results in the transformed circuit $\widehat{C}$. For both of our schemes, we consider circuits C that compute over some finite field $\mathbb{F}$ and assume that the original circuit has multiplication and addition gates that carry out the corresponding operations in the field $\mathbb{F}$.

3 Transformation for Computationally Weak Leakages

Our first transformation achieves security against a family of leakage functions $\mathcal{L}$ that are computable by polynomial-size constant depth circuits (so-called AC^0 circuits). First, we start by defining our general encoding scheme $\Pi_{LPSS} = (\mathsf{Enc}_{LPSS}, \mathsf{Dec}_{LPSS})$, which is an important tool for the transformation. We continue presenting the circuit transformation consisting of the transformations TR^1_{weak} and TR^2_{weak}, where $\mathsf{TR}_{weak} := \mathsf{TR}^2_{weak} \circ \mathsf{TR}^1_{weak}$.

The Encoding Scheme $\boldsymbol{\Pi_{\mathbf{LPSS}}} = (\mathsf{Enc}_{\mathbf{LPSS}}, \mathsf{Dec}_{\mathbf{LPSS}})$. A $(t+1)$-out-of-k secret sharing scheme takes as input a secret x from some input domain and outputs k shares, with the property that it is possible to efficiently reconstruct x from every subset of $t+1$ shares, but every subset of at most t shares reveals nothing about the secret x. Informally, Shamir's secret-sharing scheme [32] is defined by a polynomial $p(\cdot)$ of degree at most t, such that $p(0) = x$. The shares are defined to be $p(a_i)$ for every $i \in 1, \ldots, k$ where $a_1, \ldots, a_k$ are any distinct non-zero elements of $\mathbb{F}$. The reconstruction algorithm of the scheme is based on the fact that any $t+1$ points define exactly one polynomial of degree t. Thus, using Lagrange interpolation, it is possible to efficiently reconstruct the polynomial $p(\cdot)$ given any subset of $t+1$ points and compute $x = p(0)$.

Our underlying leakage resilient encoding scheme is a packed secret sharing scheme. The idea of packed secret sharing dates back to Yung and Franklin [13] who used the technique to reduce the complexity of multiparty computation protocols. The idea is similar to standard Shamir secret-sharing [32] over a field $\mathbb{F}$, but where a block of ℓ different secret values $\mathbf{x} = (x_1, \ldots, x_\ell)$ is shared at once using a single polynomial $p(\cdot)$ of degree d that now evaluates to $(x_1, \ldots, x_\ell)$ in ℓ distinct points. It is easy to see that we can obtain security against a t-probing adversary by choosing $d + 1 = t + \ell$.

To obtain security against leakages described by low-depth Boolean circuits, we need a scheme that works over constant size fields such that the underlying operations can be evaluated by small-depth Boolean circuits.[1] Hence, our underlying leakage resilient encoding scheme uses the packed secret sharing scheme of Cramer et al. [4] who showed how packed secret sharing can be combined with techniques from algebraic geometry to make it work for constant field sizes. Since we need a more general model for our purposes rather than the special case of Shamir's scheme, we follow the approach from [4] and define our packed secret sharing scheme in terms of linear codes.

More specifically, a linear packed secret sharing scheme over the finite field $\mathbb{F}$ is defined by the following parameters: number of shares k, secret length $\ell > 1$, randomness length e, privacy threshold t and reconstruction threshold r such that any subset of at most t shares have distribution independent of the secret block and from any set of at least r shares, one can reconstruct the secret block. Also, such a linear secret sharing scheme can define a linear code c with generator

[1] Jumping ahead this is needed to carry out an AC^0 reduction to the hardness of the inner product.

matrix $G \in \mathbb{F}^{k\times(\ell+e)}$, and in this case the set of all encodings $[\mathbf{x}]_c$ form a linear code c. [2]

Formally, our encoding $\Pi_{LPSS} = (\mathsf{Enc}_{LPSS}, \mathsf{Dec}_{LPSS})$, is as follows:

- *Public parameters of the scheme:* Let $G \in \mathbb{F}^{k\times(\ell+e)}$ be a fixed generator matrix of a linear code c. More details on how this matrix will look like are given in [4].
- *Encoding algorithm* $\mathsf{Enc}_{LPSS}(\boldsymbol{x})$*:* On input the block $\mathbf{x} = (x_1, \ldots, x_\ell)$, choose a random vector $\boldsymbol{\rho} \leftarrow \mathbb{F}^e$ and compute the encoded block under the code c as: $[\mathbf{x}]_c = G \cdot (x_1, \ldots, x_\ell, \rho_1, \ldots, \rho_e)^\mathsf{T}$. Output $[\mathbf{x}]_c \in \mathbb{F}^k$.
- *Decoding algorithm* $\mathsf{Dec}_{LPSS}([\boldsymbol{x}]_c)$*:* On input r shares of $[\mathbf{x}]_c$, recover the block $\mathbf{x}$ consisting of the first ℓ values of the computation $(G^{-1} \cdot [\mathbf{x}]_c) \in \mathbb{F}^{\ell+e}$.

Multiplying Shares. If two encodings $[\mathbf{x}]_c$ and $[\mathbf{y}]_c$ are multiplied then $[\mathbf{xy}]_{c^*}$ is obtained where the multiplication yields a codeword under a new code c^* defined considering the derived generator matrix[3] G^*. The code c^* is defined to be the code obtained by taking the linear span of all products of the codewords in c. Hence, the above encoding scheme Π_{LPSS} applies to any generator matrix, e.g. the matrix G^*.

An important feature of the above encoding scheme, which makes it applicable for masking schemes, is the fact that it exhibits homomorphic properties. In particular, any linear combination of encodings corresponds to a linear combinations of the underlying secrets, provided that the *same* secret locations were used in all encodings (in the above case these are the position $(1, \ldots, \ell)$).

The Transformation $C' \leftarrow \mathsf{TR}^1_{weak}(C, k)$**.** The circuit transformation takes as input the security parameter k and the description of an arbitrary circuit C and compiles it into a transformed circuit C'. The goal of $\mathsf{TR}^1_{weak}(C, k)$ is to prepare the circuit C such that it can efficiently compute on values encoded with Π_{LPSS}. The use of packed secret sharing allows to securely compute addition/multiplication on ℓ values in parallel, at the price of what a single operation would cost using normal secret sharing. To this end, the circuit has to be arranged in such a way that it can operate on blocks of secrets in parallel. The work of [5] achieves this goal with overhead $\mathcal{O}(\log |s|)$, a detailed description can be found in the full version of this extended abstract. Intuitively, the transformation arranges the circuit C such that every layer contains only one type of gates, i.e., either addition or multiplication operations. In addition, sets of shared blocks $S_1, S_2, \ldots$ must be produced such that blocks in S_i contain the i'th input bit to the gates in a given layer, in some fixed order. In order to achieve the above, the values in the computation will have to be permuted between layers

[2] For example, in the special case of packed Shamir secret sharing over polynomials of degree d, the set of all encodings $[\mathbf{x}]_d$ forms a linear code, where the generator matrix is formed from a product of two VanderMonde matrices.

[3] Analogously, if packed Shamir secret sharing over polynomials is used, then the multiplication of the shares $[\mathbf{a}]_d$ and $[\mathbf{b}]_d$ under a polynomial of degree d yields a share of $[\mathbf{ab}]_{2d}$ under a polynomial of degree $2d$.

by swap gates in arbitrary ways that depend on the concrete circuit. The basic idea is to handle the arbitrary permutations between blocks using Benes permutation networks. The only non-trivial issue is how to permute the elements *inside* a shared block. For this reason we add permutation block-gadgets which are described in TR^2_{weak}.

Since the resulting circuit C' is compiled to work on packed secret sharing, it can be described by block-gadgets, which operate on blocks of ℓ secrets. More specifically, we have the following block-gadgets: (1) multiplication and addition block gadgets that carry out the respective operation over blocks of ℓ field elements, (2) copy gadgets that handle fan out and output two copies of their inputs, and (3) permutation gadgets (for some fixed number of $\log(k)$ permutations), which take as input a block and output a permutation of the elements. Note that the block-gadgets is just an abstraction to make the exposition in the next step simpler, and all block-gadgets are built out of the elementary multiplication and addition gates that work over the underlying field $\mathbb{F}$.

The Transformation $\widehat{C} \leftarrow \mathsf{TR}^2_{weak}(C', k)$. $\mathsf{TR}^2_{weak}(C', k)$ takes as input a circuit C', prepared by TR^1_{weak} to work on blocks of ℓ secrets, and compiles it into a circuit $\hat{C}$ that works on encodings of blocks. Recall that C' operates on blocks, so as a first step, $\mathsf{TR}^2_{weak}(C', k)$ encodes blocks using Π_{LPSS}. Moreover, we can replace the block-gadgets by operations that work on encoded blocks. The gadgets are built out of the elementary operations: multiplication, addition and swapping. Moreover, we will use a class of a leak-free gadget which is described in more detail below. The transformation for the different block-gadgets is described in Figure 1. The addition block-gadget directly uses the fact that the encoding scheme is additively homomorphic, and hence to compute the output it suffices to compute component-wise addition of the shares. The transformation for the multiplication block-gadget is more complicated and makes use of a leak-free gate. Specifically, we first compute the component-wise product of the shares. Notice that the resulting shares $[\mathbf{ab}]_{c^*}$, are now shares of the code c^* since once we multiply them the underlying code was changed from c to c^*. Next, we use the opaque gadget $\mathcal{O}_r$, which returns encodings $[\mathbf{r}]_c$ and $[\mathbf{r}]_{c^*}$ of a random block $\mathbf{r}$. We use $[\mathbf{r}]_{c^*}$ to "mask" $[\mathbf{ab}]_{c^*}$, which will allow us to open/decode the random encoding $[\mathbf{ab}+\mathbf{r}]_{c^*}$ to the block $(\mathbf{ab}+\mathbf{r})$. Intuitively, the opening is allowed since the secret data is masked by a random unknown value $\mathbf{r}$. Hence, the opened values does not reveal any information about the secret data. After reconstructing/encoding the block $(\mathbf{ab}+\mathbf{r})$ under the code c, we subtract $[\mathbf{r}]_c$ from it, which results into a random encoding $[\mathbf{ab}]_c$. The reconstruction of $[\mathbf{ab}+\mathbf{r}]_c$ carried out during the multiplication operation can be implemented using a small sub-circuit with linear complexity. Notice that the Permutation block-gadget acts in the same way as the multiplication block-gadget. Moreover, we use a similar leak-free gate, $\mathcal{O}_\pi$, which for some fixed permutation π outputs $[\mathbf{r}]_c$ and $[\mathbf{r}']_c$ for some random block $\mathbf{r} \leftarrow \mathbb{F}^\ell$ and $\mathbf{r}' = \pi(\mathbf{r})$. The leak-free gates do not perform any *heavy* computations, take no inputs and keep no secret internal states, which makes them independent of the computation in the circuit.

Addition block-gadget : $\mathbf{g} = \mathbf{a} + \mathbf{b} \Rightarrow [\mathbf{a}+\mathbf{b}]_c \leftarrow [\mathbf{a}]_c + [\mathbf{b}]_c$
$[\mathbf{s}]_c = [\mathbf{a}]_c + [\mathbf{b}]_c = (a_1 + b_1, \ldots, a_k + b_k)$
$[\mathbf{0}]_c \leftarrow \mathcal{O}^a$
$[\mathbf{a}+\mathbf{b}]_c = [\mathbf{s}]_c + [\mathbf{0}]_c$
Multiply block-gadget : $\mathbf{g} = \mathbf{ab} \Rightarrow [\mathbf{ab}]_c \leftarrow [\mathbf{a}]_c[\mathbf{b}]_c$
$([\mathbf{r}]_c, [\mathbf{r}]_{c^*}) \leftarrow \mathcal{O}_r$
$[\mathbf{ab}]_{c^*} = [\mathbf{a}]_c[\mathbf{b}]_c = (a_1 b_1, \ldots, a_k b_k)$
$[\mathbf{ab}+\mathbf{r}]_{c^*} = [\mathbf{ab}]_{c^*} + [\mathbf{r}]_{c^*} = (a_1 b_1 + r_i, \ldots, a_k b_k + r_k)$
$(\mathbf{ab}+\mathbf{r}) \leftarrow \mathsf{Dec}_{LPSS}([\mathbf{ab}+\mathbf{r}]_{c^*})$
$[\mathbf{ab}+\mathbf{r}]_c \leftarrow \mathsf{Enc}_{LPSS}(\mathbf{ab}+\mathbf{r})$
$[\mathbf{ab}]_c = [\mathbf{ab}+\mathbf{r}]_c - [\mathbf{r}]_c$
Permutation block-gadget : Compute $[\pi(\mathbf{x})]_c$ given $[\mathbf{x}]_c$
$([\mathbf{r}]_c, [\pi(\mathbf{r})]_c) \leftarrow \mathcal{O}_\pi$
$[\mathbf{x}+\mathbf{r}]_c = [\mathbf{x}]_c + [\mathbf{r}]_c = (x_1 + r_1, \ldots, x_k + r_k)$
$(\mathbf{x}+\mathbf{r}) \leftarrow \mathsf{Dec}_{LPSS}([\mathbf{x}+\mathbf{r}]_c)$
$[\pi(\mathbf{x}+\mathbf{r})]_c \leftarrow \mathsf{Enc}_{LPSS}(\pi(\mathbf{x}+\mathbf{r}))$
$[\pi(\mathbf{x})]_c = [\pi(\mathbf{x}+\mathbf{r})]_c - [\pi(\mathbf{r})]_c$
$\mathcal{O}_\pi$ **gate** : Compute $[\mathbf{r}]_c$ and $[\pi(\mathbf{r})]_c$
Sample $\mathbf{r} = (r_1, \ldots, r_l) \in_R \mathbb{F}^l$ and compute $\pi(\mathbf{r})$
Sample two random vectors $\boldsymbol{\rho}$,$\boldsymbol{\rho}'$
Compute $[\mathbf{r}]_c = G(\mathbf{r}, \boldsymbol{\rho})$ and $[\pi(\mathbf{r})]_c = G^*(\pi(\mathbf{r}), \boldsymbol{\rho}')$
$\mathcal{O}_r$ **gate** : Compute $[\mathbf{r}]_c$ and $[\mathbf{r}]_{c^*}$
Sample $\mathbf{r} = (r_1, \ldots, r_l) \in_R \mathbb{F}^l$ and two random vectors $\boldsymbol{\rho}$,$\boldsymbol{\rho}'$
Compute $[\mathbf{r}]_c = G(\mathbf{r}, \boldsymbol{\rho})$ and $[\mathbf{r}]_{c^*} = G^*(\mathbf{r}, \boldsymbol{\rho}')$

[a] *The gate* $\mathcal{O}$ *is a special case of the* $\mathcal{O}_r$ *gate in the sense that the random value* r *is freezed to* 0 *and only the encoding* $[\mathbf{0}]_c$ is given as output.

Fig. 1. Multiply, Addition and Permutation block-gadgets for the Π_{LPSS} encoding

Efficiency of TR_{weak}. As already mentioned above, $\mathsf{TR}^1_{weak}(C, k)$ introduces overhead of $\mathcal{O}(\log s)$ resulting into a circuit of size $\mathcal{O}(s \log s)$. Regarding TR^2_{weak} (C', k), the blow up of the circuit is linear in k because we replace the gates by block-gadgets of cost $\mathcal{O}(k)$. The efficiency of the transformation $\mathsf{TR}^2_{weak}(C', k)$ is achieved by using the packed secret sharing scheme Π_{LPSS} which allows to amortize the cost over many gates. More specifically, our multiplication/permutation block-gadgets have at most quadratic overhead due to the matrix-vector multiplication induced by running $\mathsf{Enc}_{LPSS}, \mathsf{Dec}_{LPSS}$. However, with packed secret sharing we process ℓ blocks in parallel, so amortize the cost over many gates we get linear complexity since $\ell = \Theta(k)$. Therefore, the total size of the transformed circuits is $\mathcal{O}(k\ s \log s)$.

Soundness of TR_{weak}. We show that the input-output functionality of the circuit C is identical to that of $\hat{C}$. The proof of soundness can be found in the full version.

Lemma 1. [Soundness] *The circuit transformation* $\mathsf{TR}_{weak}(C, k)$ *is sound.*

3.1 Security Against Global and Computational Bounded Leakage

We now show that our leakage-resilient compiler TR_{weak} protects against global, continuous and computationally weak leakages. In particular, we will prove that such circuits protect against leakages that can be computed by circuits of constant depth – so-called AC^0 leakages. The security proof of our transformation follows the general approach presented in the work of Faust et al. [12] which requires two main ingredients, a secure encoding scheme and simulatable gadgets. Informally, given these two ingredients one can apply the composition theorem of Faust et al. [12] and get security for the entire transformation.

We start by showing a general result, working with arbitrary fields, such that the operations of the computation are efficiently simulatable by $\mathsf{SHALLOW}$ circuits – in particular, we will require that they can be computed by circuits in the class $\mathsf{SHALLOW}(d, s)$ for some constant depth d and size s. We assume that these circuits are deterministic and the only basic operations they are allowed to carry out are additions, permutations and multiplications. Next, in Section 3.2, we consider the case where the shallow functions operate over binary fields and in this case we can show that the class of leakage functions that can be tolerated is in AC^0. For this we need that the size of the field is constant and we need to show that our encoding scheme Π_{LPSS} is secure against AC^0 leakages.

To formalize the notion of shallow simulators we use the formalism of *reconstructors*, defined in [12], for some class $\mathcal{L}$ of leakage functions. A reconstructor takes as input the inputs and outputs of a gadget and is able to simulate its internals in a way that looks indistinguishable for leakages from $\mathcal{L}$. Since we are interested in efficient simulations and reductions, we will explicitly state the complexity of the *reconstructors*. Below, we denote the distribution on the wires of $\widehat{C}$ on input X conditioned on the output being Y by $\mathcal{W}_{\widehat{C}}(X|Y)$.

Definition 2 (Reconstructor for circuits [12]). *Let $\widehat{C}$ be a (transformed) circuit. We say that a pair of strings (X, Y) is* plausible for $\widehat{C}$ *if $\widehat{C}$ might output Y on input X, i.e., if $\Pr[\widehat{C}(X) = Y] > 0$.*

Consider a distribution $\mathsf{REC}_{\widehat{C}}$ over the functions whose input is a pair of strings, and whose output is an assignment to the wires of $\widehat{C}$. Define $\mathsf{REC}_{\widehat{C}}(X, Y)$ as the distribution obtained by sampling $R_{\widehat{C}} \leftarrow \mathsf{REC}_{\widehat{C}}$ and computing $R_{\widehat{C}}(X, Y)$. Such a distribution is called a $(\mathcal{L}, \epsilon)$-reconstructor for $\widehat{C}$ *if for any plausible (X, Y), the following two wire assignment distributions are ϵ close under leakages from $\mathcal{L}$, i.e., for any function $f \in \mathcal{L}$ the following holds:*

$$\Delta(f(\mathcal{W}_{\widehat{C}}(X|Y)); f(\mathsf{REC}_{\widehat{C}}(X, Y))) \leq \epsilon,$$

where the randomness above is over the randomness of sampling $\mathsf{REC}_{\widehat{C}}$ and the internal randomness used by $\widehat{C}$. We say that $\widehat{C}$ is reconstructible by $\mathsf{SHALLOW}(d, s)$ *if the support of the distribution $\mathsf{REC}_{\widehat{C}}$ is computable by circuits in* $\mathsf{SHALLOW}(d, s)$.

Security and Reconstructibility of Block-gadgets. We show that our transformed Multiply block-gadget is reconstructible by SHALLOW circuits. Moreover, the proofs and the reconstructors of the Permutation and Additon block-gadget can be found in the full version.

Lemma 2 (The Multiply block-gadgets of TR_{weak} are Reconstructible). *Let k be the security parameter. The* Multiply *block-gadget is $(\mathcal{L}, 0)$-reconstructible by* $\mathsf{SHALLOW}(2, \mathcal{O}(k))$ *for any $\mathcal{L}$.*

3.2 Security Against AC^0 Leakage

While the above work, in this section, we consider security with respect to AC^0 leakages. To this end, we will first show that the packed secret sharing scheme from Section 2 is secure against AC^0 leakages (see lemma below). Then, we use the composition theorems from [12] together with the fact that all block-gadgets are proven to be reconstructible by shallow circuits. This will show that the transformed circuits are resilient to AC^0 leakages according to Definition 1. In the following, we show that the encoding is secure against AC^0 leakages.

Security of the Π_{LPSS} Encoding. Recall that the circuit compiler TR_{weak} uses the Π_{LPSS} packed secret sharing scheme over a constant size field $\mathbb{F}$. In general, the decoding function Dec_{LPSS} is a function that maps a set of shares to a secret block and the adversary gets to apply an AC^0 function to an encoding. We show that the decoding function is hard to compute in AC^0. The proof of the Lemma can be found in the full version.

Lemma 3. *For $k \in \mathbb{N}_{>0}$, the decoding function Dec_{LPSS}[4] defined by a decoding vector $(\boldsymbol{a}) = (a_1, \ldots, a_k) \in \mathbb{F}^k$ as $\mathsf{Dec}_{LPSS} : (s_1, \ldots, s_k) \mapsto \sum_{i=1}^{k} s_i a_i = \boldsymbol{a}^T \boldsymbol{s}$ does not belong to the AC^0.*

Notice that the above claim proves that the Π_{LPSS} encoding is secure against AC^0 leakages, i.e., for any leakage function f computable by AC^0 circuits and for any two blocks of secrets $\mathbf{x}$ and $\mathbf{x}'$ the following two distributions are statistically close: $\Delta(f(\mathsf{Enc}_{LPSS}(\mathbf{x})); f(\mathsf{Enc}_{LPSS}(\mathbf{x}')) \leq \mathsf{negl}(k)$.

Security of Circuits. In Section 3.1, we showed that all block-gadgets can be reconstructed by shallow simulators given only the inputs and outputs. Moreover, all gadgets are re-randomizable as the outputs are re-randomized by fresh encodings that are output by leak-free gates. Furthermore, by the claim above the encoding is resilient to AC^0 leakages and all block-gadgets are reconstructible by constant depth circuits, and the composition theorem of [12] combines both terms additively, i.e., the loss in the reduction is only in an additive constant factor in circuit depth. This all together shows that the transformed circuits are secure against AC^0 circuits.

[4] We actually encode blocks of elements, but for simplicity of exposition we will assume that we encode single elements.

4 Transformation for Independent Leakages

Dziembowski and Faust [10] proposed a compiler, which transforms arbitrary circuits over some field $\mathbb{F}$ into functionally equivalent circuits secure against any continual leakage assuming that: (*i*) the memory is divided into sub-computations, which leak independently, (*ii*) the leakage from each sub-computation is bounded and (*iii*) the circuit has an access to a leak-free component, which samples random pairs of orthogonal vectors. Their transformation also proceeds in a *gate-by-gate* fashion but the pivotal ingredient of their construction is a protocol for *refreshing* the Leakage-Resilient-Storage used among others in the multiplication gadget protocol.

We present a more efficient and simpler protocol for *refreshing* using the same assumption of a leak-free gadget. Our solution needs $\mathcal{O}(k)$ operations to fully refresh the encoding of the secret in contrast to $\Omega(k^2)$ that was required by earlier works, where k is a security parameter proportional to the bound on the leakage from each sub-computation. More precisely, the blow-up of the circuit's size during compilation with the new refreshing protocol is equal to $\mathcal{O}(k^2)$. It is a significant improvement compared to $\Omega(k^4)$ by Dziembowski and Faust [10], and $\Omega(k^3)$ by Goldwasser and Rothblum [30] where ciphertext-banks are needed.

Moreover, we show that by operating over larger fields (exponential in the security parameter k) and using an efficient field multiplication algorithm, we can achieve even more efficient construction, namely one with a (multiplicative) overhead $\mathcal{O}(k \log k \log\log k)$ regardless whether the new refreshing algorithm is used or not. Although the usage of the more efficient refreshing algorithm does not influence the complexity of the compiler in this case, it is still valuable, because it decreases the size of the transformed circuit by a huge constant factor and is simpler than the original refreshing algorithm.

In contrast, in the previous section we aimed on protecting against low-complexity leakages, in this section we are interested in bounded leakage that occurs independently from two parts of the memory. To this end, we need to use an alternative encoding – the so-called inner-product encoding – that has been used in a series of works [7,10,17]. The inner product encoding scheme, defined by $\Phi_{IP} = (\mathsf{Encode} : \mathcal{M} \to \mathcal{X} \times \mathcal{Y}, \mathsf{Decode} : \mathcal{X} \times \mathcal{Y} \to \mathcal{M})$, works as follows. On an input message $m \in \mathcal{M}$, we choose uniformly at random two vectors $\mathbf{l}$ and $\mathbf{r}$ over the field $\mathbb{F}$ subject to the constraint that their inner product $\langle \mathbf{l}, \mathbf{r} \rangle$ is equal to the message m. The encoding scheme outputs $(\mathbf{l}, \mathbf{r})$. The decoding function is deterministic and takes as input two shares $(\mathbf{l}, \mathbf{r})$ and outputs their inner product $\langle \mathbf{l}, \mathbf{r} \rangle$. In [6,9] it is shown that the above encoding is secure, which means that the adversary learning some partial information $f(\mathbf{l})$ about $\mathbf{l}$ and (independently) $g(\mathbf{r})$ about $\mathbf{r}$ gains no information about the encoded message m. The idea is to keep $\mathbf{l}$ and $\mathbf{r}$ separated (e.g. on different memory chips). We will model this setting assuming that they are kept by different players, which can perform computation and exchange messages. We discuss our encoding more formally in Section 4.2.

4.1 Leakage Model

Our model of leakage is based on [10] and we only recall some important details. The compiler produces a circuit, which is divided into sub-computations. An adversary will be allowed to extract from each sub-computation no more that λ bits of information for some constant λ. For a non-adaptive adversary, it means that it is allowed to adaptively choose any (e.g. polynomially uncomputable) function with range $\{0,1\}^\lambda$, which value depends on all information used in that sub-computation. Except from the above condition, the total amount of leakage during the whole computation is unlimited (in comparison to models, when an adversary can for example obtain values on a fixed number of wires). Because of that, this kind of leakage model is usually called *continual leakage*.

Moreover, we will assume that the sub-computations leak *independently*, that is a leakage function in each observation may only depend on data from one sub-computation. In practice, the separation of sub-computations may be achieved by dividing the memory into parts (e.g. separate RAM chips) and placing the data used in different sub-computations on different memory chips.

We model the execution of such circuit as a protocol executed between ℓ players (denoted $P_1, P_2, \ldots, P_\ell$), where each player performs one of the sub-computations. The adversary can then learn some partial information about the internal states of the players. Informally, an adversary, called *λ-limited leakage adversary* is allowed to extract at most λ bits of information about the internal state of each players. More formal definitions follows in the next paragraphs.

Leakage from Memory. Based on Definition 1 we model independent leakage from memory parts in form of a *leakage game*, where the adversary can *adaptively* learn information from the memory parts. More precisely, for some $u, \ell, \lambda \in \mathbb{N}$ let $M_1, \ldots, M_\ell \in \{0,1\}^u$ denote the contents of the memory parts, then we define a *λ-leakage game* in which a *λ-limited leakage adversary* $\mathcal{A}$, submits (adaptively) tuples of the form $\{(x_i, f_i)\}_{i=1}^m$ where $m \in \mathbb{N}$, $x_i \in \{1, \ldots, \ell\}$ denotes which memory part leaks at the current step and f_i is a leakage function, such that $f_i : \{0,1\}^u \to \{0,1\}^{\lambda'}$ and $\lambda' \leq \lambda$. To each such a query the oracle replies with $f_i(M_{x_i})$ and we say that in this case the adversary $\mathcal{A}$ *retrieved the value* $f_i(M_{x_i})$ *from* M_{x_i}. The only restriction is that in total the adversary does not retrieve more than λ bits from each memory part. In the following, let $(\mathcal{A} \rightleftarrows (M_1, \ldots, M_\ell))$ be the output of $\mathcal{A}$ at the end of this game. Without loss of generality, we assume that $(\mathcal{A} \rightleftarrows (M_1, \ldots, M_\ell)) := (f_1(M_{x_1}), \ldots, f_m(M_{x_m}))$.

Leakage from Computation. We model an execution of the circuit as a protocol executed between players $P_1, P_2, \ldots P_\ell$, where each player corresponds to one of the sub-computations. At the beginning of the game, some of the players (so-called input-players) hold inputs. The execution of the protocol proceeds in rounds. During each round one player is active and can send messages to the other players. The messages can depend on his input (if it is an input-player), the messages he received in previous rounds and his local randomness. An the end of the game, some of the players (called output-players) output the values, which

are considered the output of the protocol. Let view_i denote all the information, which were available to P_i, that is all the messages sent or received by P_i, his inputs and his local randomness. After the protocol is executed the adversary plays a game $\mathcal{A} \rightleftarrows (\mathsf{view}_1, \ldots, \mathsf{view}_\ell)$.

4.2 Leakage-Resilient Storage

The notion of leakage-resilient storage $\Phi = (\mathsf{Encode}, \mathsf{Decode})$ was introduced by Davi et al. [7]. An Φ allows to store a secret in an *encoded form* of two *shares* $\mathbf{l}$ and $\mathbf{r}$, such that it should be impossible to learn anything about the secret given independent leakages from both shares. One of the constructions that they propose uses two-source extractors and can be shown to be secure in the independent leakage model.

A Φ scheme is said to be (λ, ϵ)-secure, if for any $\mathbf{s}, \mathbf{s}' \in \mathcal{M}$ and any λ-limited adversary $\mathcal{A}$, we have $\Delta(\mathcal{A} \rightleftarrows (\mathbf{l}, \mathbf{r}); \mathcal{A} \rightleftarrows (\mathbf{l}', \mathbf{r}')) \leq \epsilon$, where $(\mathbf{l}, \mathbf{r}) \leftarrow \mathsf{Encode}(\mathbf{s})$ and $(\mathbf{l}', \mathbf{r}') \leftarrow \mathsf{Encode}(\mathbf{s}')$, for any two secrets $\mathbf{s}, \mathbf{s}' \in \mathcal{M}$. We consider a leakage-resilient storage scheme Φ that allows to efficiently store elements from $\mathcal{M} = \mathbb{F}$. It is a variant of a scheme proposed in [9] and based on the inner-product extractor. For some security parameter $n \in \mathbb{N}$ and a finite field $\mathbb{F}$, $\Phi^n_{\mathbb{F}} := (\mathsf{Encode}^n_{\mathbb{F}}, \mathsf{Decode}^n_{\mathbb{F}})$ is defined as follows. Security is proven by [10] with the lemma below.

- Encoding algorithm $\mathsf{Encode}^n_{\mathbb{F}}(\mathbf{s})$: On input the vector $\mathbf{s}$ sample $(\mathbf{l}[2, \ldots, n], \mathbf{r}[2, \ldots, n]) \leftarrow \left(\mathbb{F}^{n-1}\right)^2$ and set $\mathbf{l}[1] \leftarrow \mathbb{F} \setminus \{0\}$ and $\mathbf{r}[1] := \mathbf{l}[1]^{-1} \cdot (\mathbf{s} - \langle (\mathbf{l}[2, \ldots, n], \mathbf{r}[2, \ldots, n]) \rangle)$. The output is $(\mathbf{l}, \mathbf{r})$.
- Decoding algorithm $\mathsf{Decode}^n_{\mathbb{F}}(\mathbf{l}, \mathbf{r})$: On input $(\mathbf{l}, \mathbf{r})$ output $\langle \mathbf{l}, \mathbf{r} \rangle$.

Lemma 4. *Let $n \in \mathbb{N}$ and let $\mathbb{F}$ such that $|\mathbb{F}| = \Omega(n)$. For any $1/2 > \delta > 0, \gamma > 0$ the $\Phi^n_{\mathbb{F}}$ scheme as defined above is (λ, ϵ)-secure, with $\lambda = (1/2 - \delta)n \log |\mathbb{F}| - \log \gamma^{-1} - 1$ and $\epsilon = 2(|\mathbb{F}|^{3/2 - n\delta} + |\mathbb{F}|\,\gamma)$.*

Exponentially Large Fields. In the above construction we have two security parameters: n and $|\mathbb{F}|$ (notice that δ and γ are just artifacts of the lemma and do not influence the construction), which influence the leakage bound λ, statistical closeness parameter ϵ and the complexity of the scheme. So far [9,10], n has been treated as a main security parameter and $\mathbb{F}$ was implicitly assumed to be rather small (operations in $\mathbb{F}$ were assumed to take constant time).

To simplify the exposition we introduce a single security parameter k and assume that n and $|\mathbb{F}|$ are functions of k. We are interested in choice of n and $|\mathbb{F}|$ (as functions of k) such that $\lambda = \Omega(k)$ and ϵ is negligible in k. The instantiation from [10] can be viewed as $n = k$ and $|\mathbb{F}| = \Theta(k)$. The fact that length of the shares n was of the same order as k caused overhead $\Omega(k^4)$.

In this paper we propose a different choice of the parameters n and $|\mathbb{F}|$. Namely, we show that by using shares of the *constant* length and an exponentially big (in terms of k) fields $\mathbb{F}$ we get the same security guarantee and a much better efficiency. Namely, taking $n = 24, |\mathbb{F}| = 2^k, \delta = 1/4, \gamma = 2^{-2k}$ in Lemma 4 gives $\lambda = 4k - 1$ and $\epsilon = \mathcal{O}(2^{-k})$. With a constant n each gadget produced

by the compiler from [10] performs a constant number of operations over the field $\mathbb{F}$. In [31] they show that multiplication in $GF(2^k)$ can be done in time $\mathcal{O}(k \log k \log\log k)$. Hence, setting $\mathbb{F} = GF(2^k)$ and a constant n we get the compiler with the $\mathcal{O}(k \log k \log\log k)$ overhead. This complexity does not depend whether the new refreshing algorithm is used or not, because in this case all shares have constant lengths independent of k.

In the rest of this section we present the new refreshing algorithm. The only assumption about $\mathbb{F}$, which is necessary for its security is $|\mathbb{F}| \geq 4n$.

Non-zero Flavor of Leakage-Resilient Storage. For technical reasons we slightly change the encoding by assuming that the coordinates of $\mathbf{l}$ and $\mathbf{r}$ are all non-zero (e.i. $\mathbf{l}, \mathbf{r} \in (\mathbb{F} \setminus \{0\})^n$. Therefore, $\mathsf{Encode}^n_{\mathbb{F}}$ procedure can easily be modified to generate only vectors with non-zero coordinates. It is enough to use the already presented $\mathsf{Encode}^n_{\mathbb{F}}$ protocol and check at the end if the computed vectors have all coordinates non-zero. If it is not the case, the protocol is restarted with fresh randomness. It does not influence the efficiency of the construction, because a random vector has at least one coordinate equal to zero with a probability at most $1/4$ regardless of n (but assuming $|\mathbb{F}| \geq 4n$). It is easy to see that this modification changes the security of the Leakage-Resilient Storage only by a negligible factor.

4.3 The Leak-Free Component

As in [10], we assume that the players have access to a leak-free component that samples uniformly random pairs of orthogonal vectors. More specifically, we will assume that we have access to a gate $\mathcal{O}'$ that samples a uniformly random vector $((\mathbf{a}, \tilde{\mathbf{a}}), (\mathbf{b}, \tilde{\mathbf{b}})) \in (\mathbb{F}^n)^4$, subject to the constraint that the following three conditions hold: (i)$\langle \mathbf{a}, \mathbf{b} \rangle + \langle \tilde{\mathbf{a}}, \tilde{\mathbf{b}} \rangle = 0$, (ii) $\{\mathbf{a}_i \neq 0\}_{i \in [n]}$ and (iii) $\{\tilde{\mathbf{b}}_i \neq 0\}_{i \in [n]}$.

Note that this gate is different from the gate $\mathcal{O}$ used earlier in [9] that simply samples pairs $(\mathbf{a}, \mathbf{b})$ of orthogonal vectors. It is easy to see, however, that this "new" gate $\mathcal{O}'$ can be "simulated" by the players that have access to $\mathcal{O}$ that samples pairs $(\mathbf{c}, \mathbf{d})$ of orthogonal vectors of length $2n$ each. First, observe that $\mathbf{c} \in \mathbb{F}^{2n}$ can be interpreted as a pair $(\mathbf{a}, \tilde{\mathbf{a}}) \in (\mathbb{F}^n)^2$ (where $\mathbf{a}||\tilde{\mathbf{a}} = \mathbf{c}$), and in the same way $\mathbf{d} \in \mathbb{F}^{2n}$ can be interpreted as a pair $(\mathbf{b}, \tilde{\mathbf{b}}) \in (\mathbb{F}^n)^2$ (where $\mathbf{b}||\tilde{\mathbf{b}} = \mathbf{d}$). By the basic properties of the inner product we get that $\langle \mathbf{a}, \mathbf{b} \rangle + \langle \tilde{\mathbf{a}}, \tilde{\mathbf{b}} \rangle = \langle \mathbf{c}, \mathbf{d} \rangle = 0$. Hence, condition (i) is satisfied. Conditions (ii) and (iii) can be simply verified by the players $P_{\mathbf{l}}$ and $P_{\mathbf{r}}$ respectively. If one of these conditions is not satisfied, then the players sample a fresh pair $(\mathbf{c}, \mathbf{d})$ from $\mathcal{O}$ (it happens only with a constant probability, because $|\mathbb{F}| \geq 4n$).

4.4 Leakage-Resilient Refreshing of LRS

Recall than the pivotal element of the compiler from [10] is the protocol for refreshing the encodings of the secrets encoded with $\Phi^n_{\mathbb{F}}$. Such protocol takes an encoding of the secret and produces a fresh encoding of the same secret. However, we can not just decode the secret and then re-encode it with a fresh randomness,

Protocol $\mathsf{Refresh}^n_{\mathbb{F}}(\mathbf{l}, \mathbf{r})$:

Initial state:
- player $P_{\mathbf{l}}$ holds $\mathbf{l} \in (\mathbb{F} \setminus \{0\})^n$ and player $P_{\mathbf{r}}$ holds $\mathbf{r} \in (\mathbb{F} \setminus \{0\})^n$.

1. Let $((\mathbf{a}, \tilde{\mathbf{a}}), (\mathbf{b}, \tilde{\mathbf{b}})) \leftarrow \mathcal{O}'$ and give $(\mathbf{a}, \tilde{\mathbf{a}})$ to $P_{\mathbf{l}}$ and $(\mathbf{b}, \tilde{\mathbf{b}})$ to $P_{\mathbf{r}}$.

Refreshing the share of $P_{\mathbf{r}}$:

2. $P_{\mathbf{l}}$ computes a vector $\mathbf{v}$ such that $\{\mathbf{v}_i := \mathbf{l}_i^{-1} \cdot \mathbf{a}_i\}_{i \in [n]}$ and sends $\mathbf{v}$ to $P_{\mathbf{r}}$.
3. $P_{\mathbf{r}}$ computes a vector $\mathbf{x}$ such that $\{\mathbf{x}_i := \mathbf{v}_i \cdot \mathbf{b}_i\}_{i \in [n]}$ and sets $\mathbf{r}' := \mathbf{r} + \mathbf{x}$.
4. If there exists i such that $\mathbf{r}'_i = 0$, then the protocol is restarted from the beginning with new vectors sampled from $\mathcal{O}'$.

Refreshing the share of $P_{\mathbf{l}}$:

5. $P_{\mathbf{r}}$ computes a vector $\tilde{\mathbf{v}}$ such that $\{\tilde{\mathbf{v}}_i := \mathbf{r}_i'^{-1} \cdot \tilde{\mathbf{b}}_i\}_{i \in [n]}$ and sends $\tilde{\mathbf{v}}$ to $P_{\mathbf{l}}$.
6. $P_{\mathbf{l}}$ computes a vector $\tilde{\mathbf{x}}$ such that $\{\tilde{\mathbf{x}}_i := \tilde{\mathbf{v}}_i \cdot \tilde{\mathbf{a}}_i\}_{i \in [n]}$ and sets $\mathbf{l}' := \mathbf{l} + \tilde{\mathbf{x}}$.
7. If there exists i such that $\mathbf{l}'_i = 0$, then the protocol is restarted from the beginning with new vectors sampled from $\mathcal{O}'$.

Final state:
- player $P_{\mathbf{l}}$ holds $\mathbf{l}'$ and player $P_{\mathbf{r}}$ holds $\mathbf{r}'$.

Views: The view $\mathsf{view}_{\mathbf{l}}$ of player $P_{\mathbf{l}}$ is $(\mathbf{l}, \mathbf{a}, \mathbf{v}, \tilde{\mathbf{a}}, \tilde{\mathbf{v}})$ and the view $\mathsf{view}_{\mathbf{r}}$ of player $P_{\mathbf{r}}$ is $(\mathbf{r}, \mathbf{b}, \mathbf{v}, \tilde{\mathbf{b}}, \tilde{\mathbf{v}})$.

Fig. 2. Protocol $\mathsf{Refresh}^n_{\mathbb{F}}$. Gate $\mathcal{O}'$ samples random vectors $(\mathbf{a}, \tilde{\mathbf{a}}, \mathbf{b}, \tilde{\mathbf{b}}) \in (\mathbb{F} \setminus \{0\})^n \times \mathbb{F}^n \times \mathbb{F}^n \times (\mathbb{F} \setminus \{0\})^n$ such that $\langle \mathbf{t}a, \mathbf{b} \rangle = -\langle \tilde{\mathbf{a}}, \tilde{\mathbf{b}} \rangle$. Note that the inverses in Steps 2 and 5 always exist, because $\mathbf{l}, \mathbf{r} \in (\mathbb{F} \setminus \{0\})^n$. Steps 4 and 7 guarantee that this condition is preserved under the execution of the protocol $\mathsf{Refresh}^n_{\mathbb{F}}$. The protocol is restarted with a bounded probability regardless of n (but keeping $|\mathbb{F}| \geq 4n$), so it changes the efficiency of the algorithm only by a constant factor.

because an adversary could leak the secret, while it is decoded. Therefore, we need a way to compute a new encoding of a secret without decoding it. The new refreshing protocol performs $\mathcal{O}(n)$ operations over the field $\mathbb{F}$ in comparison to $\Omega(n^2)$ for a protocol from [10].

The refreshing protocol, $\mathsf{Refresh}^n_{\mathbb{F}}$ described in Figure 2 is based on the one proposed in [10] (cf. Section 3), but it is more efficient. The protocol $\mathsf{Refresh}^n_{\mathbb{F}}$ refreshes the secrets encoded with $\Phi^n_{\mathbb{F}}$. $\mathsf{Refresh}^n_{\mathbb{F}}$ is run between two players $P_{\mathbf{l}}$ and $P_{\mathbf{r}}$, which initially hold shares $\mathbf{l}$ and $\mathbf{r}$ in $(\mathbb{F} \setminus \{0\})^n$. At the end of the protocol, $P_{\mathbf{l}}$ holds $\mathbf{l}'$ and $P_{\mathbf{r}}$ holds $\mathbf{r}'$ such that $\langle \mathbf{l}, \mathbf{r} \rangle = \langle \mathbf{l}', \mathbf{r}' \rangle$. The refreshing scheme is presented in Figure 2. The main idea behind this protocol is as follows. Denote $\alpha := \langle \mathbf{a}, \mathbf{b} \rangle (= -\langle \tilde{\mathbf{a}}, \tilde{\mathbf{b}} \rangle)$. Steps 2 and 3 are needed to refresh the share of $P_{\mathbf{r}}$. This is done by generating, with the "help" of $(\mathbf{a}, \mathbf{b})$ (coming from $\mathcal{O}'$) a vector $\mathbf{x}$ such that $\langle \mathbf{l}, \mathbf{x} \rangle = \alpha$.

$\langle \mathbf{l}, \mathbf{x} \rangle = \alpha$ comes from the above summation: $\langle \mathbf{l}, \mathbf{x} \rangle = \sum_{i=1}^n \mathbf{l}_i \mathbf{x}_i = \sum_{i=1}^n \mathbf{l}_i \mathbf{v}_i \mathbf{b}_i = \sum_{i=1}^n \mathbf{l}_i \mathbf{l}_i^{-1} \mathbf{a}_i \mathbf{b}_i = \langle \mathbf{a}, \mathbf{b} \rangle = \alpha$. Then, vector $\mathbf{x}$ is added to the share of $P_{\mathbf{r}}$ by setting (in Step 3) $\mathbf{r}' := \mathbf{r} + \mathbf{x}$. Hence, we get $\langle \mathbf{l}, \mathbf{r}' \rangle = \langle \mathbf{l}, \mathbf{r} \rangle + \langle \mathbf{l}, \mathbf{x} \rangle = \langle \mathbf{l}, \mathbf{r} \rangle + \alpha$. Symmetrically, in Steps 5 and 6 the players refresh the share of $P_{\mathbf{l}}$, by first generating $\tilde{\mathbf{x}}$ such that $\langle \tilde{\mathbf{x}}, \mathbf{r}' \rangle = -\alpha$, and then setting $\mathbf{l}' = \mathbf{l} + \tilde{\mathbf{x}}$. By similar reasoning as before, we get $\langle \mathbf{l}', \mathbf{r}' \rangle = \langle \mathbf{l}, \mathbf{r}' \rangle - \alpha$, which, in turn is equal to $\langle \mathbf{l}, \mathbf{r} \rangle$. Hence, the following lemma is true. The reconstructor for the $\mathsf{Refresh}^n_{\mathbb{F}}$ and its proof can be found in the full version.

Lemma 5 (Soundness). *For every* $\boldsymbol{l}, \boldsymbol{r} \in (\mathbb{F} \setminus \{0\})^n$ *we have that* $\mathsf{Decode}^n_{\mathbb{F}}(\mathsf{Refresh}^n_{\mathbb{F}}(\boldsymbol{l}, \boldsymbol{r})) = \mathsf{Decode}^n_{\mathbb{F}}(\boldsymbol{l}, \boldsymbol{r})$.

References

1. Bitansky, N., Dachman-Soled, D., Lin, H.: Leakage-tolerant computation with input-independent preprocessing. In: Garay, J.A., Gennaro, R. (eds.) CRYPTO 2014, Part II. LNCS, vol. 8617, pp. 146–163. Springer, Heidelberg (2014)
2. Castagnos, G., Renner, S., Zémor, G.: High-order masking by using coding theory and its application to AES. In: Stam, M. (ed.) IMACC 2013. LNCS, vol. 8308, pp. 193–212. Springer, Heidelberg (2013)
3. 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)
4. Cramer, R., Damgård, I., Pastro, V.: On the amortized complexity of zero knowledge protocols for multiplicative relations. In: Smith, A. (ed.) ICITS 2012. LNCS, vol. 7412, pp. 62–79. Springer, Heidelberg (2012)
5. 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)
6. Davi, F., Dziembowski, S., Venturi, D.: Leakage-resilient storage. Cryptology ePrint Archive, Report 2009/399 (2009)
7. Davì, F., Dziembowski, S., Venturi, D.: Leakage-resilient storage. In: Garay, J.A., De Prisco, R. (eds.) SCN 2010. LNCS, vol. 6280, pp. 121–137. Springer, Heidelberg (2010)
8. 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)
9. Dziembowski, S., Faust, S.: Leakage-resilient cryptography from the inner-product extractor. In: Lee, D.H., Wang, X. (eds.) ASIACRYPT 2011. LNCS, vol. 7073, pp. 702–721. Springer, Heidelberg (2011)
10. Dziembowski, S., Faust, S.: Leakage-resilient circuits without computational assumptions. In: Cramer, R. (ed.) TCC 2012. LNCS, vol. 7194, pp. 230–247. Springer, Heidelberg (2012)
11. Dziembowski, S., Pietrzak, K.: Leakage-resilient cryptography. In: Annual IEEE Symposium on Foundations of Computer Science, pp. 293–302 (2008)
12. Faust, S., Rabin, T., Reyzin, L., Tromer, E., Vaikuntanathan, V.: Protecting circuits from leakage: The computationally-bounded and noisy cases. In: Gilbert, H. (ed.) EUROCRYPT 2010. LNCS, vol. 6110, pp. 135–156. Springer, Heidelberg (2010)
13. Franklin, M., Yung, M.: Communication complexity of secure computation (extended abstract). In: STOC, pp. 699–710. ACM, New York (1992)
14. Genkin, D., Pipman, I., Tromer, E.: Get your hands off my laptop: Physical side-channel key-extraction attacks on PCs. In: Batina, L., Robshaw, M. (eds.) CHES 2014. LNCS, vol. 8731, pp. 242–260. Springer, Heidelberg (2014)
15. Genkin, D., Shamir, A., Tromer, E.: RSA key extraction via low-bandwidth acoustic cryptanalysis. In: Garay, J.A., Gennaro, R. (eds.) CRYPTO 2014, Part I. LNCS, vol. 8616, pp. 444–461. Springer, Heidelberg (2014)

16. Goldwasser, S., Rothblum, G.N.: Securing computation against continuous leakage. In: Rabin, T. (ed.) CRYPTO 2010. LNCS, vol. 6223, pp. 59–79. Springer, Heidelberg (2010)
17. Goldwasser, S., Rothblum, G.N.: How to compute in the presence of leakage. Tech. Rep. TR12-010, Electronic Colloquium on Computational Complexity (2012)
18. Grosso, V., Standaert, F.-X., Faust, S.: Masking vs. multiparty computation: How large is the gap for AES? In: Bertoni, G., Coron, J.-S. (eds.) CHES 2013. LNCS, vol. 8086, pp. 400–416. Springer, Heidelberg (2013)
19. 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)
20. Ishai, Y., Sahai, A., Wagner, D.: Private circuits: Securing hardware against probing attacks (2003). Unpublished manuscript ([19] is a revised and abbreviated version)
21. Juma, A., Vahlis, Y.: Protecting cryptographic keys against continual leakage. In: Rabin, T. (ed.) CRYPTO 2010. LNCS, vol. 6223, pp. 41–58. Springer, Heidelberg (2010)
22. 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)
23. Kocher, P.C., Jaffe, J., Jun, B.: Differential power analysis. In: Wiener, M. (ed.) CRYPTO 1999. LNCS, vol. 1666, pp. 388–397. Springer, Heidelberg (1999)
24. Micali, S., Reyzin, L.: Physically observable cryptography. In: Naor, M. (ed.) TCC 2004. LNCS, vol. 2951, pp. 278–296. Springer, Heidelberg (2004)
25. Miles, E., Viola, E.: Shielding circuits with groups. In: Proceedings of the 45th Annual ACM Symposium on Symposium on Theory of Computing, STOC 2013, pp. 251–260. ACM, New York (2013)
26. 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)
27. Prouff, E., Roche, T.: Higher-order glitches free implementation of the AES using secure multi-party computation protocols. In: Preneel, B., Takagi, T. (eds.) CHES 2011. LNCS, vol. 6917, pp. 63–78. Springer, Heidelberg (2011)
28. Quisquater, J.-J., Samyde, D.: ElectroMagnetic analysis (EMA): Measures and counter-measures for smart cards. In: Attali, S., Jensen, T. (eds.) E-smart 2001. LNCS, vol. 2140, pp. 200–210. Springer, Heidelberg (2001)
29. 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)
30. Rothblum, G.N.: How to compute under $\mathcal{AC}^0$ leakage without secure hardware. In: Safavi-Naini, R., Canetti, R. (eds.) CRYPTO 2012. LNCS, vol. 7417, pp. 552–569. Springer, Heidelberg (2012)
31. Schönhage, A.: Schnelle multiplikation von polynomen über körpern der charakteristik 2. Acta Informatica **7**(4), 395–398 (1977)
32. Shamir, A.: How to share a secret. Commun. ACM **22**(11), 612–613 (1979)